Experts and Creativity in Programming

A thesis submitted to Middlesex University in partial fulfilment of the requirements for the degree of Doctor of Professional Studies (Transdisciplinary)

Lisias Vieira Loback
Student number: M00609623
Research Ethics Application no: 1896

Faculty of Business and Law
Middlesex University London

August 2023

Abstract

This research aims to explore the intricate relationships between creativity and programming, encouraging programmers to embrace a more creative approach to their work. It is based on answering the question, "How do experts perceive creativity in programming". There is not much research in the area of creativity and programming, and this work bridges the gap between academia and my community of practice. In order to have a better understanding of the role of creativity in programming, I conducted a qualitative cross-sectional study where I collected information from sixteen expert programmers using semi-structured interviews. The transcripts produced 163,806 words leading to the interpretation of participants' experiences using Applied Thematic Analysis (ATA). Initially, I produced 164 codes organized into sub-themes, originating seven main themes analysed in their relationship with creativity in programming: creativity, programming code, expertise, innovation, learning and development and impact. This study identified the participants' perception of creativity in programming code and inferred that creativity is one of the characteristics that software developers can have that becomes a competitive advantage for modern organizations. In general, this research aims to contribute to the development of creativity in the field of programming and offer practical insights and recommendations to promote creativity, innovation, and excellence in the practice of software development.

Index terms

Creativity, computer programming, expertise, software development, innovation

Table of Contents

Table of Figures

Table of Tables

Glossary of Acronyms

.NET: Cross-platform computer software framework from Microsoft

ADHD: Attention Deficit Hyperactivity Disorder

AI: Artificial Intelligence

ASP: Active Server Pages

ATA: Applied Thematic Analysis

BCS: The Chartered Institute for IT

CERN: Conseil Européen pour la Recherche Nucléaire

CSS: Cascading Style Sheets

DA: Discourse Analysis

DevOps: short for “Development” and “Operations”

DRM: Digital Rights Management

EBP: Evidence-Based Practice

FFH: Fast and Frugal Heuristics community

GDPR: General Data Protection Regulation

HCI: Human-Computer Interaction

HB: Heuristics and Biases community

HTML: HyperText Markup Language

HTTP: HyperText Transfer Protocol

IPA: Interpretative Phenomenological Analysis

IT: Information Technology

LINT: Static program analysis

NDM: Naturalistic Decision Making community

NodeJS: JavaScript runtime environment

PAP: Middlesex University Program Approval Panel

PHP: Hypertext Pre-processor

SFIA: Skills Framework for the Information Age

STEM: Science, Technology, Engineering, and Mathematics

UDI: Universal Document Identifier

URL: Uniform Resource Locator

VB: Visual Basic

VUE: Javascript framework for building interfaces

Acknowledgements

I want to express my gratitude to my wife, Ana and my children, Gustavo and Mathilde, for their support, understanding and encouragement on this journey. Your presence brought joy and perspective to my life during the intense research and writing phases.

I am grateful to my parents, Rev. Ananias and Lília, for the life experiences we shared, the sacrifices they made and the values they instilled in my life. In particular, I wanted to honour the memory of my mother, who did not live to see this project completed.

I extend my appreciation to my supervisor, Professor Brian Sutton, who, through his invaluable guidance, mentoring, and patience during this project, managed to get us to the end.

Finally, I appreciate the academic community with whom I interacted at Middlesex University — London. They all touched me somehow, and their contributions, feedback, and constructive criticism were instrumental in shaping my research trajectory. Thank you!

Part 1: What we know

1. Introduction

After a few years of thinking about programming and creativity, and after many debates with myself, with the community of programmers, academics and professionals, I present my dissertation on the subject.

This is the first part of four in which this research is divided, and in this first part, I will address what is already known about creativity and programming, the context of this research and the knowledge landscape.

In this introduction, I will make a personal presentation so that you can evaluate my content based on what you may learn about me. Then, I review the background of this research outlining my academic and professional path with various details about the companies I worked for and how my learning and development took place. I also talk about the objectives of this research, ending with an overview of the content of the various chapters of this research.

1.1. Personal Introduction

First and foremost, I would like to introduce myself. In the following few paragraphs, you will understand where I came from, where I am going, my values, and who I am. I present curiosities about my path, some of my experiences, and how they shaped my personality. You will also see the importance of family, language, culture and religion in shaping my character and how it affects how I see the world and the connection and influence that all of this has on my research project.

I was born in the Amazon rainforest of Brazil. My parents were evangelical Christian missionaries working with indigenous and riverside communities at the time of my birth. This idea of unfamiliar people going to try to convert or preach a message of salvation to the isolated populations of a country is controversial nowadays. My parents would be “cancelled” immediately on these days of social media. Both my parents were simple people, who had a sense of mission and service to others in a very selfless way, and who always believed that the best way to live would be to give themselves body and soul to a noble cause that they believed in. They did not have the backing of large European or American organisations. This is important because missionaries supported by foreign organisations seemed like aliens who arrived from space with all the comfort, whether in capital, means of transport or housing that had nothing to do with the culture and way of life of the people they wanted to reach. This was not true to my parents’ way of thinking and living. My parents decided to move to the Amazon jungle, arriving from southern Brazil on their own, assuming all responsibility for their support and ministry. Being self-sufficient proved essential as they shouldered their responsibilities and had no one to blame for their adventures and misadventures.

They persevered, showed that one should not give up quickly, and always maintained an independent spirit by constantly paddling against the current and following their principles and values.

In hindsight, I identify myself with a spirituality that emphasises others, with a genuine concern for the well-being of all, respecting the individuality of each one and the way one chooses to live their life. Looking at how my parents took over their choices, I learned about individual responsibility and ownership of my opportunities. Individual responsibility is a well-rooted principle in Christianity (that leads to a certain political conservatism) that reflects my work ethic and the way I appropriate the tasks and obligations entrusted to me. When making my choices, I am solely responsible for making things happen and I am the owner of my future. This is something that I learned later in life, as per Kegan's Order of Consciousness, the concept of self-authorship being the principal transformation of consciousness in adulthood: the ability to "write" your own life (Love and Guthrie, 1999).

My entire family is a combination of immigrants. On my father's side, they are mostly Germans, farmers with imagination and determination who left Germany in the mid-nineteenth century and emigrated to southern Brazil, where most European emigration took place. They brought with them a Protestant spirituality with a sound ethnic identity and a strong will to win. In the subsequent generations, the family was assimilated by the Brazilian "culture", losing the German language, culture and religion. On my mother's side, on the one hand, we have a mixture of the same ancestors as my father (German). On the other hand, the principal emigration to Brazil, which is Portuguese (Brazil is a former colony of Portugal). Above all, the Portuguese are loyal and hard workers. They have been adventurers for centuries; more than 20% (Agência Lusa, 2019) of the population has emigrated overseas and lives outside the country. They are peaceful people who do not get into trouble wherever they are and maintain a solid connection to their family and homeland. Furthermore, one can see this connection through gastronomy — all of life revolves around meals, and if necessary, people will spend hours socialising while they eat. This creates strong bonds between people and lasting and loyal friendships. The way I deal with work relationships is primarily influenced by this culture of conviviality, bonding, sharing, especially during meals, and interest in genuine relationships. When the Covid-19 pandemic occurred in 2020 and 2021, most people started working from home. What I missed the most was socialising with co-workers, sharing conversations, ideas, and the moments during meals. I feel energised when I am in the company of other people, which breaks a little the (false?) stereotype of programmers, who close in on themselves, preferring to spend time away from other people. Maybe I am a programmer outside the moulds expected by most. Later I will elaborate on my path to becoming a professional programmer (on page 18).

My life is an adventure. No wonder that after my parents started a family, their life was an adventure, and consequently, my life was an adventure from the moment I was born. Until I was eight years old, I had no electricity at home, no tap water, and no bathroom inside the house. I played in the dirt yard with improvised toys; chased chickens, ducks and pigs; jumped in mud puddles; tore my pants and legs on the barbed wire in the pastures; bathed in the streams; I scraped my knees on the bluffs; I continued to play in the rain and drew on the wet sand. I did everything that would be normal for kids my age. I rode horses, drank milk straight from the cows, drank fresh water from wells, had organic food without knowing and was happy about it all. I thought that my life was conventional and that everyone lived the same way I did. It turned out to be completely untrue. The reality I found in the rest of the country was completely different. When I visited relatives in the "civilization", I discovered electricity and electric lights, where you did not need to add oil nor a wick to light them – you just had to turn on the electric switch. Furthermore, I really liked the electrical switch! I remember standing on one side of the aisle, switching on the lights, and my brother would be on the other side switching it off, both fascinated by the miracle happening in front of us. Even to sleep, we asked to sleep with the lights on such was the enchantment. It was new, and it was magic – the button produced the charm. This was not simply curiosity to know how things work or how technically and scientifically they are created. However, it falls within the spectrum of what Opdal (2001, p. 331) calls wonder – a prevalent feeling of perplexity and humility in the face of not only what is at stake but the new possibilities that open up when one establishes contact with unimaginable objects and concepts in the world in which one lived. Opdal argues that a prerequisite for developing creative people with a keen critical sense is precisely this desire to know more and understand better – beyond what is taught – this fascination for the unknown and the enchantment for the unexpected. I have had my dose of wonder at various times in my life, and I am fascinated with the results I have achieved.

I still think about how a boy who was born in the Amazon rainforest, living in a world totally different from the Western world, managed to study at university, work for large companies in the technological area, and contribute with unprecedented research to the enhancement of scientific knowledge in the area of programming and creativity.

My parents largely influenced the development of my values. I feel I have inherited their spirit of adventure and an insatiable curiosity to see, experience and learn new things. When I think about adventure, I always conceive of an almost accidental, unexpected circumstance or event whose outcome is uncertain. This uncertainty involves risk, and possibly some danger, but it brings immense pleasure to undergo. I was particularly inspired by the way my parents lived what they believed in, particularly concerning family, spirituality, honesty, integrity, and, amongst other things, living a purposeful life. Not only living or surviving but finding meaning in everything you set out to do. In opposition to my parents' way of life, I have challenged their over-simplistic approach to life, lack of ambition and initiative, and their perception of the world and the people around them. Over time, I acquired business acuity, greater confidence in my abilities and what I could achieve and longstanding determination, all of which were alien to them and their lifestyle.

When I turned twelve years old, I had already lived in eight cities in three different countries, spread over two continents. I grew up in Portugal. For me, it was the third country I have lived in (after some time in Paraguay), and perhaps from where I received the most significant cultural influence that shaped me into what I am today. It is the place that most influenced me, maybe because it was where I did all my schooling until the end of my first university degree, or maybe it was the culture into which I got married. It is a special place because, by choice, I decided to assimilate it, just like my ancestors did when they moved from Europe to Brazil. I took the opposite route to theirs: I came from South America to Europe, and I had no problems connecting to Portugal and its culture in the same way that I have a connection to Brazil. I am very proud of my roots, but I am also very proud to be Portuguese. Incorporating different cultures is also part of the curiosity to experiment and discover new things – this constant desire to know more, live different experiences and create lasting memories. This is reflected in preserving a mother tongue other than English – and language is an essential element in cultural dissemination and ethnic identity. By expressing myself in another language, I have a different identity and my personality changes. Montuori (2008, p. 13) talks about his realization that thinking in different languages made him see the world differently and make different decisions. Myself when expressing in Portuguese, for example, I transform into a funnier, more open person and in my opinion, much more fun. Words flow differently, and thoughts fly at a different speed than when I am communicating in English. As opposed to Portuguese, when I communicate in English, I am much more considerate, I always think much more before I say something, and I am very polite. In short, I am uninteresting and boring.

Different languages allow for different interactions with people. In my work environment, this cultural sensitivity helps in the relationship with everyone, especially with ethnic minorities, given that computer programming workers tend to come from different nationalities. Like no one else, I understand the challenges of emigration, and the need for integration in the workplace, while preserving different aspects of people’s culture, their values, and the different way they see the world. The very structure of language creates a way of thinking and seeing things, and what we need is to broaden our perspectives to realize that other cultures do things differently and also have different beliefs from ours (Bolton, 2018).

One of the most pivotal junctures in my life occurred when I relocated from Portugal to London, United Kingdom, in 2006, after living in Portugal for twenty-four years. Certain events carry the power to reshape our lives and shake out our careers, the way we think, and who we are. This change was one of them. Moving from Portugal to London was not my inaugural experience of moving countries, but this marked the first instance where I, as an adult, made such a decision independently. Previously, when I moved to Paraguay and from Brazil to Portugal, I was a child and just tagged along with my parents. This was a big one, and it was a challenging process requiring careful consideration. Moving to another country for work had multiple and substantial implications for me and my family. The critical requirement for success in this endeavour was believing that life would improve and everything would improve.

In these circumstances, there is a complete reset in life, marked by setting new targets and changing your dreams and aspirations. The change leads to a strong desire to be the best in everything you do and fight hard for your goals. You are willing to change your assumptions, career, way you work and code, learn and develop, and most importantly, you are open to change. If I did not want change, it would have been better not to leave where I was. This is the beauty of emigration and an excellent advantage for any country that wants to receive immigrants from other places. Immigrants arrive with ambition, determination, grit and much desire to work hard and improve their lives. In this new mindset, changes are not only expected but embraced. The flexibility I developed from previous life experiences was crucial in this new journey. This decade was marked by change, both in my professional career and personal life, with the growth of my family and further career developments (Loback, 2018).

1.1.1. Tri-cultural: Brazil, Portugal and United Kingdom

Moving to the UK would have been much more difficult if I had not had a long cross-cultural experience. While you are growing up, there is a dichotomy between your home culture and the country where you are living. In a way, adults realise this and unconsciously or consciously fight against the union of their family culture with the foreign country they are living in. There is a desire to maintain some type of connection with the origins. This manifests itself mainly in the food, language and regular visits to the country of origin. From a very early age, I realised that I am a citizen of the world. The nuances of immigration, the creation of roots in different places and continents, the understanding and acceptance of different cultures and ways of thinking, and the certainty that territorial borders impede human development.

On the one hand, you feel that you belong to three different cultures, but on the other hand, you are not fully accepted in any of them. The fact that you have a different perception of each of them makes you never entirely of either. You assimilate the best of each one of them, and you create your very own world. The accent becomes a differentiating factor in this argument. While I was in Portugal, the place where I grew up, people would say that I was Brazilian; when I was in Brazil, the place where I was born, they called me Portuguese, and when I'm in the United Kingdom, a country of my third nationality, they ask me with a subtle delicacy, where I am from originally! The answer is always complex because it is difficult to explain in a single word what my origins are. It is impossible to name one of the countries and say that I am from there. Because I am not from one country and because I do not want to define myself that way.

"The exposure to different cultures early on gave me a sense of the plasticity of human beings, of the creativity in cultures, but also of the way in which one could become trapped in one's own creations without realizing they were, in fact, one's own creations…" — Alfonso Montuori (2008, p. 12)

In the previous paragraphs, it was possible to understand where I came from and how I became what I am today as a person and researcher. These opening paragraphs provided personal arguments, evidence and insights into my life, which inform how I will develop my research and produce my outcomes. It is obvious that how I became the person I am today has influenced how I see my life, work, studies and research.

1.2. Background of the research

I am a Software Engineer. I build websites and applications from scratch, from elementary web pages to multi-million views interfaces for the most renowned companies in the world[1]. I use the most suited competencies and skills to create, code, test and deliver high-quality products by applying the technologies and functionalities that are the accepted best practice at the time. These are the right skills for the job, not solely because I chose to develop them personally but because the stakeholders and employers prefer them (Loback, 2018).

I started designing for the web employing the knowledge learnt from my degree in Communication Design in the early years of the internet (Loback, 2018). At first, it was about transferring experiences from other media types, such as knowledge of layout and composition, colours, texture, contrast, typography, imagery and photography, and aesthetics (Beaird, 2014, p. 1, 53, 89, 123 and 159). However, with time it involved developing and acquiring new knowledge. Chapman (2006, p. x), in his complete introduction to web design, described the knowledge a person should have to be an effective web designer. It was necessary to know the digital technology without which the World Wide Web could not exist, including the markup that supports web pages, stylesheets, graphics and images, servers and databases, as well as scripts that add dynamism and better experience from the users' point of view. It was all very technical, and it looked like anyone could do it. The bar to enter the profession is very low. This assertion will be developed further down on page 132 in the research when I talk about expertise, but for now, let's say that anyone can learn how to code and apply a number of rules and concepts using any programming language. In my practice, in addition to the technical factors, I have always been interested in improving the design of the pages so

[1] To find a comprehensive list of projects and companies, check my profile on linkedin.com/in/loback.

that they are accessible to everyone and effectively communicate their message in a creative way. I acquired skills in terms of planning (with a hint of user experience), design (everything related to the user interface) and updating websites (including the user performance). All of this correlates with Human-computer Interaction (HCI), whose study existed before the advent of the internet. I found inspiration in Jakob Nielsen in my early years of creating for this medium with his studies on usability[2] and the web user experience (Nielsen and Loranger, 2006; Nielsen, 2000). My burgeoning interest in technology led to a master's degree in Information Technology, several courses in programming and a desire to continuously self-develop (Loback, 2018).

My informal training in terms of programming involved self-learning. Self-learning was necessary here because I chose another area of study at the university that was much more creative as opposed to studying computer science. At least that was my perception of programming at the time – a boring, calculating area related to maths and logical thinking, and not creative at all. In the same way that children learn a new language by interacting with their own language through their parents and those around them, programming began as a field of experimentation for me. Building from the knowledge of the basic syntax, as the challenges at work became more complex, knowledge of the language also became more complex, and it evolved to become the knowledge I have today. The hardest part was learning the first programming language. In my case, it was Visual Basic, adapted for use on the internet. The language is now known as "Classic ASP" (Active Server Pages). Different programming languages use a different idiomatic mode, but it is possible to transfer knowledge from one language to another. Next, there needs to be an idiomatic adaptation because of the way people programmes also adapt to different types of language. It works like the various languages in the world (the natural language) – we may know the translation of words, but the way sentences are constructed is different. To become an expert in a programming language is necessary to know the idiomatic of the language – and this takes time to achieve. Moreover, the best way is by reading code written in that language. Furthermore, that is what I did and continue to do over time.

My formal training in Information Technology was already held in London and opened my horizons to strategic thinking in terms of Information Technology, but above all to the importance of research. I would not be writing this thesis today if I had not experienced during my master's degree a fascination for research and experimentation. My thesis was based on a program and a system that I created to contend with closed systems of Digital Rights Management – DRM. At this time in 2012, several companies were starting to sell digital books that imposed many restrictive measures on users. While the interests of publishers and authors had to be safeguarded, traditional DRMs placed a burden on readers that made it difficult to read books in different media, the need to install proprietary software and difficulty to use a product that had

[2] "Usability is a quality attribute relating to how easy something is to use. ... how quickly people can learn to use something ... how efficient ...memorable ... and how much users like using it." — Nielsen, J. and Loranger, H. (2006, p. xvi)

been legally purchased. My solution was based on what came to be called “Social DRM”, where part of the non-private and confidential information was inserted into digital books (e-pub) to hold the person who bought them responsible for possible misuse. The information was randomly added, but if necessary, it could be extracted, and the buyer could be identified through access to the software that generated the book. The idea was to make buyers responsible for the use they could make of the books by completely unlocking the acquired books. The solution was beneficial to both publishers and users. In my final project, I developed the complete planning and programming system, wrote about the theory and the development process, and created an end-to-end solution. With this, I developed my programming skills and my critical and reflective thought, which are helpful to me in my practice and research nowadays.

Doing programming reminds me of the sensations I felt when I discovered electricity as a child – the wonders of how the switch can control the flow of electricity and create spectacles. I think about how you can produce everything from a screen with nothing in it. Programming had brought this ecstasy of creating and developing things that did not exist before. Montuori (2008, p. 10) talks about this joy, passion, wonder and excitement when he mentions the power of the creative inquiry process. It seems almost magical. There is a portion of the divine in it that makes it feel exceptional.

Web design refers to aesthetic and usability aspects and aggregates knowledge from centuries of graphic design convention. Unsurprisingly, it reclaimed several concepts and terms from that convention, such as composition and layout; texture, proportion, and patterns; the psychology of colour; typography with letterforms, fonts and typefaces; and imagery (Beaird and George, 2014). Ethan Marcotte (2011), in his book ‘Responsive Web Design’, suggests that the significant distinction between traditional graphic design work and web design is that the product of web design is not tangible as a book/object, becoming ethereal and does not reach the user in the same way that the creator idealised (Marcotte, 2011). When it is published online, it is at the mercy of who sees it – the user. Many different variables affect how the user sees and interacts with the applications. Some of them are the number of screen colours, font descriptions, and the browser window size and format (desktop, mobile or tablet). With this multitude of media, formats and sizes, the designer and programmer need to work together in order to create an adaptable and flexible product without forfeiting the main features. A designer creates the layout and various website elements using design software (for example, Adobe Photoshop). On the other hand, a programmer uses the design created and builds a functioning website using markup languages (for example, HTML and CSS) and other script programming languages (the likes of .NET, JavaScript, PHP and Java) so that images produced by designers gain “life” through interactivity and functionality. As explained before, programming languages are a set of simple rules, which, when used in software programs, introduce their own rules and which, in turn, become complex systems (Loback, 2018). The complexity will depend on different factors since our primary concern as programmers is to keep our programs under control. (Haverbeke, 2014). One of the aspects to be studied in this research is the complexity of computer systems.

Working with this hybrid interaction was an impressive experience: I left behind the design part of my work but kept the creativity. All that I wanted was to be able to express my creativity in different ways, mainly through design, communication and multimedia, and by doing that, the internet had become my platform (Loback, 2018). The change arrived when the industry quickly developed and moved towards specialisation by adopting different skillsets. I still use many transferable skills and competencies from the "design" part of my life today.

The possibility of doing a doctorate arose when I reached a ceiling point in my career regarding technical knowledge and the ability to do business consulting with that knowledge. I had the notion that I had reached a certain level of expertise, and in collaboration with colleagues, I understood that this knowledge acquired in the job market had the potential to be shared and be part of academia. I and the co-researchers have already worked at the doctorate level but have not had the opportunity to demonstrate academically. This is how this project was born.

1.3. Aim and Objectives

The question that arises is, where is creativity amid programming? I would like to know how a programmer with a creative edge can nurture their creative inclination in situations where they are willing to take calculated risks and assume responsibility for their creations. They should be able to develop that creative side. I find that there are processes, best practices, and code structures that are rigid in their genesis that inadvertently stifle creativity, damaging the holistic evolution of the practitioners and contributing to missed opportunities in terms of code exploration and timely delivery to the stakeholders (Loback, 2018). In contrast, I advocate for practitioners to actively foster an environment that embraces diverse ideas and fosters innovative thinking. Initiating this shift represents a positive start on this research journey, and I am enthusiastic about contributing to creating that space on the policies, the frameworks, and the code (Glass, 2006). I want to build an online platform that facilitates collaboration (co-development) between various programmers interested in exploring creative solutions. This will be a space for sharing ideas, code snippets, and best practices related to creativity in programming.

The long and complex history of failed IT systems, late delivery and high-profile outages has led to the adoption of a risk-averse and formulaic approach in programming. To this day, this has influenced the way that programmers today have approached software development. Some known cases include the Ariane V rocket (Lynch, 2017), which on the maiden voyage self-destructed forty seconds after launch due to a problem in the guidance system software that was inherited from previous versions without having been upgraded to take into account the increase in speed and manoeuvrability. A more recent example happened in 2018 and 2019 with planes that crashed (Boeing 737 MAX) due in part to a glitch in software designed to automatically adjust the plane's angle of attack to prevent stalling (Travis, 2019). The investigation concluded that the software had not been adequately tested or certified. This case has been cited on "The New York Times" (Kitroeff et al., 2019), "The Guardian" (Jolly, 2020), "The Verge" (Campbell, 2019) and many others as an example of the dangers of prioritising time-to-market and cost savings over safety and thorough testing and leads programmers, in mission-critical situations, to prefer to choose already tested and established patterns and frameworks. However, only using what is tested and known means that there is no innovation and technological advancement, and for that, I intend to make several suggestions based on the collected data so that experts can have more freedom in their use of code, removing some limitations that have been imposed on programmers and bet heavily on dedicated time for experimentation and risk management. I intend to bring case studies and document best practices in my workplace so that they can serve as a guide for other programmers and their teams.

Professionals must grasp the nature of creative talent and how to harness it to foster innovation. Managers ought to recognize that affording programmers greater autonomy unleashes creativity and nurtures innovation. Within this project, there is an opportunity to cultivate novel approaches to coding by embracing concepts like complexity and chaos theory. This can lead to the emergence of fresh code patterns and frameworks. I would like to create workshops and training sessions in my workplace and my field of influence to encourage creative thinking and problem-solving.

My objective encompasses enhancing how my practice is perceived while simultaneously exerting influence within the professional community of this field. The focal point of my research endeavour is to construct an academic framework that facilitates the creative advancement of programming code, drawing from tangible insights of industry practitioners. As a participant in this research, my perspective and viewpoints hold significance. However, I also emphasize the importance of amplifying the voices and perspectives of fellow co-researchers engaged in this project. While some might share my viewpoint, it is equally possible that their outlooks diverge, offering a spectrum of insights.

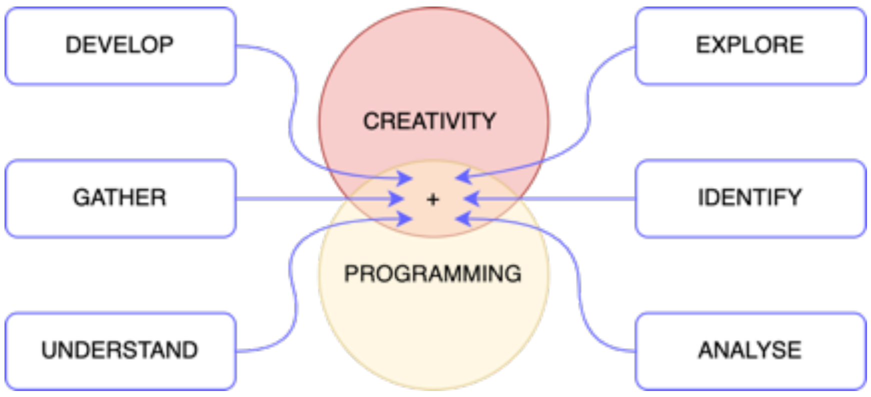


Figure 1: Aim and Objectives of the research

Below there is a list with a summary of the aims and objectives of this research:

- **Explore** the relationships between creativity and programming and investigate how programmers with a more creative view of programming can be encouraged to develop this aspect of themselves;
- **Identify** limitations imposed on programmers through rigid processes, best practices and code structures and propose ways to create new spaces for innovative thinking and the development of different ideas;
- **Analyse** the impact that the risk-averse and formulaic approach to programming has had on the quality and safety of applications created by programmers and identify ways to balance the need for safety and the need for innovation and technological advancements;
- **Understand** how creative talent can be utilised to create more innovation in terms of programming and identify the factors managers must consider to generate creativity and innovation among programmers;
- **Develop** new code patterns and frameworks linked to new ways of thinking about complexity and chaos theory; **Develop** an academic foundation for the expansion of creativity in programming based on real-world professional experiences and be an influence in the area (Loback, 2018);
- **Gather** co-researchers' perceptions on creativity and add their perspectives to the research findings;

1.4. Justification

There was an opposition between academic work and the practice community in general. Schön (1994, p. viii) explains this disagreement as follows:

> "They feed into the university's familiar dichotomy between the 'hard' knowledge of science and scholarship and the 'soft' knowledge of artistry and unvar-nished opinion." (Schön, 1994)

This dichotomy comes from the undervaluation of the practical knowledge of those in the professional environment in relation to the knowledge produced by academia. Schön (1994, p. viii) suggests that academia looks at the knowledge produced through real-world experiences within the practice in communities. The suppression of this community knowledge can lead to a lack of understanding and appreciation for practitioners' valuable insights and expertise. As a result, this project is an attempt to prioritise creative and experiential knowledge over merely academic knowledge in order to increase the contribution of professionals in less studied areas, such as the discussion of this project in terms of creativity and programming in order to reduce the gap between the two vectors of knowledge. This explains why the setting of this research is under a Professional Doctorate because we are looking at a practical component of my professional work to create knowledge that constructs a bridge between my profession and the academy.

A more detailed justification would examine the creativity component and its relation to the code. Why is this topic relevant? Programmers can be much more than task executors in the programming world. They have the ability to create and communicate ideas and concepts through programming based on their experiences and knowledge acquired over time. Due to the risk inherent in critical programming systems and other factors that limit the range of code usage, way of programming, and other administrative pressures, programmers' freedom to innovate is curtailed right from the start, undermining all the creativity and innovation that could eventually be produced if there was an understanding of the benefits of allowing expert programmers some freedom for experimentation. This will be my contribution to the scientific community in general and programmers in particular.

This work is a path to a new strand of myself. It is a new version and a new story for my life that started when I decided to embark on this project and will continue for many years after it ends.

1.5. Overview

This thesis is organized into four different parts. As mentioned above, in the first part, I will address what is already known about creativity and programming, the context of this research, and the knowledge landscape. The second part looks at what we do not know and what we want to discover. More specifically, it looks at the methodology of this research and the theory surrounding all the project activities that were developed. In the third part, we have the analysis made to the data and from which the implications for practice were highlighted. The last part (fourth) contains the impact of this research, the reflection, references and appendices.

Within the parts, the thesis was organized into chapters that address different topics in relation to the project and the arguments that I will develop during this project. The structure was designed to organise the various parts that made up this project in independent blocks but interconnected in context.

The structural organisation of the thesis did not follow a standard convention of having a more condensed number of chapters. Initially, the thesis was organised into five or six chapters, but as I added more text to the chapters, I realised that with the complexity and depth of my divisions and sub-divisions, the organisation became unintelligible. Accustomed to dealing with big data and organising and structuring information coherently and understandably in my career in computing, I chose to depart from the most commonly accepted norm and promote to the highest level in the hierarchy of the thesis the themes that emerged from my data analysis adding six more chapters to the top level. The effect on the thesis was to immediately contain the themes in smaller chapters in more focused segments that facilitate navigation and help digest the specific themes encapsulated in their own space. Each chapter also serves as a lens on the topic that I propose to develop in this research, allowing it to be better examined, analysed and synthesised. This lower granularity, while decreasing the visual depth or organisation, increases the profoundness of the theme itself, allowing greater flexibility in how readers intend to engage with the content due to greater ease of navigation. Moreover, finally, bringing my computational experience back, the fact that there are more chapters that can be read and analysed separately also brings to mind the computer modularisation I am already used to having in everything I build.

Part 1: What we know

You have just read the introduction to this chapter and now have a more intimate knowledge of my personal background and the objectives of this project (the ‘Introduction’ chapter).

The context is defined in chapter 2, where I address issues related to my professional environment and my relationship with other interlocutors. This chapter aims to understand how my profession is intrinsically linked with research and affects my judgement in the decisions I came to make in this project.

Chapter 3 is a literature review, which was developed throughout the project, and not just at the beginning, as is usually the case. It included the scope definition and was developed as the data analysis defined the themes.

Part 2: What we want to know

Chapter 4 discusses the methodology used and the rationale for using Ground Theory associated with practice-led research.

In chapter 5, I wrote a critical analysis of all project phases from the beginning to its conclusion. I explain the whole process from deciding on the research questions, the demography of the co-researchers, how the data was collected, transcribing, preparing the data for analysis, thematic analysis and writing up of the thesis.

Part 3: Analysis

Part three discusses the themes that emerged in the data analysis. In this part, I develop the arguments that link creativity to code production; I dissect the role of experts in the development of creativity; and I develop some ideas based on innovation and experimentation; the role of my co-researchers trajectory in the development of creativity and the impact of research.

Part 4: Remarks

Chapter **Error! Reference source not found.** contains the findings and recommendations of this research. It also has a reflection on the project based on what was written in chapter 5. It also has the personal and professional impact of this research.

Finally, we have the references (chapter 15) and attachments (chapter 16).

1.6. Summary

I started this chapter with a personal introduction in order that the reader might better understand my origins, values, who I am, how my life experiences shaped my personality, how I see the world, and the importance of all of this in my research. From my personal experiences, I moved into the background of this research. Here, I began to explore my profession and how I learned my trade from my days as a designer to my entry into the world of programming. The Aims and Objectives were defined by considering the constraints I encountered in my professional career, which led me to want to use a more creative and innovative side within the production of lines of code. I justify this research with the dichotomy between academic teaching and the world of professions, emphasising the use of creativity in programming to bridge the gap between these two areas of knowledge. I end the chapter by providing an overview of the various chapters on this research and what I will propose in each of them.

2. Context

In a qualitative analysis, it is essential to pay attention to the context in which the research is inserted (Bazeley, 2013). Knowing the context is crucial not only for understanding the research but also for the transfer of acquired knowledge to a new situation and the consequent application of this theory in other settings (p. 81). In this chapter, I will write about the professional context in which I find myself inserted. I will start with the work setting and my relationship with employers and colleagues; then, I will explain the recruitment process, as it is crucial to understand how it affects the work context. I will also address the power factors concerning the ability to exercise creativity within the professional context, as well as covid and the fact that working from home changed the course of this research. Finally, I will address the relationship with the researchers and the power structure there.

2.1. Work setting

I work as a contractor. I am a team member, but I am not an employee of the company that hires me. I am called to fill some gaps in hands-on-deck development. I easily integrate into the teams where I am about to intervene, but I am aware that the time on the team will be limited between three months and two years. Joining the team as a contractor is not the same as joining as a permanent employee. If a priori, the contract is short-term, the context of the work is not so broad since I will be focused on a specific area or an approaching deadline. Information that will be shared will be for immediate resolution and may not involve the broader context of the entire project. Although it depends a lot on the company's culture, in some places where I worked, even daily updates were separated from permanent employees, so you do not waste time in meetings whose context is irrelevant. This creates some constraints on your work in terms of creativity and implementation of ideas. Without a general knowledge of the project, the solutions you eventually present may not fit into the general framework, leading to you losing your decision-making and creation autonomy due to the lack of a broader context. You always depend on others to evaluate your solutions, and you feel your expertise is not being used to the fullest.

I produce programming code during the first hours since the start of a contract. This is only possible because there is a match between my abilities and the team's needs. Whoever hires me knows what they want and need for the team, and during the selection process, everything is narrowed down to the specifics of the tasks to be performed. Because of that and the experience accumulated in so many other projects, I can start and confidently produce code straight away. Herein lies the difference between a contractor and a permanent employee: how quickly you can assimilate the project and quickly produce the desired output.

2.1.1. Hiring process

The recruitment process is always based on the most popular technologies used the most at the time. There is always a variation over time. My programming language of choice is Javascript, but the selection questions are usually based on the libraries in use at the moment and the entire tech stack and operations of the development teams. Nowadays, a programmer needs to know the programming syntax in his language and the process necessary to create, test, and deploy what the team produces to live environments. Besides that, we have to squeeze here the libraries we use based on the specific programming language.

JavaScript libraries are code files containing a series of features that allow programmers to perform some tasks within an application or system. It allows programmers to reuse code that other programmers have done for certain functions without reinventing the wheel. This is a good rationale for using libraries and frameworks. Instead of wasting time creating code for a particular feature, a programmer can "borrow" the code from someone who has dedicated themselves wholeheartedly to creating an optimal solution to a problem. This programmer does not have his head on another project, but he thought about all aspects of that solution: reusability, performance, delivery, design, etc. Moreover, using a library is usually the best option when we want to develop an application in record time. There is a differentiation between libraries and frameworks in the sense that frameworks provide a whole skeleton on which the programmer can base its application. The programmer will use the organisation, the frame, and how the framework thought about the project from its adoption. What is gained in terms of structure and development speed is lost in terms of code freedom since it is necessary to follow the rules and conventions adopted by the framework. As a result, libraries allow greater code and organisation freedom than frameworks usually do. For this marriage to work, it is necessary to accept the concepts and ideas of the framework fully; otherwise, the programmer will always be complaining about what the framework does not do instead of quickly creating solutions based on the best use of the framework.

What used to be done by the Operations Team in the past, such as deploying code to live or managing servers, nowadays is merged into DevOps (which is short for "Development" and "Operations"). DevOps combines the practices of both areas so that applications are developed and deployed quickly. This brings advantages for organisations but greatly increases the level of knowledge that a programmer must have about the entire process, from writing a line of code until it reaches the people who will use and benefit from the application. It is almost impossible to know everything and be an expert in the whole process, not to mention the fact that you have to keep up to date on everything.

Because of all of this, the recruiting process is sometimes a little frustrating. Sometimes it seems that notwithstanding much you know about a subject, it is never enough to keep you in business. Moreover, all the years of accumulated experience are worthless because the combination of what is needed for a given gig is miles away from what you have valid experience and expertise.

The great advantage of being a contractor and not being part of the company as a permanent employee is the simplicity of the working relationship and the team itself. The relationship is very straightforward and task-based. There is no prospect of career progression, as you joined the team to perform a particular task or set of tasks, and when those tasks are over, you move to another project. We can analyse this from several angles. One of the angles might be the lack of commitment to the future of code in terms of reusability or maintainability since you are sure you will not be in business for the long term. The other angle is the type of assessment of your work that is very binary: either you can or cannot carry out the tasks you were given with a degree of expertise. If the programmer is an expert, he receives his reward, which is a paycheck at the end of the month, and if he cannot prove that he is an expert, then his contract will not be renewed, and he will have to look for another project.

2.1.2. Creativity

The degree of creativity in terms of code will depend on several factors. The influence you can have depends on what stage of the project you join the team. Suppose one enters the project at an embryonic stage. In that case, one can be creative in terms of architecture and the future direction one wants to give the project, both in terms of organisation and in terms of frameworks and libraries. If you enter a project at a more advanced stage, your code has to blend with the lines of code that other programmers produce, and that also limits your creativity and the way you solve problems. There is no time for experimentation at a more advanced stage; the focus is on speed and completion, not creativity, to solve a given problem. The influence for using creativity and for “doing my way” comes with time and the ability to demonstrate the reliability of a particular solution over others. Time is usually short. On the other hand, there are also few opportunities to exercise leadership when you are a contractor. Usually, companies prefer to promote their permanent employees, which seems natural. Even if they offer a promotion opportunity to a contractor, negotiations become difficult because, typically, contracts appear at a rate of at least 30% valued. Is a programmer willing to lower his rate to get a promotion? This is questionable.

2.1.3. The way I operate

I particularly enjoy working as a contractor. I get bored very quickly, and I need new challenges in my life. The boredom may come from the fact that many tasks assigned to me are tasks that no other programmer wants to perform in the company, and for that, they call a contractor or because doing repetitive tasks is not my forte. A paycheck can be an excellent motivation to work, but everything has a limit. Because I already have many years of experience and have worked for many prestigious brands and projects, I have been getting exciting contracts that challenge my imagination and creativity and keep me entertained for a long time. One such project was the creation of the "Helios Platform", a Renault-Nissan joint venture, where five of its global car brands (Dacia, Datsun, Infinity, Nissan and Renault) that sell millions of cars worldwide, with hundreds of variants between them have joined together to create three-hundred distinct websites hosted on a single digital platform (O'Leary, 2015). The Helios platform was the largest single implementation of the Adobe Experience Manager (the engine behind it) at the time. The idea was fascinating and consisted of providing customer experiences from the different brands with minimal maintenance. All the brands shared the same template, with brand-specific content and different styling. In short, each site's look and feel and content are different based on the same code/template. I was there when they launched the first website in India, and I was there when the hundredth website went live, and I felt great satisfaction pointing out the different components that I built on the various websites. There were about twenty teams in the development area: two frontend developers, two backend developers, a project manager, a designer, and a tester. We all worked round-the-clock in agile teams to create and deliver hundreds of components that comprise the template that all the brands were using (Dingsøyr et al., 2012).

2.2. General Context

All my experience and the years I have spent as a programmer are being put into this research. This work would not be possible without my personal development and professional development over the years. I can see in the trajectory of this constant struggle between a more creative side, linked to Fine Arts, willing to explore new paths, have new experiences and venture into the unknown with a more pragmatic side, controlling the risks associated with programming, which wants projects to follow the rules and reach a successful conclusion. It is a constant swing in which the latter wins most of the time. Regardless of that, I still have a fixation on seeing what we can do corporately and individually to bring out the full potential of the teams we work with, extracting creativity and the desire for innovation, which is probably hidden somewhere inside some compartment that the idealistic programmer once stored. Maybe I will find that day – the moment when the adventure ends, and we surrender to the possibility of being mere executive manufacturing machines of projects that do not bring anything creative and that do not advance the programming field or bring anything new or a line of innovation.

2.2.1. Covid and working from home

The lockdowns due to the Covid-21 pandemic in 2020 and 2021 resulted in drastic changes in the landscape of development teams in organisations. Because of my personality, I did not notice any difference in how

I fit into the teams. I have a unique way of dealing with people, integrating into teams and resolving differences and conflicts. Technology permits good communication, and there was no disruption to continuing teamwork, but we lost the ability to have random interactions – the informal conversations in the corridors or the kitchen – crucial to developing creativity. Sawyer (2012, p. 256) argues that these informal mixtures, which he calls the "collaborative webs", can develop creativity by allowing the exchange of diverse perspectives, bringing with it a flow of ideas in a more open and informal environment. Sawyer uses several examples of how informal mixing has led to significant creative developments in various fields, such as science and the arts. Montuori (2012, no page) also talks about the emergence of ideas and creativity in these unstructured and informal environments that go against the way traditional academic thinking develops. These interactions are an essential cross-pollination to exchange ideas and be the catalyst for creativity in the workplace. Moreover, in that sense, it was a waste of time. I am sure that organisations will have the opportunity to compensate for this lost time and channel their creativity into other activities and work formats. In terms of delivery, there were no changes as productivity increased. Companies have no trouble believing that remote workers are doing their job well because they have access to the code produced. Furthermore, the subject is binary concerning deliveries: either the job is done or not.

2.2.2. Language setting

When writing this thesis, I use a language that is not my mother tongue. Although an effort has been made to free it from grammatical errors, the creative process sometimes is being done in a different language and being translated simultaneously, sometimes in my mind, sometimes in writing, sometimes mixed and at the same time. I am trying to make sense of different linguistic realities either in real time or asynchronously. My reality, initially, did not have an English linguistic background; hence there are nuances and ideas that I had to adapt before writing them in the final text. I hope you read this not because my grammar is perfect or my prose is eloquent but because my content resonates with you, and you can take some wisdom from it (Van Essen, 2014).

> "... translation is essentially a boundary crossing between two cultures." — Nelofer Halai (Bazeley, 2013)

2.3. Co-researchers

The context of the co-researchers in this study was reported in chapter 5.2 below, and some characteristics are described in detail and include various demographic data, such as years of experience, organisational role, the level and the area of study, the type of developer, the type of employment and the gender.

2.3.1. Equal co-research

The idea that I have of the relationship with the co-researchers is one of equality. The fact that I do not exercise any hierarchical power over the participants helps that the ideas produced are a sincere reflection of what they think about the various topics that we talked about. There was no measure of strength or small power here that could sway the answers here. I did not exercise hierarchical power over any of the co-researchers, which helped create the trust and rapport necessary to conduct this research (Braun and Clarke, 2022, p. 18). I feel like no one had anything to prove to me. They already had their careers established, so there was no need to withhold information. Best of all, we were co-workers, producing our best within a cross-functional team. When I interviewed some of the people who were leaders in the projects I worked on, we were no longer working together at the time of the interview. The moments we spent in the interview were a pleasant reflection of their careers, how they saw programming and what they thought could be different. We are together on this journey, searching for better solutions, learning more, and perhaps, influencing the world in which we operate.

2.3.2. Power dynamics

In a way, I occupy a position that can be seen as marginal in relation to my profession since my academic trajectory did not include Computer Science as the main university course. Of the people I interviewed, 60% studied Computer Science in their higher education courses, and this was how they chose to enter this profession. In my view, and for the purpose of this research, I consider myself privileged to have followed a different path. I accept that not everyone will be a supporter of this position, but I think we need different perspectives and worldviews so that our teams of developers can make a difference. I also feel privileged to have chosen to do this project. It is an admirable process for the level of commitment, the focus, the abnegation and the effort. Not only to do the project but to look at my path, compare it with the co-researchers paths and ideas and draw my conclusions.

It is interesting to think that in the early days of Information Technology, almost all practitioners came from other disciplines, where Arts and Humanities were rich in providing new professionals. BCS – The Chartered Institute for IT has no data regarding the demographics of changes in routes into programming from other areas, but in a report on age diversity (BCS, 2021a), one of the key findings was that older IT specialists were less likely to have a higher education. And younger specialists are more likely to have taken an IT-related course compared to those over 50 years old (8% versus 14%). On the one hand, this proves that the older ones either did not have university degrees or, if they did, came from other study areas. Nowadays, there are other sources of recruitment to the profession – University courses in computing and programming and grassroots professionals whose main skills were acquired by courses provided by IT training companies. Looking into the future, it would be interesting to see if we have fewer and fewer professionals from the Arts and Humanities within the field of programming and software development, and this will have to be left for a more specific study.

2.4. Summary

In this section, I looked at the context in which my research was developed. I scrutinised my work setting and the nature of my work environment. We visited the recruitment process and how I was hired to be integrated into the development teams. We also glimpsed here the intricacies and complexity associated with the production of computer code and the role of creativity in software generation. I ended up looking at the power relationships between myself and my co-researchers.

3. Knowledge Landscape

The purpose of the knowledge review was to align the actual knowledge produced on the themes with the output of my research. I was looking for something other than a gap in literature because my project starts in the real world, and I am responsible for bringing it to the Academy. I started by understanding that some things in my professional world are intriguing to me and that I want to shed some light on this research project. Consequently, I tried to join the discourse of some themes to understand the state of research on these same themes to understand my problem better. As the practitioner that I am, especially in a Professional Doctorate project, knowledge in the area can help me better understand my problem. The events and literature produced in recent years can give me some ideas about my project and lead to some solutions.

The literature review process started when the project was presented at the Middlesex University PAP – Program Approval Panel back in April 2018. A pre-analysis of the available literature was carried out, focusing on what I wanted to reveal throughout the project. The purpose was not to analyse everything available in the academy on the themes but to choose the most relevant ones that had been developed. This preliminary analysis provided an insight into the broader context of the area. Contextualisation allowed me to see my research within an already existing body of knowledge. Ridley (2012, p. 6) explains the context as follows:

> "... your research is a small piece in a complicated jigsaw puzzle; … Your reader therefore needs to know about the whole jigsaw puzzle and not simply the shade and shape of your particular piece." (Ridley, 2012)

The Review of Knowledge and Information was initially divided into the themes that arise from the proposed research questions: complexity, creativity, and innovation. They followed the line of thought that I had at the time and formed the initial basis of the project (Loback, 2018).

Snyder (2019, p. 334) proposed three approaches to the literature review depending on the typical purpose of the review. A systematic review, which is best suited for quantitative analysis, allows for the comparison and synthesis of evidence by analysing the entire literature. When the purpose is not to cover everything that was published about a theme or topic, an integrative analysis is proposed. This permits the combination of new perspectives to allow the creation of new theoretical models. Although I wanted to combine different ideas about the themes, I would still be looking to identify patterns, disagreements or relationships between the different topics (Snyder, 2019).

The methodological option was for narrative or semi-systematic analysis since we could not examine all the literature produced about the themes. This literature review method allows us to analyse how knowledge about specific subjects has evolved over time, their implications for the topic being studied, and the link with the questions for which answers are sought (Snyder, 2019). When opting for the narrative proposal, one way to make the Literature Review is to use thematic analysis – which a variation will also be used to analyse the data from the semi-structured interviews.

The possible problems with this approach are that there is not as much documentation and guidelines for its implementation as there are for systematic analysis.

3.1. Scope

The purpose of this review is to present the broader context of what I set out to investigate (Ridley, 2012). It will appear in the form of a description of what has already been done in this area and create an image of this proposed project. I wanted to join the conversation.

My criteria were not only to search for the most recent publications. Computer Science is a relatively new field of research if one compares it with Physics or Chemistry, and there is still much to investigate and significant gaps in the knowledge. We are still in the first or second generation of programmers, with a significant number of scientific works waiting to be published. The main idea would be to compare how the area of programming, specifically how code production for computers, relates to the themes defined for the literature review. The themes were defined by analysing the data collected from the conversations in this project. A priori, although I had an idea of what I hoped to develop, there were no certainties, and at the same time that the themes were developed during the data analysis, the literature review was growing and taking the form that is presented below.

3.2. Introduction

It all started with the reading of three paragraphs in a book about Javascript programming in 2015 (Haverbeke, 2014). The book was about a modern introduction to programming. Right at the beginning, the author decided to explain his ideas about basic programming principles that went beyond the book's central theme, which was, of course, Javascript. In addition to writing about how, as programmers, we create parallel worlds of things that did not previously exist (Haverbeke, 2014), the author criticised how most programmers only use specific and commonly known techniques in their programs to overcome complexity. He dissents from the programmers that refuse to explore an infinite number of solutions and techniques available to them, favouring, instead of trying new ideas, to use what is predictable and widely used and tested by the community. This aversion results from the creation of strict rules, whose creators call them "best practices", which determine how programs should exist and how the programming language should be applied. Hence, a hard line of followers of the so-called "best practices" would label "bad programmers" all other individuals who do not follow the rules created by them (Haverbeke, 2014). Haverbeke argues against this creativity reduction, leaving a cautious message stating that "there will always be new challenges and territories" and that programmers who refuse to undertake the programming adventure will stagnate and become "bored with their craft" (p. 3).

Haverbeke's book (2014, 2edn) is part of my personal library and has had a massive impact on my career and research. The author later put the book online and published more editions thanks to financial endorsements. When I compared the last version with the one I had as a paperback, I noticed that the author changed a few sentences and paragraphs in the book's introduction, seeming to notice a change of thought – which is perfectly normal when publishing new editions. I sent an email to ask if he had changed his mind regarding his reflection on the second edition (Loback, 2022). Haverbeke kindly replied to me:

> "As I made a pass over the text for the next edition, I edited some stuff that looked awkward or that I thought could be smoothed, and there's not necessarily a change in philosophy behind every edit. Best, Marijn" (Haverbeke, 2022)

This idea of an adventure, reflected in the work of Haverbeke, resonates with this research because it presupposes something unexpected and unknown. If it is unknown, it is exciting, and if it is exciting, it needs exploration. The question that arises is: is this adventure what the programmers are looking towards?

My frustration comes from the general idea that creativity and Information Technology do not mix. A dichotomy is created between the two areas, alluding to the fact that the two areas are exclusive, and there is no place for an intersection somewhere in the middle or perhaps at the ends of it if we think more creatively. This idea comes from the myth that our brains have a right section linked to creative potential and a left part used for logic, rational and analytical thinking (Jarrett, 2015). Following this reasoning, we could define programming as purely analytical without any ounce of creativity. Jarrett refutes this idea that we use the left and right sections of the brain separately and assumes that the never dies myth has an intrinsic metaphorical appeal. In healthy citizens, the two sides of the brain are entirely intertwined. We could even try to box repetitive tasks on one side and put creative ones on the other, but the reality of life is more complex than that, and there are many ways for people to be creative (Jarrett, 2012).

Current knowledge regarding creativity and innovation is product-centric. We look at a pen, a table, a song or a painting, and we can talk about creativity in design, music, functionality or simply aesthetics. The creation process can also be creative, but we tend to look at the end result. So, it becomes difficult to look at programming code and find it creative. It is easier to look at the design of the application and the interface with users or even the program's features and see creativity. Pikkarainen (2011, p. 3) explains that "software can be used to support innovation processes while, at the same time, being an instrument for innovation". There is no mention of the code itself, and they are thinking more about the features and the processes. Software avows that because it is "malleable" – delivered in increments, based on previous versions, and "intangible" – it is not a physical product.

3.3. Creativity

We do not know how to define creativity and often think our actions are not creative (Selinger, 2004; Nussbaum, 2013). There is an accumulated knowledge of creativity, and we are closer to explaining the subject (Sawyer, 2012), although there is not enough documentation on the different contexts of creativity yet. The research focuses on the expressions of creativity "highly valued" in our societies, like fine arts, basic science, or classical music, leaving behind applied science, graphic art, technology and the violinist, among others! Creativity does not appear naturally, although it is, by definition, something new and different that we all already have within us, everything that is needed to be creative (Sawyer, 2013).

Tom and David Kelley are some of the most well-known authors in terms of creativity, and they defend a belief that everyone is creative, with creativity being a natural part of how people think and act from childhood to adulthood (Kelley and Kelley, 2015). The Kelleys write about creative confidence and appeal to our ability to create change in the world around us through the use of our imagination, bearing in mind that, in the business world, creativity manifests itself in the form of innovation.

> "Creativity comes into play wherever you have the opportunity to generate new ideas, solutions, or approaches." (Kelley and Kelley, 2015)

Montuori and Purser (1995, p. 71), on the other hand, have a vision that creativity must be studied within a more interdisciplinary context in which historical, ecological, and systemic aspects must be considered taking into account cultural and differences of gender, against the idea of a "solitary creative genius" who appears with a work “out of nowhere” in a total abstraction of the environment that surrounds (p. 74) “emerging fully formed and with no need for diligent study or the development of a craft” (p. 78). This past “natural talent” view of creativity has been deconstructed by Montuori (1995, p. 78), Csikszentmihalyi (2009, p. 1) and Sawyer (2013, p. 2), leading to the idea that creativity comes in small steps, some insight and small changes that grow incrementally.

> “Creativity did not descend like a bolt of lightning that lit up the world in a single, brilliant flash. It came in tiny steps, bits of insight, and incremental changes.” (Sawyer, 2013)

In any way you use creativity, and in any situation or profession, creativity will bring new and better solutions, directing to looking beyond current norms and producing innovative products. Kelley and Kelley (2015, p. 25) developed an existing methodology called “design thinking” that relies on the human ability to be intuitive while recognizing patterns and building ideas that are functional and make sense. The suggestion is to find a balance between intuition, on the one hand, and rational and analytical thinking, on the other, when choosing the way to manage a career or an organisation. The main attribute of design thinking is to get ideas out into the world as quickly as possible to get feedback as early as possible. To carry out this task, they used “rapid prototyping”, which consisted of taking an idea and building the first version in a few hours or days (Sawyer, 2013).

3.3.1. Environment

Creativity needs to be nurtured. There is a need to create a flexible environment where experimentation is emphasised and unorthodox is challenged in order to break the rules (Smith, 2010). In the right environment, the impact of critical experts can be increased and optimised (Pikkarainen, 2011). Companies and managers are responsible for nurturing and developing an environment conducive to creativity (Catmull, 2014). Susan Cain also discusses this idea in her best-seller book ‘Quiet’ (2012, p. 71), where she is very critical of the environmental settings that do not foster creativity. She suggests that transposing successful online collaborative settings to the real-world open-plan office harms the creative process. It has “been found to reduce productivity and impair memory”, and the “way forward [...] is not to stop collaborating face-to-face but to refine the way we do it” by changing the working conditions that allow practitioners to focus on their creative process. The same is valid for innovation. According to Miller, we cannot change the software developers, but we can change the environment where they work to make innovation more attractive (Miller and Wedell-Wedellsborg, 2013). For Catmull, the environment must be full of challenges, uncertainty and unresolved problems that allow “the creation of new technologies, skills, and knowledge”, cited by Smith and Paquette (2010, p. 120). She also states that the flexibility of the setting allows for experimentation that defies the norm, which creativity requires. Kelley (2016, ebook without page number) argues that truly creative individuals are not “few and far between” but the opposite, and they can be developed if one adopts a culture that encourages creativity with tolerance to the occasional failure, embracing risks and novel ideas (Loback, 2018).

3.3.2. Creatives are disruptors

Keith Sawyer (2013) argues that creatives are disruptors because they have the ability to see things in different and new ways (p. 74-77). One path used is combining and recombining ideas from different fields, domains and disciplines to create something new and unexpected (p. 102, 163). Sawyer also notes that creatives are willing to take risks and explore and try new things to have a breakthrough in new ideas and discoveries (p. 222).

3.3.3. Creative as a choice

Miller and Wedell-Wedellsborg (2013, p. 85) argue that creativity is not like a talent that people are born with but a skill that can be cultivated and developed through deliberate practice and effort. They suggest that people and companies can choose to be creative by adopting specific mindsets and behaviours (p. 9). For example, the book suggests that individuals can choose to be curious and open-minded, actively seek out new experiences and perspectives, challenge assumptions, and experiment with new ideas. Throughout the book, they argue that by making a conscious effort to be more creative, organisations and individuals can open new possibilities and drive innovation (Miller and Wedell-Wedellsborg (2013).

3.3.4. Limitations on creativity

The tendency of developers is to “reject creativity in favour of predictability and conventionality” (Nussbaum, 2013). Catmull (2014) is a creative person who created ‘Toy Story’ from Pixar Studios, the first entirely computer-animated film has the same views on ideas and thoughts about creativity in programming that is transpiring in this project, namely the existence of many elements blocking creativity. He says that we must take the necessary measures to limit the blockages (p. xv), especially those arising from uncertainty and instability (p. xvi), and that as managers, we must loosen control, not tighten it by trusting the people who work with us, always being prepared to take risks (p. xvi). Although his thinking is about creativity that drives innovation in a company or institution, we can infer some ideas into programming (Loback, 2018). This was precisely what Glass (2006, n.p.) did in his book on software creativity when he described creativity as being the key to solving complex problems. It explores the dichotomy between flexibility and discipline in software creation, linking the latter to creativity.

3.4. Professional Practice

A discipline can be described as an individual science that studies certain subjects independently of other disciplines, and a profession is based on the application of scientific knowledge of a particular discipline, thus being a mediator between theory and practice (Mahler et al., 2014).

There is a strong connection between the study of professional practice and the application of scientific knowledge in professions. What I hope is that professionals can look at their practice in a methodical way and manage to extract knowledge that can come to the academy and be applied again to their work. The study of professional practice is also concerned with understanding how scientific knowledge is applied and used back in practice. This involves examining the ways in which professionals develop, learn and apply their skills, knowledge and experience. It is also concerned with the social, cultural and ethical aspects of professionals at work.

There is an ongoing debate about the nature and role of knowledge in our contemporary society. Transdisciplinarity becomes that mediator between academia and professional practice, as it seeks to create a dialogue and collaboration between different professional practices to address complex social problems that require different perspectives. As a result, the work on creativity by Montuori and Purser (1995, p. 70) suggests the need for an approach that encompasses the context in which creativity is expressed, such as history, ecology, systems, cultural differences and gender, among which he highlights the interdisciplinarity. Within the transdisciplinarity, Montuori (2008, p. 24) refers to introducing the researcher in the inquiry process as a central issue (Augsburg, 2014). In complementarity, citing Csikszentmihalyi, he expresses that the study of creativity cannot be done by isolating individuals and their practices from their social and historical sides, given that creativity is the result of interdisciplinarity between the individual, the social system and the creative domain (Montuori and Purser, 1995). Csikszentmihalyi (2009, p. 23) argues that creativity is an interactive process in that we only know if an idea is new or not through the interaction of our thoughts with other external entities in order to pass a social evaluation. Creative ideas, then, cease to be an individual phenomenon and become part of a broader context.

Interdisciplinarity can help improve the effectiveness of professional practice by allowing an understanding of the issues to be resolved by promoting dialogue and collaboration between different professional practices.

> "... my own position is that there is knowledge outside of the academy and there's knowledge within the academy, and they're not necessarily the same thing. I think there is a welcome trend in trying to build an interconnection between those two different realms of difference so that we do not

> silence or revere one kind of knowledge above the other but expand knowledge for both." — Maguire (Gibbs and Beavis, 2020)

We can see that there are several types of knowledge, coming from different sources. On the one hand we have knowledge that is validated by academia through formal research and on the other hand we have the knowledge that arises outside the academic environment, from sources such as personal experience, community knowledge and other forms of practical knowledge. As a result and looking for a link between these two paradigms, we find transdisciplinarity as an approach or methodology for creating knowledge or for solving specific problems (Augsburg, 2014). This is related to the 'Mode 2' knowledge defined in the chapter dedicated to methodology (section 4.1 below), where I refer both to the aspect of work-based research and also to 'transdisciplinarity' that draw on different theoretical and practical perspectives methods to solve problems; being "socially distributed, application-oriented, transdisciplinary, and subject to multiple accountabilities" linked to research and innovation (Nowotny, Scott and Gibbons, 2003).

People leaving universities today are starting a professional life where there are new ways of working that did not exist when they started their academic course, and they have to be prepared to deal with situations where they have to transfer knowledge from different fields and different professional practices (Gibbs and Beavis, 2020). These are new skills that need to be part of the degree nowadays.

3.4.1. Expertise

After looking at concepts like creativity and innovation (in the previous paragraphs), I would like to explore the idea of expertise and intuition as expressions of creativity in relation to programming. First, let us understand the epistemology of expertise and its connection with insights and intuition. Expertise is part of a wider field of study. One of the areas of the study of expertise is the association with the Evidence-Based Practice (EBP). Expertise is part of the multiple sources of evidence used in EBP to solve problems in an informed way.

> "When we apply EBP to management decisions, the four main sources of evidence used are scientific literature, organizational data, stakeholders' concerns, and professional expertise." (Briner, 2019)

This demonstrates the importance that is given to expertise and its expression through intuition within a broader field of evidence-based practice.

In philosophical terms, it could be said that expertise refers to the accumulation of knowledge or information that someone has, together with their "skill or performance ability" (Goldman, 2018), in a direct link between what the person knows and what people can do or produce. This idea is in line with The Five-Stage Model of Skills Acquisition from Dreyfus and Dreyfus (2005), which focused on the subjects' experiences using a phenomenological approach. Dreyfus defined the stages as Novice, Advanced Beginner, Competence, Proficiency, and Expertise, the highest stage (Dreyfus and Dreyfus, 2005). Novices learn and act "like a computer following a program", a system of a programmed set of rules (Dreyfus and Dreyfus, 2005), in contrast to experts, who do not need to follow the rules, as they have "holistic understanding" and go "beyond existing interpretations". Dreyfus goes further in his analysis as he adds the concept of intuition to the accumulation of knowledge and experience. Dall'Alba and Sandberg (2006) are critical of this type of knowledge model that focuses on fixed sequences of knowledge accumulation, arguing that by placing the focus on stages, models lose track of the skill being developed, which has everything to do with how development in the profession is seen (Dall'Alba, 2006). How professionals view their practice defines how they understand the degree of expertise they are in and their own position where expertise is concerned.

> "More specifically, the knowledge and skills that professionals use in performing their work depend on their embodied understanding of the practice in question. The professionals' way of understanding their practice forms and organises their knowledge and skills into a particular form of professional skill". (Dall'Alba, 2006)

The knowledge of your practice depends on your level of understanding of your own system of meaning creation (Love and Guthrie, 1999). When mentioning Kegan's Order of Consciousness Theory, they refer to the fact that it is essential to understand what a person thinks and how to construct their own experiences (p. 67). Moreover, these experiences and the level of consciousness itself have an evolutionary nature, indicating a change that produces shifts in perspective or mental complexity, often termed vertical development, as opposed to the horizontal knowledge accumulation model, such as Dreyfus' model of knowledge acquisition. Expertise is more than just knowing stuff. It is also about understanding how to develop yourself, learning how to learn, and acting to improve the world around you, with values entrenched in your own identity. This vertical development allows people to view and capture new perspectives from the same knowledge base.

> "One of the central features of this new way of thinking is an orientation toward contradiction and paradox. Rather than feeling a need to choose between the two poles in a paradox, an individual recognizes the contradiction and orients toward the relationship between the poles. Therefore, contradictions do not threaten the system or necessarily need to be reconciled." — Love and Guthrie (1999, p. 75)

Dreyfus' idea was to be able to create a model that could maintain a very high level of expertise while preventing the risks of things going astray. In contrast to these layers of knowledge, Anders Ericsson (1993) argues that experience, or years of practice, are not enough to take a person to a higher level of expertise. Simple practice does not work. It is necessary to structure the practice. To go beyond expertise, be innovative, and make a contribution to the domain, a person must first acquire the knowledge and skills of an expert in the domain and then, through deliberate practice, over at least ten years, reach another higher level of expertise. This deliberate practice includes activities specifically designed to improve the current level of performance (Ericsson et al., 1993). This idea does not exist without its critics, as Macnamara, Hambrick and Oswald (2014, p. 1608) carried out a meta-study on how people acquired their skills and became experts in what they do. Their study found that deliberate practice only explains 26% of the variation in performance for games, 21% for music, 18% for sports, 4% for education, and less than 1% for deliberate practice in professions. It is not that deliberate practice is not important, but it is not as important as the argument raised by Ericsson, although other studies are also needed to discover other predictors. When studying the development of expertise, it is necessary to consider multiple factors since the relationship between deliberate practice and the amount of training and expertise can be much more complex than initially imagined.

To better understand expertise, think of a task such as driving a car. If you have been driving for many years, you do not need any mental effort to know the order in which you start the car, when to change gears, or how to park. This information is already acquired. The novice will take more time to collect the information, start the car, and move on the road. An expert may be going on a road that was never driven by him before and still be able to quickly adapt to the weather, speed, light and other road conditions. Expertise is based on the response that is given immediately, without reflection (Dreyfus and Dreyfus, 2005).

There is a social aspect to expertise associated with reputational validation, so much so that there is a need for recognition of expertise in the professional environment.

> "A person is an expert only if she/he has a reputation for being one." (Goldman, 2018)

Goldman disagrees with this statement, and it makes sense to think that one does not need a reputation to be an expert. In this informational view, perhaps the quantity and quality of the accumulated information make the person an expert, even if a layperson disagrees.

When we look at the work carried out by several specialists in expertise, it is not very difficult to conclude that expertise has a fundamental contribution to obtaining positive results in the world of the professions, such as making effective decisions and producing high performance at work (Dreyfus and Dreyfus, 2005; Dane, 2010). Erik Dane (2010, p. 579) produced a paper where he devised an idea that experts are inflexible in some aspects of their practice. His conclusions suggested that as expertise is acquired, at the same time, some flexibility is lost, mainly in domains where knowledge stability is part of the domain schemas. He called this phenomenon "cognitive entrenchment". The argument is that cognitive entrenchment restricts the experts' ability to find the solutions that would be optimal, to adapt to new situations and to generate radical creative ideas. The way out of this entrenchment lies in two factors: on the one hand, being involved in work environments where knowledge is not stable and there is an inherent dynamism in expertise. IT professionals and programmers are usually more immune to entrenchment because their knowledge domain is dynamic, and this project is also about that. Furthermore, on the other hand, the other moderating factor of entrenchment is the extent to which individuals focus their attention on tasks outside their domain of expertise. This matter will be discussed in section 3.5.4 below ("Talent Development").

Through experience, the professional practitioner reaches a level of expertise and knowledge that allows one to recognize patterns in the situations one faces. As a result of this pattern recognition, this professional is better equipped to handle these new situations efficiently and effectively.

> "A professional practitioner is a specialist who encounters certain types of situations again and again." (Schön, 1994)

As mentioned earlier, anyone can become a programmer. Although open to everyone, becoming a programming expert is only achieved with time and dedication by a few.

3.4.2. Intuition

Experts rely almost entirely on intuition, and intuition is one of the most important characteristics when referring to expertise (Dreyfus and Dreyfus, 2005). There are several communities for studying intuitive decision-making, and Klein (2015, p. 164) defines three. Each of the groups understands what intuition is differently:

> "The Naturalistic Decision Making (NDM) community defines intuition as based on large numbers of patterns gained through experience, resulting in different forms of tacit knowledge. This view contrasts with Fast and Frugal Heuristics (FFH) researchers, who view intuition in terms of general purpose heuristics. The NDM view also differs from the Heuristics and Biases (HB) community, which sees intuitions as a source of bias and error." — Gary Klein (2015, p. 164)

I often compare expertise and intuition to the accumulation of case studies of situations and lived experiences housed as patterns of knowledge and insights. This fits Klein's (2013, p. 166) thinking in the naturalist view (NDM). It discusses several possible ways to strengthen this naturalistic view to make more reliable decisions by providing a solid experiential foundation that allows for the creation of better tacit knowledge. In hindsight, Klein's goal is to convince the FFH and HB community to use the NDM approach to solve their problems by providing them with strategies. When the expert faces a new hurdle or has to solve a problem that he has never experienced before, he combines different experiences, mixes and matches and comes up with a new solution to be presented and tested that was not there beforehand (Schön, 1994). True intuition happens when the expert recognises familiar elements (patterns) in a new situation and acts appropriately to solve a new problem (Kahneman, 2011).

> "It is our capacity to see unfamiliar situations as familiar ones, and to do in the former as we have done in the latter, that en-ables us to bring our past experience to bear on the unique case. It is our capacity to see-as and do-as that allows us to have a feel for problems that do not fit existing rules."
> — Donald Schön (1994, p. 140)

Intuition in the decision-making process happens when patterns that have been previously learned are used (Klein, 2013). Klein also makes a distinction between intuition and insights, referring to insights as to discovering new patterns. Moreover, experiences are acquired and internalised the more interactions happen, and the more often patterns are recollected. These patterns constitute knowledge and depend on years of experience to build numerous patterns.

Harry Collins (2010) defends the idea of tacit knowledge as knowledge that is not explained. It follows the thought of Polanyi (2009), who introduces this idea of unexplained knowledge in which learning and discovery are difficult to explain. This tacit dimension is a type of intuition as if it were a leap of faith that takes place through prior knowledge and lived experiences. The most used example is about "cycling", in which, in order to learn, we were not taught the rules, and it becomes difficult to explain to others how it works. However, knowledge exists from the moment we learn because we can balance ourselves on top of the bike, even if we do not recognize how we learned or recollect all the rules. One of Collins' (2010, p. 83) contributions was identifying and classifying three types of tacit knowledge. Relational tacit knowledge based on relationships between people is very difficult to codify or teach but essential for successful collaboration (p. 85). Next is tacit somatic knowledge based on physical experience, which is challenging to articulate in words but can be learned through practice and experience (p. 99). Finally, we have collective tacit knowledge, which refers to the type of knowledge shared by a group of people, such as a community of practice or science. This type of knowledge is usually implicit and may not be explicit in the literature, but it is understood by the community to which it belongs (p. 119).

Following the intuition communities of thought from Klein (2013), we have the third current – HB, which argues that intuition is associated with the failure to use an evidence-based approach. Jolley (2020), in his work for health professionals, develops the idea that intuition is like a considered assumption used when there is not enough information available at the time about a given situation. Obviously, in the health professional milieu, there are severe consequences for the use of intuition and instead, the search for missing information is encouraged to make evidence-based decisions. The concept is the same in the professional computer programming environment since when an expert uses intuition, he is applying past experiences in a new context. The counterargument about the validity of using intuition as a method of decision-making is also valid and defended by Gigerenzer (2008, digital page not available), in which intuition, when performed by experts, produces the same result as a more considered and evidence-based approach.

If we think about the rule-based system of a novice, then we realise that one needs to make an effort to understand all the rules and constraints of the task to be performed. For an expert, all the rules are there. However, they are collated because they have been assimilated, and the expert can recognize patterns and do what they did last time.

A "proficient performer", when faced with the need to create a solution to a problem, can see what needs to be done and decide how to perform it. Through a wide range of alternatives, he can understand the best way to achieve an optimal result. Dreyfus concludes that the best way to distinguish an expert from other people is the ability to execute these subtle and refined choices, to distinguish situations that require an immediate reaction in a certain way from alternatives that would not have the desired effect.

3.5. Innovation

Creativity is different from innovation. There is a difference between innovation and creativity, although the result is heavily interlinked. They are not the same thing. Creativity is finding something new (Cropley, 2015) with some value attached to it in the context of the creation (Sternberg, 1999). Innovation is finding new ways to combine existing things to achieve new, unanticipated outcomes. Pikkarainen (2010) explains that the starting point of innovation is a creative idea. Then organisations have to make an effort to turn that idea into an excellent product or service. Keith Sawyer asserts that innovation is widely accepted by scholars as being the "successful implementation (by organisations) of creative ideas" or products that individuals generate (Sawyer, 2012). The focus of innovation studies typically goes to "technology innovation", "technical innovation", or "product innovation" (Hamel, 2007, p. 35), and there is no published material that talks about innovation associated with the lines of code produced for computers – at least when I started this study. An article published by Selinger (2004, p. 48) in a magazine tries to lift the veil concerning programming and creativity, arguing that there is creativity in the activity of programming, but that, apparently, it is difficult to consider that there is creativity when the whole process is built on top of other technologies, not from a blank slate.

Clayton Christensen (2016, p. 225), in his book published over 25 years ago, explores how stable and well-managed companies have failed to adapt to new technologies that have appeared on the market in the meantime, and which, in the end, due to the lack of innovation, led to their downfall (p. 61). Christensen explores the problems faced by companies in implementing innovative ideas, which usually involve taking risks and pursuing ideas that are still unfounded or profitable (p. 228). Ultimately, he highlights the importance given to innovation in generating growth and helping to remain competitive.

Programmers innovate daily. Every day they sit at their computers to create a new feature, solve a problem or change code. They are developing something that was not there the day before. It is like an artist who starts his work with a white canvas and gradually generates colours, shadows, and contrasts, making sense until they become a creation of art.

Programmers are creators out of nothing. Moreover, it is almost magical to see the programs take shape. Any programming language produces a base framework that allows code creation to carry out the most diverse and complex applications. It works in the same way as English or Portuguese, the Human Interface Language (Loback, 2018). There is grammar, a series of rules and conventions, which, if minimally followed, can produce any text and explain the most complex concepts. There are no limits to what a text can produce or explain. Programming languages follow the same principles. They have the basic elements of the environment, the fundamental syntax, data types, loops, variables, basic operators, keywords, decision-making, numbers, characters, arrays, functions, strings, input/output, and so on (Loback, 2018). Furthermore, from this, one can build the most complex systems (Haverbeke, 2014).

Companies see innovation as a “group” of people/heroes confronting day-to-day operations bringing new ideas and concepts (Govindarajan and Trimble, 2010). Organisations are ideal to “deliver consistent and reliable performance” – not to innovate. In this discernment, innovation must be managed together with all its processes (Pikkarainen, 2011).

> “It’s easy to see how business innovation is propelled by formulating the right question, staying open to new cues, and focusing on the right problem.” (Sawyer, 2013)

Successful innovation is one that not only comes up with new ideas but also asks the right questions to identify which problems will be solved through an open mind to new information and ideas, keeping the focus on the problem to be solved. By doing so, companies can create innovative solutions to problems and experience growth that will lead to success.

Many studies about creativity and innovation derive from technology usage, and it is commonly accepted that creativity brings innovation, being a differentiator factor in companies. I am of the belief that as code innovation flourishes, the growth in creative capacity will bring significant advantages to companies and organisations (Loback, 2018).

Another dimension of innovation is brought by Gary Hamel (2007, p. 32), where he identifies the four levels of innovation, which can be incremental improvements or disruptive innovations that can change the way we think and operate at a business level. At a lower level, we have "operational innovation", which involves improving existing processes and systems to increase efficiency, reduce costs and increase quality. In a second stratum, we have "innovation in products and services" that involve development and improvements that provide additional value to customers. Thirdly, there is "strategic innovation", capable of developing new business models, entering new markets and creating new propositions. Finally, and at the top, we have "management innovation", which involves new practices, systems and structures that allow an organisation to innovate and compete more efficiently. Competitive advantage typically comes through innovations that work simultaneously on two or more levels of these strata. When mentioning disruptive innovators, I cannot fail to mention Christensen (2016, p. 143), who points to the fact that ordinarily successful disruptors look to the needs of people and markets that do not yet exist. Looking at people who are not consumers of a particular product instead of looking for solutions only for those who are already consumers, in particular, requires looking to the future while acknowledging uncertainties. And this is the true disruptive innovation.

3.5.1. The culture of Innovation

Different authors explain that it is necessary to create and encourage a flexible environment that allows for experimentation against established norms for innovation in companies (Euchner, 2017; Smith and Paquette, 2010). All the support that can be given to people who want to create new things within an organisation can be defined as a culture of innovation. The goal of developing a culture of innovation in an organisation is to make innovation appear with a certain regularity (Euchner, 2016; Meyerson, 2016). Organisational culture is also problematic because while it is necessary to have a culture of innovation, companies can become hostages of that same culture, preventing the emergence of a culture better positioned for innovation. This is because we know that changing or adapting to a culture is difficult. It is so complex, and that is why we embrace a culture of innovation in the first place. What if the new culture does not work? It is not easy to introduce a new culture (Euchner, 2017). Innovation and the management of processes and operations will inevitably conflict because organisations were not created for innovation.

The traditional need to create structures and controls, typically applied to all projects, works to defeat and delay innovative capacity (Meyerson, 2016; Govindarajan, 2010; Glass, 2006).

An important component of innovation is "boundary spanning", described by Lynda Gratton (2007, p. 3) as the process of linking or bridging different groups, disciplines or organisations to facilitate collaboration and innovation. The fact that it is open to other domains is a valuable strategy for organisations that want to innovate, as it allows the exchange of ideas, knowledge and resources between different groups or organisations. This combination with the outside domain is the thing that "fuels innovation through novel combinations" (Gratton, 2007, p. 77). Ernst and Chrobot-Mason (2010, p. 2) also recognize that there are boundaries around us but with two different meanings. On the one hand, we have borders that limit creativity and innovation and prevent social and business changes. On the other hand, we have borders that are the limit where the most advanced thinking and the most innovative ideas live. As a result, boundary spanners are those individuals who reach beyond current borders and look for new frontiers in the unknown. For this, they argue that there is a need to create a boundary-spanner leadership capable of "create direction, alignment, and commitment in search of a higher vision or goal" (p. 5). Edmondson and Nembhard (2009, p. 123) explains that one of the benefits of innovation teams is the boundary-spanning skills that largely contribute to the team's success since communication with teams outside their own integrates deep expertise in different areas in order to create innovative products. The concept of "boundary-spanning" becomes elementary when we think about promoting collaboration and innovation in companies. It involves creating a culture that brings together different groups of people, disciplines, or even other organizations in order to allow an exchange of ideas, knowledge and resources between the diverse groups that, combined, generate innovation. To facilitate interactions, we have "boundary-spanners" who are able to reach others beyond their borders and are skilled in integrating experts and expertise from other areas.

3.5.2. The talent for Innovation

People are the ones who innovate, which is the fundamental element of innovation management (Meyerson, 2016). We could not talk about innovation without mentioning people. Euchner (2016) explains that there is a desire among people to create new things and that employees will do whatever it takes to innovate if they believe that they actually have a chance to create new things (Euchner, 2017; Pikkarainen, 2011). When we talk about programming, it is people who create code. Unless you have a vision that advances in Artificial Intelligence (AI), bring the possibility that machines can write their own code. With the appearance of chatGPT in 2023 (Hurst, 2023), some challenges were posed for programmers in particular, such as the possibility of generating all the functional code of a website from a hand-drawn mock website draft or simply generating computer code from almost any prompt (Sanderson, 2023). Some experiments were carried out with AI to correct errors or generate code, and the results are satisfactory as an additional troubleshooting platform, but human oversight is still needed since the platforms are not yet infallible (Stokel-Walker, 2023). The creation of new knowledge passes through the minds of programmers who create new programs and software through their encounters with new environments and information (Smith and Paquette, 2010). It is necessary to think deeply about the different elements of human resources in order for an organisation to be innovative. Furthermore, this has to be understood and internalised since we are talking about innovative thinkers. Everything has to be part of a culture of innovation, from recruitment, selection, training to ongoing motivation – intrinsic, internal based on personality and extrinsic – based on external circumstances (Pikkarainen, 2011). As a result, programmers who are contractors represent a different way of seeing the world, which can be used within the project as a resource from a completely different setting. According to Amabile's Componential Theory of organisational creativity, "expertise and creativity skills define what a person can do, and … task motivation defines what a person is willing to do". According to Meyerson (2016), the incredibly effective innovators are those people who are experts in a system or a speciality but who at the same time manage to have an intelligent conversation in more than one subject, knowing how to work in a team.

3.5.3. Diversity

Another critical factor in finding talent for innovation is diversity – not just in terms of demographics (gender, ethnicity, age, disability, etc.) but also in the diversity of technical disciplines and different approaches to thinking (Euchner, 2016; Pikkarainen, 2011). Most creative problem-solving processes happen in teams. People with different backgrounds and life experiences may have different thinking and approaches to solve programmatic problems and increase the possibility of emerging creative and innovative solutions (Pikkarainen, 2011). If an organisation is interested in creating innovations that really matter, a diversity of approaches will always be needed. "Diversity adds value" (Meyerson, 2016). In the same way, people who speak different languages may express themselves in different styles and use programming languages to programme in different ways. Programmers whose university education was different from computer science may not be attached to preconceived ideas about programming. They are also holders of skills that are unmatched in other schools. When we understand the value of diversity, we will all be closer to realising the potential highlighted by the people in our organisations.

David Epstein (2019, no page) argues that having diverse experiences and perspectives is essential for innovation, problem-solving and adaptability. In terms of diversity, Epstein (2019) discusses not only the importance of different perspectives but also diversity in terms of gender, culture and ways of thinking for innovation. Recently we have a new way of thinking and seeing neuro-diverse individuals – such as those with autism, ADHD, dyslexia and other conditions – and their integration into innovative teams. Studies, such as that by Nancy Doyle (Doyle, 2020), show that individuals with neuro-diverse conditions offer a number of skills and perspectives that can be beneficial in some types of tasks, such as problem-solving, pattern recognition and creativity (Pikul, 2023; Axbey et al., 2023). Innovation teams have recognized the value of neuro-diversity in part due to the hype created by teams at Microsoft, SAP and Virgin, who have implemented neurodiversity programs that provide opportunities for these individuals and explore innovation for organizational success (Doyle, 2020).

3.5.4. Talent Development

The development of talent for innovation encompasses continuous education. It forces people to keep up-to-date within their area of expertise and create new horizons of knowledge. It is essential to know how to use the knowledge generated and all the creative potential of people within an organisation (Pikkarainen, 2011).

> "Never turn down a chance to learn something. You never know when that knowledge might turn out to be just what you need, years later, in a changed world, for a situation you can't possibly anticipate right now." (Sawyer, 2013)

Returning to Dane (2010, p. 593) and his cognitive entrenchment thesis (see 3.4.1 above), one of the moderating factors that counteract the entrenchment is the extent to which individuals focus their attention on tasks outside their domain of expertise. As a result, talent development that is not simply focused on the area of expertise can help experts to be more effective in their professions when carrying out activities in multidomain engagement. This applies not only to scientific areas, where a body of research points to many being polymaths – in addition to their scientific expertise, they also have interests in music, art or literature (Dane, 2010).

Some organisations, namely IBM, give weekly time for employees to develop their own educational plans. It may be one day a week or a few hours, but the final aggregate product is immense as it encompasses all employees getting smarter with each passing week. In the end, the organisation is the biggest beneficiary (Meyerson, 2016). Google also became known for implementing a job twenty per cent rule, in which employees were encouraged to use one day a week to develop personal projects in their area of expertise or in a completely different domain. Initially, this rule was introduced as a motivational action, but the company received immense benefits in return, as it allowed its employees to free themselves from cognitive entrenchment as they acquired job expertise (Dane, 2010).

3.5.5. Risk

In my research about innovation, "risk" always appears in one way or another. Risk can be seen in at least two ways: in the general context of innovation or software development in particular. On the one hand, we have the risk that a programmer places in their code to gain a competitive advantage and the way the organisation deals with it, or the way the organisation embraces that learning process. On the other hand, we have an understanding of where innovation meets risk, analysing the innovative aspects of what we are building (Pikkarainen, 2011). There is no tradition of using creativity in building software. Some studies have shown that software innovations are rarely planned, being much more associated with risky behaviour. (DeSouza, 2005)

> "The innovator (or innovation entity) must take risk in experimenting with new ideas, inventing novel solutions, and questioning the status quo" (DeSouza, 2005)

Desouza argues that only a handful of engineers actually take risks in software engineering, and he calls Radical Engineers the ones that do take risks. These engineers find themselves in both super-controlled environments (order) and looser systems (chaos) where they are left to their own devices to create, test and experiment as long as it does not interfere with the organisation's goals. At first glance, this may seem good. However, organisations do not always capitalise on the learnings of these radical engineers.

The ability to tolerate, manage and encourage risk taking leads us to the following concepts: expertise and complexity.

3.6. Complexity

Complexity theory is a tool / lens that I plan to consider for creativity and innovation in software development.

Just as in human language, there are styles, degrees of complexity and expertise, in computer language, we find the same. As we program, we become experts, our programs increase in complexity, and we manage to create and represent more elaborate concepts. Programming languages consist of fundamental rules, which, when employed within software programs, introduce their own rules and which, in turn, become complex systems (Loback, 2018). The process of crafting computer code inherently entails a level of complexity. Our primary concern as programmers is to keep our programs under control (Haverbeke, 2014). Moreover, as we advance in our specific knowledge and in each field of study, the complexity we create increases exponentially, decreasing our ability to deal with it (Meyerson, 2016). There is a constant need to remain relevant in our field of action, and for that, we have to keep up with reading papers and new information, programming quizzes, and all the new trends that appear in relation to our technology. This exponentially growing technology, leaving us obsolete, is the same one that moves us forward, opening our horizons to a new era – the Cognitive Era, according to IBM.

3.6.1. Chaos Theory

Creativity and innovation are linked to chaos theory. A *chaotic system* is a system driven through many interactions, seemingly random. Each interaction produces new knowledge, which in turn forces a change in the entire system, influencing future outcomes. The chaotic system must embrace change in order to continue to be innovative. It is not that a chaotic system is complex or unorganised, but it is a system that adapts quickly (Smith and Paquette, 2010). The idea here is "embrace and encourage chaos to facilitate knowledge creation". David Snowden created a framework dubbed CYNEFIN (The Cynefin Co, 2023) to help individuals and organisations deal with complex situations and make decisions according to complexity, emphasising the importance of using different approaches to different situations. The four domains of complexity are "clear" for situations where the relationship between cause and effect is evident and can be understood by everyone; "complicated" when it needs expert analysis or research to be understood; "complex" for situations in which the relationship between cause and effect can only be perceived in hindsight and only appears through the interaction of several factors; and finally "chaotic" for situations where there is no apparent relationship between cause and effect, and quick action is needed to restore order. In this case, there is no single answer or best practice approach, and we will have to resort to "good" practice in opposition to "best" practice. Good practice refers to a set of principles or guidelines that have been developed through experimentation and learning (such as instinct or past experiences), and not through predetermined solutions, and has the ability to be more flexible and better adapt to the changing nature of complex situations. The CYNEFIN framework is a valuable tool to help one understand the different types of problems so that one can choose the appropriate decision-making approach to deal effectively with the situation.

This involves ensuring that all routines and acquired knowledge that inhibit knowledge growth are abandoned and destroyed by the influence of chaos and the idea of crisis. A crisis comes from the uncertainty caused by leaving the comfort zone. This is not caused by system problems but is related to a search for different things, like acquiring new technologies or entering new markets. It can also be derived from system change, but whatever it is, it will disrupt and create sustainable innovations (Pikkarainen, 2011).

3.7. Summary

In the knowledge landscape, I made an introduction to the literature review, looking first at the scope, defining the limits of my review and defining the objective of comparing the area of programming with the production of code for computers. I showed my interest in the area through the challenge posted to programmers in Haverbeke's book (2014, p. 3) and how this influenced the course of my research.

I researched creativity by looking at well-known authors who defend it as accessible to everyone and as a catalyst for innovation. Nevertheless, creativity needs to be nurtured, and for that, we have to create an environment conducive to developing creativity in the organisations. I also looked at the literature that places creativity as a choice, considering one can adopt different mindsets and behaviours. I looked at the possibility of creative people being disruptors when they see life differently and, finally, the limitations on creativity that, in essence, are placed to resolve the inherent risks.

In the innovation part, I explained the differences between creativity and innovation by looking at it as the implementation of creative ideas. In this portion, I talked about programmers' capacity to innovate daily by being creators of new things that did not exist before. As with creativity, I showed how creating a culture of innovation that lasts and consistently produces new solutions is necessary.

Still, within the scope of innovation, I conveyed the importance of talent so that organisations can be competitive and the contribution that programmers bring in this area within the challenges brought about by advances in Artificial Intelligence and the evolution itself of the programming profession. In this respect, the diversity of teams is important, not only in terms of demographics but also in terms of technical disciplines and different ways of thinking. Afterwards, I also looked at talent development, where experimentation plays a relevant role. Finally, and still, within the scope of innovation, I looked at risk in a general context and in the field of software development in particular.

I examined complexity theories as a lens when considering creativity and innovation in software development, considering the complexity of computer code generation. Within the complexity, I investigated the chaotic systems that, in a way, encourage the creation of new knowledge.

Finally, I looked at the expertise and its connection with intuition to conclude my investigation into the knowledge landscape. The expertise was studied together with Evidence-based practice, which also includes knowledge acquisition models and the role of deliberate practice in knowledge acquisition and its ramifications and critics. Intuition appears as one of the most important characteristics when referring to expertise, which was enriched with some ideas about tacit knowledge.

Part 2: What we want to know

4. Methodology

This chapter marks the beginning of the second part of this research. This part addresses how I will approach the data analysis and all the theory involved in the extraction of what I still do not know about the project. It includes the methodology and all the project activities developed.

This chapter will address the methodology used to carry out this project. I will begin by addressing the theories of knowledge that influenced my epistemology, which in turn generated my ontological position. Epistemology justifies the way knowledge is created in this project. Next, I discuss the methods used to collect and analyse the data. Finally, I explore practice-led research.

In the introduction to this project, I wrote a little about myself and my life path until I started this research. You can find some nuances about my life principles, values, and what I consider important there. It is important to point out that I consider myself a creative person with different ideas and ready to question the status quo, however absurd it may seem. From drawing in the wet sand as a child after the daily tropical rain in my backyard to the creation of a Digital Rights Management system in my master's thesis throughout innovative programmatic solutions in the various web software development projects where I worked, my life was based on this idea that there are new paths to be followed and that I must always try to put my creativity into action to solve the most varied problems. To this almost rebelliousness, I add the confidence to take risks. Life does not make sense if you have everything pre-programmed and if you do not have an adventure in between. Moreover, this adventure of doing academic research was one of the bravest I have ever taken. I embarked on it because I wanted to learn and because it would be the time and space where I could leave a mark on my profession.

There is a transformative power in the learning process (Montuori, 2008). Throughout our lives, we become experts in what we develop daily, and most of the day-to-day we spend in our professions, sometimes in repetitive tasks, other times in really challenging situations that take us to the limits. Most of this development comes from professional practice, and there is a gap between the community of practice and the academy. It happens because there is no production of academic knowledge in this area. It is not that knowledge does not exist. Knowledge exists, and it is there in the community of practice. However, it needs extraction, analysis and theorization to reach the academy, be shared, and reach more academics and practitioners. It is not enough to value only academic knowledge. Donald A. Schön (1987, p. 309) places the reflective practitioner at the centre of the gap as a bridge between the academic world and the community of practice. He also mentions interdisciplinarity in the context of bridging the gap as a lens for problem-solving (p. 310).

4.1. Knowledge

In the search for knowledge within a work-based knowledge setting, Raelin (2008) defines an epistemology of practice that intends to explore two types of knowledge, tacit and explicit (p. 67). Explicit knowledge is that knowledge that can be formally transmitted to others and can be articulated through language. Tacit knowledge is that knowledge associated with the context, embedded in the individual's attitudes and actions, but which, although the person knows what he is doing, sometimes it becomes difficult to articulate to others what he really knows (p. 68).

Nowotny, Scott and Gibbons (2003, p. 179) introduced a new paradigm of scientific discovery and knowledge production which they called 'Mode 2' – knowledge that is "socially distributed, application-oriented, trans-disciplinary, and subject to multiple accountabilities" linked to research and innovation. This type of knowledge is in opposition to 'Mode 1', which is characterised by being mainly theoretical. Some characteristics of this 'Mode 2' knowledge are the fact that this knowledge is created within an application context, where the problem is generated and methodologies are developed, as is the case of work-based research; this knowledge comes from a 'trans-disciplinarity', that draws on different theoretical perspectives and practical methodologies to solve problems; there is a great diversity of sites where knowledge is generated and also about the types of knowledge produced; knowledge is highly reflexive, becoming an intense dialogue between 'research actors' and 'research subjects'; and the last characteristic of this 'Mode 2' knowledge is the fact that it has given rise to new forms of quality control, with "multiple definitions" of quality.

My biggest influence is how I see knowledge and the process in which this knowledge was developed (Saunders, Lewis and Thornhill, 2009).

4.2. Epistemology

The theories of knowledge explained in the previous point influenced the way I see my epistemology, which is the nature of knowledge and how I understand that there is something to be known, the critical study of its validity, its methods and scope (Willig, 2008; King, 2010).

There are three aspects of my epistemology that I would like to use to justify the creation of knowledge in this project: Empiricism, on the one hand, and, the other hand, the adaptation of two theories of knowledge that, in a certain way have more in common with each other, which is Constructivism and Interpretivism.

In taking an empiricist stance to research, it is common for the researcher to posit an Empiricist Epistemological position in relation to knowledge, asserting that our world knowledge is derived from our experiences. In other words, by collecting and systematically classifying the observations, the sense of perception produces the basis for the acquisition of knowledge. According to this view, the set of simple observations is combined to give rise to more complex ideas, obtaining the theory through these observations. The Empiricist Epistemological position works on the idea of adding the researcher's experience to the project.

The empiricist idea contrasts with a more interpretive position in the areas of arts and creativity. Because it is impossible to obtain direct access to the co-researchers' entire lives in this study, it is essential to obtain the co-researchers' experience from my perspective – an insider's perspective. This exploration implies obtaining my particular views as well as the nature of the interactions between myself and the co-researchers. The co-researchers try to make sense of their own experiences and I try to make sense of the co-researchers trying to make sense of their experiences (Smith and Osborn, 2008). The aim is to discover how people feel about the subject and how they make sense of their lives within their vantage points (King and Horrocks, 2010).

Interpretivism and Constructivism are closely related in the sense that they both go against the idea that an objective reality can be studied without considering each individual's interpretations, personal perceptions and experiences. While Constructivism focuses more on the individual construction of knowledge, Interpretivism looks at the importance of cultural and social factors that shape knowledge. In short, in Constructivism, knowledge is built on top of the individual's previous experiences, which includes what one believes in, as well as all one's social interactions; in Interpretivism, all emphasis is given to the role of interpretation and meaning creation in the process of knowledge creation. Here, we do not have knowledge as a reflection of the world, but we have an active interpretation and sense-making process. Knowledge also takes shape within a context where subjective experiences shape knowledge against the backdrop of cultural and social factors that influence individual interpretations (Willig, 2008).

Looking at the epistemology adopted, I make a direct connection with what I intend to study, which is creativity combined with programming. If, on the one hand, we have me and the co-researchers as individual people who will collaborate in this project with our particular experiences as programmers, on the other hand, we have our collective experience in our jobs, where, eventually, we work together, and that collaboratively we will explore our views, myself and each co-researcher. As a result, our reality is also built through our social context in which we operate and to which we bring our entire history, career, learning processes, visions about work and paths taken individually and collectively. The interpretive part is done both by me and a few by co-researchers. Moreover, there is an inherent subjectivity because we are also talking about cultural and social factors that led to my relativist ontological position (Costley, 2010).

When I mention relativism here, I am simply acknowledging that the environment in which the knowledge I seek in this research develops is diverse and unstructured, being a product of various individuals interacting with others, open to different interpretations (King and Horrocks, 2010).

4.3. Methodology

The choices made in methodology are influenced by epistemology. I mentioned Interpretivism above, and usually, but not always, qualitative approaches are associated with Interpretivism (King and Horrocks, 2010).

In this qualitative research, I am making an interpretation of the co-researchers' experience using the lenses and pre-defined assumptions about the meaning of what we are trying to understand. Interpretation goes beyond descriptive analysis, having an orientation towards meaning and the importance of these meanings. At least three tools can be used in a qualitative study to create knowledge with an interpretation-oriented direction. The first interpretive tool is the Interpretative Phenomenological Analysis (IPA) which tries, as an approach, to investigate why the phenomenon occurs by exploring the perspectives of those who experienced the event. The second is the Discourse Analysis (DA) that looks at reality as a sociological production that focuses on explaining meaning using language/discourse. The third tool used in qualitative research with an interpretive orientation is Grounded Theory.

Grounded Theory is a qualitative research methodology whose objective is to create a theory through data analysis. The fact that it is entangled in a close inspection of the data reveals the name "grounded". Two sociologists, Barney Glaser and Anselm Strauss, developed this methodology and later extended the subsequent work with Juliet Corbin (Corbin and Strauss, 1998). In the Grounded Theory, the analysis is organised around key categories that are extracted from the data. Due to this type of construction, it is advisable not to explore the relevant literature before starting the analysis to prevent pre-conceptions about existing research to make the research really emerge from the data. Within Grounded Theory, there is also the idea that research is part of the things we are exploring and the data we generate (Braun and Clarke, 2013, p. 184). In this logic, my past and present involvement and interactions with the research and the co-researchers helped move forward in the research.

> "... its ability not only to generate theory but also to ground that theory in data. Both theory and data analysis involve interpretation, but at least it is interpretation based on systematically carried out inquiry." (Corbin and Strauss, 1998)

Based on the grounded theory, Guest, MacQueen and Namey (2012, p. 15) developed a methodological framework for qualitative studies that is a synthesis of various approaches and procedures in which it is necessary to perform a thematic analysis, which they called Applied Thematic Analysis (ATA). ATA is a research method that involves a detailed analysis and systematic examination of qualitative data in order to identify, analyze and report themes or patterns within the data. As it is based on grounded theory, the themes and patterns are derived directly from the data without being the imposition of the researcher's preconceived ideas. One of the strengths of this methodology is that it is adapted for use in large datasets that seek to find solutions to real-world problems, not just building theoretical models. This flexibility allows one to capture rich and nuanced insights from the experiences, perspectives and behaviours of individuals across various disciplines.

I was not so purist in my approach to Grounded Theory because when I knew in advance that I wanted to study creativity associated with programming, I had the notion that these subjects would appear in my data, which later happened. However, for the development of the project and the presentation to the Program Approval Panel, it became necessary to analyse the context of creativity and programming within an already existing body of knowledge at the beginning. In a way, this goes against the principles of Grounded Theory, which recommends against exploring relevant literature before beginning the analysis to prevent pre-conceptions about the research and for the investigation of themes to emerge from the data. In order to mitigate these pre-conceptions, I had to make an effort to pay special attention to patterns, new categories or concepts that could emerge throughout the data creation process (Walliman. 2004, pp. 260). The framework I used, Applied Thematic Analysis (ATA), which is based on Grounded Theory (GT), shares some similarities but differs in some aspects. ATA and GT involve qualitative data analysis. GT emphasizes the creation of grounded theories without preconceived ideas (Braun and Clarke, 2022, pp. 288), while ATA is more flexible in its theoretical orientation by focusing on the analysis of themes to resolve specific issues or objectives of the research questions, not necessarily to create a theory grounded in empirical data (Guest, MacQueen and Namey (2012).

Below is a table with a summary of the ATA approach to data analysis:

Defining Features	• Identify key themes in text; themes are transformed into codes and aggregated in a codebook. • Uses techniques in addition to theme identification, including word searches and data reduction techniques. • Can be used to build theoretical models or to find solutions to real-world problems.
Epistemological Leaning	• Positivist / Interpretive • Positivist in that assertions are required to be supported with evidence (text). • Processes are also systematic and quantification can be employed.

	• Methods and processes in ATA (except those of a quantitative nature) can also be used in an interpretive analysis.
Strengths	• Well suited for large data sets. • Good for team research. • Inclusion of non-theme-based and quantitative techniques adds analytic breadth. • Interpretation supported by data. • Can be used to study topics other than individual experience.
Limitations	• May miss some of the more nuanced data.

Table 1: Summary of the ATA approach to data analysis
Table adapted from Guest, MacQueen and Namey (2012, p. 17).

4.4. Methods

The methods are influenced by the methodologies and choices that were made above and are the techniques or procedures used to collect and analyse the data (Strauss and Corbin, 1998). Regarding assumptions about the world around me, the reality I intend to study and the knowledge I intend to generate are constructed through language, producing a particular version of events (King and Horrocks, 2010). Based on the importance given to language in the generation of knowledge, it is not surprising that the data collection method for analysis is done through interviews since knowledge will be generated through a verbal exchange (p. 20). What I am interested in knowing is the meaning and experience that the participants had in relation to creativity and programming. Although I am not interested in causality, it may be that the participants have a perception of causality that could be interesting for this study.

4.4.1. Semi-structured interviews

The first considerations to be taken into account when choosing the data collection method through interviews are the aims and objectives defined in section 1.3 above, since the nature of any interview must be consistent with the research question, the objectives and the adopted strategy (Saunders et al., 2009). Given that I intend to find out the reasons for my co-researchers' decisions and to understand their attitudes, opinions, and experiences towards creativity and other topics, I will conduct a series of semi-structured interviews. The interviews will be semi-structured as I want to explore how co-researchers interpret their own experience with "creativity" based on a flexible interview guide. The semi-structured interviews allow me to "probe" answers, where I intend for the co-researchers to explain or add information they have already uttered. Semi-structured interviews become more relevant in this context because I am also adopting an interpretivist epistemology (as mentioned in section 4.2 above), and I am interested in the meaning that co-researchers give to their contribution. Participants will use words and share ideas in different directions, and it is important to get from the co-researchers the depth of meaning they are giving to their own thinking. And the co-researchers will not answer all the questions. The flexibility will allow the adaptation of the questions and asking follow-up questions to the answers the participants gave beforehand (Saunders et al., 2009). This study will be cross-sectional, where I will collect information from different co-researchers at a single point in time and check the relationships between all the conversations in the data mix ("Discussion") at that particular moment. The way my semi-structured interviews took place is described in the section 5.3.1 below in the "Project Activity".

4.4.2. Co-researchers

My engagement with the co-researchers was between March 2019 and June 2020 (around 15 months). Within my community of practice, I chose to bring developers with much experience in the area of programming to this project. When studying creativity, I was not only interested in developers who initially showed an interest in creativity, but I wanted to engage with the programming community in general, having as a common denominator a certain degree of expertise that was defined by me. The selection criteria are defined in detail in section 5.2 "Co-researchers" but in summary, are:

- Developers that worked with me in the past;
- The level of experience, measured using the SFIA model (Skills Framework for the Information Age);

The SFIA is a UK Government-backed framework for IT knowledge, skills and training covering the whole of IT (Loback, 2018). The framework is organised in a three-dimensional matrix, divided into six categories and nineteen subcategories, which details ninety-six skills needed by IT professionals in positions involving communication and information technology within seven levels of expertise/responsibility. The seven levels of increased responsibility, accountability and impact from Level one, the lowest, to Level 7, the highest, are:

Level 1 – Follow
Level 2 – Assist
Level 3 – Apply
Level 4 – Enable
Level 5 – Ensure, advise
Level 6 – Initiate, influence
Level 7 – Set strategy, inspire, mobilise

4.4.3. Data Analysis

In this section, I will conceptually define the data analysis process and my plan for analysing data, and in the next chapter (chapter 5 “Project Activity”) , I will show how I adapted this concept and applied it to each phase of my project.

Data analysis starts with preparation. Preparing for analysis is important because whenever you are manipulating data, there is an opportunity to think about it, and how information is organised impacts the efficiency and effectiveness of the analysis. Throughout the literature, special attention is given to data preparation, particularly the part dedicated to transcription (Saunders et al., 2009; Bazeley, 2013; Braun and Clarke, 2013), transcription formatting and text translation. My interest is not only in the words that the co-researchers said but also in the mode they were uttered, and in a way, I want to convey the tone and other contextual information that can help give meaning to semi-structured interviews.

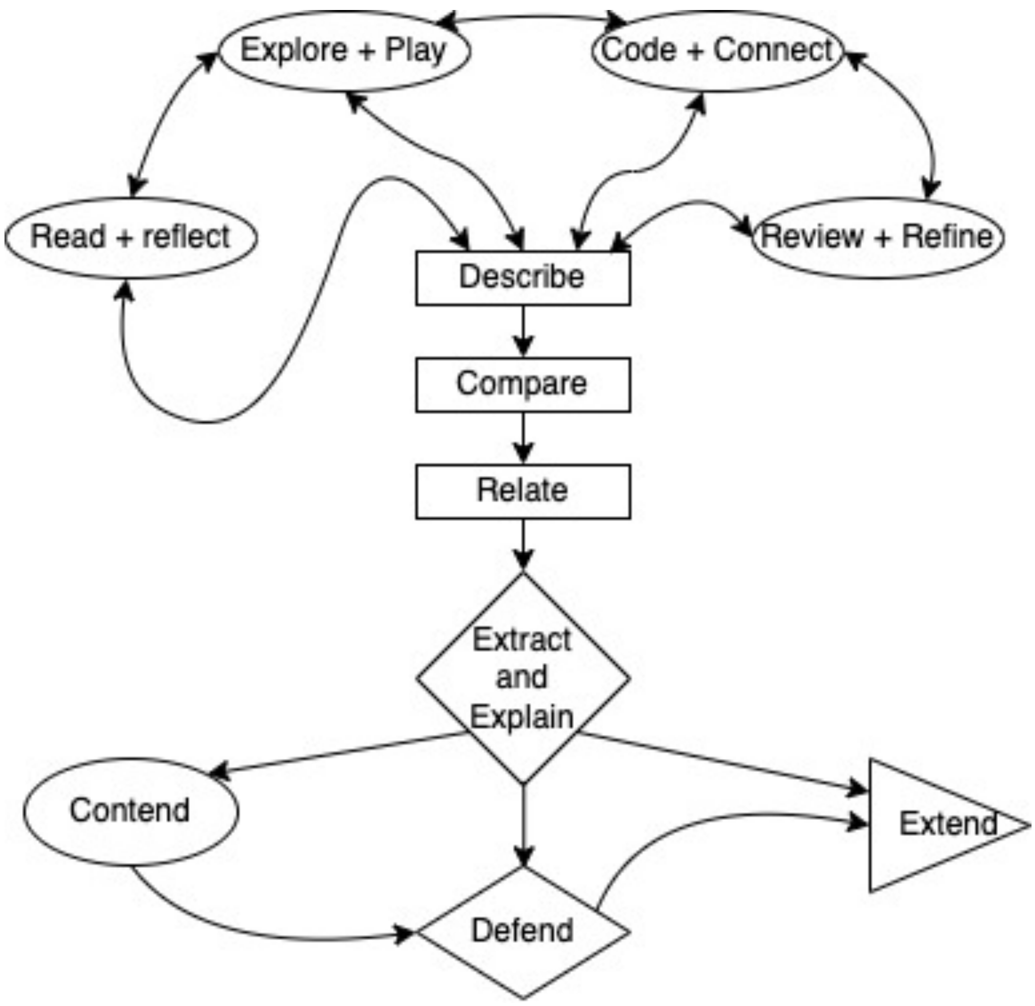


Figure 2: Foundation for Analysis
Adapted from Bazeley (2013, p. 14)

Data analysis will support the results of this project. The main objective is to study how experts perceive creativity in programming. I intend to do an exploratory analysis of the data, as the emphasis will be on what emerges from the interactions I have with co-researchers (Guest et al., 2012). These interactions will lead to the creation of code and the development of themes that will determine how successful the analysis is. The methodology used is the Applied Thematic Analysis (ATA), as described above in the section dedicated to "Methodology" (section 4.3 above). This adaptation of the Grounded Theory is more favourable to an inductive logic of analysis since the conclusions will have their starting point in the data ("grounded"), and the result appears in the way individuals feel, think and behave in a given context in relation to a research question (p. 13).

For the analysis, the following elements will be used:

- Semi-structured interviews with my co-researchers (see details in 5.3.1 below);
- Fields notes that I took during the interviews in addition to them;
- Handwritten notes and journals I took as the project progressed;
- Personal Cloud-based journals;

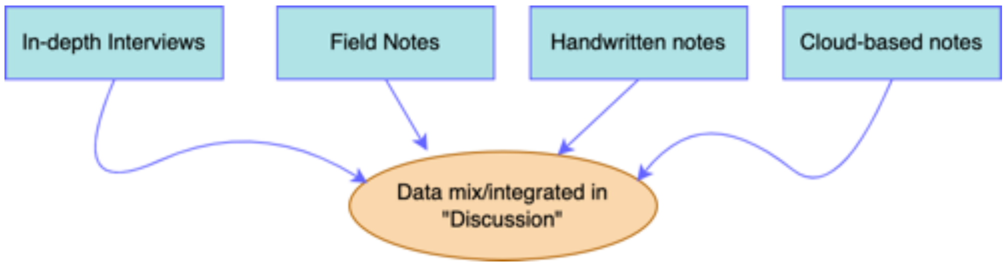


Figure 3: Research Design Illustration

Here is a transcription of a field note taken after an interview:

> "The interview went well – I think. At lunchtime, we went to a meeting room with just the two of us without distractions. The interview ran in a relaxed way, with much laughter in the mix, and <anonymous> did not refrain from talking while the interview lasted. It took us an hour and forty minutes to complete my set of questions. It was a long interview. Ideally, it would have been an hour, but <anonymous> extended out the answers, and I did not want to interrupt. For my subsequent interviews, the questions will also have to be shortened, and the order changed and re-organised so that the answers can fit within an hour. The duration was an exaggeration and later will disrupt the transcript, among other things.
>
> I fear that the questions I asked did not answer my research question. <Anonymous> was peremptory: there is no creativity in the code. However, <anonymous> is a pragmatic person who

looks first for results with the least possible effort in terms of programming. Maybe the contractors are like this. I do not know." — Loback (2019)

The way the code and themes will be defined follows the defining features of Applied Thematic Analysis (ATA) of "locating meaning in the data" by identifying themes and defining boundaries around these themes through segmentation. A different feature of the ATA concerning thematic analysis using grounded theory is that it creates two classifications for the themes, one of which is "structural" based on research design and which can directly affect the resulting data. These themes are informed by what we already know and have a direct influence of the researcher. The other categorisation of themes is "content" or "emergent" themes, based on the same principle of grounded theory in which themes "emerge" from the data and reflect what we do not know and want to investigate (Guest et al., 2012). I started with three structural themes, which were defined at the beginning of my literature review: Complexity, Creativity, and Innovation (section 3 above), where innovation was integrated inside "creativity", and "complexity" ceased to be as relevant after the data analysis was carried out. After identifying key themes in the text, they were transformed into codes, which allows the creation of a semantic boundary of a theme, later aggregated in a codebook. The development of the codebook becomes a fundamental tool to concentrate the observed meaning in the text and systematically sort it into categories, types and relationships of meaning (p. 52), including a description of how the codes relate to each other. An adaptation of my codebook is found in Appendix section 16.5 ("Codebook").

Table 2: Extract from the codebook

Name	Description	Files	References
= Expertise	Top aggregator for the Theme Expertise	17	268
Expertise	Questions about expertise	17	178
= Developer Beliefs	Opinions in relation to their own epxertise	16	34
don't believe they are an expert	Do you believe you are an expert?	10	19
pointing towards being an expert	Do you believe you are an expert?	8	15
experts take the leadership		7	12
exposure to different programming languages	exposure to different programming languages helps in the pursuit of expertise	11	28
exposure to other people	Collaboration as a way to innovate	11	17
Intuition	The use of intuition in the context of expertise	13	28

I used an entire section to talk in practical terms how I carried out my thematic analysis in the next chapter (section 5.6 "Thematic Analysis") based on the plan defined here. There, I exposed the process of creating and changing the code (5.6.1.4 and 5.6.1.5), including data analysis (5.6.1.1), the inductive process (5.6.1.2), interview labelling (5.6.1.3) and also wrote about the origin and handling of themes (5.6.2).

4.5. Practice-led Research

Nicolini, a researcher in the area of organisations, confirms a relation between practice and knowledge, culminating in the idea of "site" as the place where "knowing is both sustained in practice and manifests itself through practice" (Nicolini, 2011). According to this view, a site of knowing (a practice instance) is "not any practice in particular but is a nexus of interconnected practices" – multiple relationships between practices. When we examine the practice, we are looking at an instance of knowing linked to other instances of knowing, which in turn links to others. This idea of interconnectivity explains the need to analyse several knowing in practice situations to perceive the complexity of references between them (Loback, 2018). This is the starting point for the methodological development of this project, which is the analysis of my practice. This analysis will explore the complexity of the relationships developed in the professional environment where I work to develop a methodology to analyse different aspects of my practice and their interconnectivity in terms of "knowing in practice".

Practice-led research is more prevalent in the artistic context as it is artists who are at the forefront, always testing their visual intuitions against the logic of what was seen and known at the time (Sullivan, 2009). Artists have this ability to go against established norms and create things that cannot fit into what tradition is. The development of science and knowledge is only possible to advance with great strides when the practitioners challenge the status quo and its assumptions (Parker, 2005). This strain produces tension and moves the knowledge forward. Sullivan explains that there are two lessons that we should learn from the artists. First, the protagonists' ideas are revealed, in most cases, through the process of reflexive and reflective inquiry. This responsibility should not be left to critics or historians, but artists should have the role of the practitioner, researcher and theorist. Second, there is a need for communication between the various fields of inquiry. By understanding the different connections between different perspectives, we are giving rise to the possibility of intuitively making leaps in knowledge through the creation of new knowledge. Practice-led research "is not a methodology but a focus" (Wisker, 2007). The focus is on the researcher's area of activity, and as such, it needs to balance out between the day-to-day job and the work for the research.

The notion of transdisciplinarity in research is an area that has gained supporters in UK universities, and that has been characterised by focusing on 'wicked problems' that will only be solved when a good dose of creative solutions are thrown at them (Gibbs and Beavis, 2020). At the same time, there are also discussions about knowledge that is co-produced in partnership with entities with the scientific or academic knowledge but that also bring together other forms of practice and local and personal knowledge. Gibbs emphasises that transdisciplinarity should not only be applied to large social problems that need creative solutions, but it also works very well in situations or problems that want to be investigated by universities and the workplace. Here, the role of artists, as individuals who are willing to break societal and academic paradigms, is essential to overcome the limitations that the results of using transdisciplinarity as a research tool present. Kate Maguire, one of the world leaders in transdisciplinarity cited by Gibbs and Beavis (2020) explains that transdisciplinarity began to offer a new framework in relation to the notion of interconnection and to make sense for people outside the university, who in their positions as senior practitioners are never single disciplined. This evokes 'Mode 2' explained in section 2 above, referring to work-based research and transdisciplinarity. Do not forget that Augsburg (2014, p. 244) points out that individual experience and cultural aspects are increasingly seen as important as the educational background in the transdisciplinary aspect, which also emphasises creativity.

It makes perfect sense to look at artists and practice-led research as this scientific work seeks insights into the practitioner-researcher, the knowledge it brings, and the critical process (critical reflection and reflexive action). As a starting point, it will draw from my own experience and context, using reflection to leverage the work-based doctorate (Loback, 2018).

4.5.1. The practitioner

I am a practitioner. I am a researcher. I am a member of the community of practice under investigation in this study. I share the same frustrations as my peers when confronted with code obstacles, "purist developers" and managers, and the constraints in code development. My perspective significantly influences my approach to this project, as I am actively engaged in it. Through Applied Thematic Analysis (ATA – based on Grounded Theory, is a synthesis of several approaches and theories), my role is not exclusively to collect data and have a subjective and neutral role in the investigative process. I will reflect on the essential themes of the co-researchers' experiences about the phenomenon but also look at my own experiences and interactions with the co-researchers and with the same themes. This is an essential difference between a more descriptive transcendental phenomenology, which places the researcher in epistemological terms in a bias-free position, and a more interpretive, hermeneutic phenomenological position, which places the observer on an equal footing with the co-researchers, being part of the phenomenon. I bring up my assumptions and explain the phenomenon through interpretive means. By drawing conclusions, I become part of them. I am not an impartial or neutral observer of what is happening around me. This mirrors the importance of reflective practice throughout the process (Thompson, 2017, p. 84). In my epistemology (section 4.2), I define this interpretive stance that intends to convey my perspective on interactions with co-researchers, including my biases and opinions (Smith and Osborn, 2008; Willig, 2008; King and Horrocks, 2010).

4.5.2. The critical process

This project relates to my academic path and professional background. Although I am not using self-ethnography as a research paradigm, my writing style, when composing my story and working with co-researchers, strongly associates with the type of voice adopted when writing from a self-ethnographic perspective. This happens because what I write is also a reflection of many scattered programmers around the world, who, like myself, also feel deprived of using their full innovative and creative potential due to the lack of openness to creativity and innovation, as code is concerned (Loback, 2018). Montuori (2008, p. 24) talks about introducing the researcher into the inquiring process as a central issue. Furthermore, as a result, the "subjectivity" of the inquisitor must be replaced by the "bracketing" of the inquisitor's beliefs and the fact of transparently stating one's assumptions. This process of inquisition becomes a self-reflection, and the enquirer also becomes the object of the inquisition. What I am recognizing is that I cannot separate myself from the knowledge creation process (Montuori, 2008). *Autoethnography* is a qualitative social research method and methodology that describes the relationship between the researcher and the participants in a research project by providing the connection between the observer and what is being observed (O'Hara, 2018). The researcher's role in terms of analysis and the critical process is an interactive and iterative process. It works through cycles of data capture and reflection writing on what was captured. The iterations will lead to a robust analysis of how the data and reflection will evolve the perception of how each part contributes to understanding the phenomenon as a whole. The iterative concept is widely used in programming teams, and the researcher is very familiar with the "agile" method, which can be summarised as "small changes on a continual basis" (Cohn, 2010; Dingsøyr et al., 2012).

4.5.3. Reflective Practice

Reflective practice plays a crucial role in knowledge creation. I am theorising my practice and trying to make sense of my professional knowledge of my practice and the skills and values of the co-researchers in this project. The need to use reflective practice stems from the fact that there is a gap between theory and practice (Thompson, 2017; Raelin, 2008), which comes from a view of knowledge from a modernist tradition in terms of epistemology. I need to understand this relationship between theory and practice better, integrating the two so that "theory informing practice, while practice is also informing and testing theory" (p. 248).

“Work-based learning expressly merges theory with practice, knowledge with experience … Such learning, however, needs to be centered around reflection on work practices.” — Joseph A. Raelin (2008, p. 2)

Learning within the concept of work-based learning is more than learning a certain number of technical skills. It is reframing what we already know about the subject and incorporating what we are learning to create new knowledge. Theory only makes sense when incorporated into practice, and practice only makes sense when viewed through reflective practice augmented by theory.

According to Thompson, what should happen when we use reflective practice is to look at the situation in our practice and apply our professional knowledge, integrating the moment into a whole, which can be applied back to our current practice scenario (Thompson, 2017, p. 25).

“... theory shapes practice: knowledge shapes thought; thought shapes actions – that is, our practice.” — Neil Thompson (2017, p. 14)

Jack Mezirow, cited by Raelin (2008, p. ix), refers to learning in a work-based setting as occurring when we make “a new or revised interpretation of the meaning of an experience, which guides subsequent understanding, appreciation, and action”. This is a clear reference to reflective practice as a crucial way to grow knowledge acquired in a professional setting. Moreover, this knowledge has a more dynamic agent, which is “learning”. Learning is also a form of knowledge. Schön (1994) wrote that practitioners build their theory and knowledge as they consciously reflect on the challenges they are experiencing in their practice, iteratively choose problems, gather data, take action, make an assessment and then reflect. After this process, they then share the knowledge produced with others in their practice. Schön termed this “reflection-in-action” (1994, p. ix), which is the process of thinking and solving problems as one encounters new challenges at work by recognizing the familiar in the unfamiliar (p. 140). Professionals rely on their knowledge, experience and intuition in order to solve these challenges. The resolution is only possible through this reflection that allows bringing past experiences to the unique situation they find themselves in. Reflection-in-action is an efficient way for professionals to adapt to changing circumstances and improve their practice. This reflection-in-action involves a cycle of “knowing-in-action” and “reflection-on-action”. Through *knowing-in-action*, they will seek tacit knowledge and their past experience to make decisions at the moment, and through *reflection-on-action*, reflect a posteriori on the events and decisions taken to improve for the future, and this process should continue throughout their career. Through reflective practice, they can learn experientially about themselves, their studies, work, and how they relate to other people and the world around them (Bolton, 2018).

4.6. Ethics

Ethical considerations have been a concern since the presentation of this research to the PAP – Program Approval Panel in April 2018 (Loback, 2018). The following paragraphs reflect that submission, and the information is copied here as it should be part of the research. All research studies that involve people and investigate their relationships with the world around them must have a particular concern regarding ethical behaviour (Walliman, 2004). The ethical considerations are based on the work developed by Peter Allmark and the team, particularly in the literature review and discussion on ethical issues in in-depth interviews (Allmark et al., 2009) and the Middlesex University ethical obligations. I took the steps I deem necessary to ensure the best standard of research ethics (Loback, 2018).

4.6.1. Bias, dual role, and over-involvement

As an initial assertion on this project, I am biased towards the use of creativity in the code practice. I am a participant in this study. I am an expert in my field. I have a strong-willed opinion. I connect with hundreds of professionals working in the same area, having witnessed and participated in thousands of interactions between programmers and their relationships with other organisational agents. It is from the interconnection of these relationships that I want to obtain the knowledge needed to assert that when we allow programmers to use their full creative potential in creating innovative code systematically, the organisation builds its competitive advantage (Loback, 2018).

4.6.2. Privacy and Confidentiality

Participation in this research is confidential. There will be no possibility of identification of the co-researchers by the general public. Nevertheless, complete confidentiality is not assured because it may be the case that co-workers, who witnessed the interactions referenced in the final project, can identify the participant. Apart from that, and insofar as possible, all interactions with companies, people, co-workers, and specific locations, will be anonymised to protect the identity of the co-researchers. The remote possibility of co-worker identification was communicated to the co-researchers so they could make an informed decision as to participate or not (Loback, 2018). The co-researchers received a handout "Privacy Notice for Participants" before participating in the research (Appendix 16.4).

4.6.3. Informed Consent

I provided as much information as possible to the co-researchers about the objectives, processes, confidentiality, and personal data collection so that co-researchers could make a conscious decision to participate in this project (please see "Invitation Email" email in Appendix 16.2). Also, a brief handout was conveyed to the co-researchers with various information beforehand (16.3 "Participant Information Sheet"). The purpose of the brief was also to inform the participants about the various aspects to which they should give consent, namely the recording of data for later processing. There was also a concern about making it very clear that the participants could withdraw from this process at any time without giving any justification (Loback, 2018).

4.6.4. Harm

For the co-researchers, I did not anticipate any threat to their physical integrity, values or dignity (Loback, 2018). The co-researchers were not part of a risky or vulnerable group, and the subject evaluation was not sensitive, so there was no danger of harm to the co-researchers.

4.6.5. Politics and Power

The participants know me personally and have worked or are working with me, and because of that, I am reassuring them that there is no obligation to take part or give any information to this study. Participants are my co-workers, and I do not exercise any type of power over them. There is a risk that they may feel threatened in the future in their positions at work if we manage to work together again. I am advancing my career by doing a doctorate and gaining more knowledge and academic training, and I may become a stronger competitor if we are tied in the process of future recruitment. This may affect their participation and the degree of honesty in the responses. Another limitation is the fact that I am the editor that will cut, edit, and choose the parts of the interview that I assume to be relevant. There is a danger that they will feel misinterpreted or misrepresented in the ideas shared and their views (Loback, 2018).

4.6.6. Impact

This project will positively impact companies worldwide because they will reap the organisational and financial benefits of applying the principles outlined in this project. Herein lies the competitive advantage. The impact on the participants will be the same as the other computer programmers in the world: the ability to see their level of expertise recognised, allowing them to create the programming code with more freedom and intuition, asserting their seniority. The impact can also be detrimental if it is proven that, as participants, they do not have the necessary and inherent characteristics of creative programmers. If, after the publication of this project, it has the expected impact and companies start looking for other types of programmers with a different set of skills that they did not previously value (such as creative thinking), the insertion in the workplace could be problematic for some. They should search for new training and update their capabilities (Loback, 2018).

4.7. Summary

In this section, I analysed the methodology I will use to answer the question "How do experts perceive creativity in programming" and its variations. I laid the basis of the theories that influenced me in the choices of epistemology and the methods to obtain the data and generate the necessary knowledge for this project. The research methodology was drawn from Applied Thematic Analysis based on Grounded Theory, and I explained the reasons for using semi-structured interviews and the SFIA-based co-participant selection process. Afterwards, I articulated about Data Analysis and the theoretical basis of my choices. I also spoke about the importance of practice-led research and the contribution of Schön and Montuori to my reasoning. Finally, I described how I dealt with the research's ethical issues and the expected impact.

5. Project Activity

In this chapter, I want to describe the activities developed in this project. I am listing them in chronological order. Furthermore, this chapter also serves as a reflection on the development of this project. The thesis is the final product of my research, but I will write about the process of getting there in this chapter.

I embarked on this project without the exact notion of the amount of work needed to complete it. I knew I would have to work hard, but I had no idea how many hours it would take to complete the various tasks that this project required. The doctorate was to be conducted on a part-time basis, meaning that in addition to my regular working hours and all the time that this involves, I would still have to find time to develop my ideas, read, investigate and perform the tasks in the development of this thesis.

Initially, I tried to make sense of what I wanted to accomplish through reading and participating in activities carried out by the university within its constant support for graduate students. The reading followed a more technical logic in the sense that I wanted to learn how to do a doctoral thesis if that can be taught and learned. Nevertheless, I tend to run back and learn when I feel like I do not know something. I wanted to know the methodologies, the philosophical thinking, how to write, how to interview, and how to find what I did not know. Basically, learn to learn.

At this time, my involvement with the theory of practice was fundamental to understand my academic positioning within my practice and how to maximise my professional knowledge, skills and values (informal theory), bringing them into academic thinking – the formal theory (Thompson, 2017).

Part of the reading was based on what I understood to be the main themes I wanted to explore in this research, with a greater emphasis on expertise and intuition. These concepts have evolved over time within this project. I did not follow the accuracy of intuitions that were better explained by the effect of prolonged practice rather than heuristics, but I persevered in the search for solutions (Kahneman, 2011).

My involvement with qualitative research was fundamental to understand the importance of following guidelines to ensure reliability so that the result of this research is credible, transferable, and confirmable (O'Hara, 2018). In a way, a qualitative data analysis resonates with what I intended to do by explaining the point of view of my co-researchers on the various themes of this research (Miles et al., 2014). At this point in my research, I started to take reflection and reflexivity and the importance of observation more seriously and to revisit the way I thought about my own experiences and question the way I was following my own research (Salmons, 2014; Braun and Clarke, 2022, p. xxvi). This was followed by a deepening of knowledge regarding the interviews and how I would engage with the co-researchers to have a purposeful conversation that would produce knowledge based on the interaction between the researcher and the co-researcher (Salmons, 2014). A significant number of things were new to me, such as the involvement with storytelling as a means of communication (Bazeley, 2013, p. 113) and the use of language as a form of questioning (Braun and Clarke, 2022, p. xxvi); search for meaning through the specific context of each co-researcher; and finally, the importance of active listening in the interview process (Seidman, 2013).

5.1. Outline List of Questions

The process of writing down the questions for the semi-structured interview started after the approval of the research project. The questions reflected the themes I was studying and were created based on the areas I wanted to explore and gain clarity about how others perceived or experienced these phenomena. The initial thought was to have a set of open-ended questions that led the co-researchers to reflect on their practice and the way they see their profession. That could produce an insight into their process of thinking and how they execute their ideas. The outcome would inform my response to a set of research questions, but the process to get there was not defined. Through a refinement process and the help from the advisors, I drafted a set of 30 key questions (present in Appendix 16.1) that served as the basis for the interviews conducted. Not all co-researchers answered all questions, and the questions were more like a conversation guide rather than a rigid document to follow.

5.2. Co-researchers

To select the co-researchers for the semi-structured interviews, I had to find people that worked together with me in various organisations in the last twenty years. Over the years, working as a contractor, I had the opportunity to work on the best development teams in the United Kingdom. This experience allowed creating a contact base of more than three-hundred Front End programmers and seven hundred in other areas. Very early in the day, the notion that it would be not difficult to choose co-researchers for the project was present. The most significant difficulty would be for the co-researchers to accept to be part of the project and be available to grant their precious time for the research. Therefore, it would be necessary to create different selection criteria to narrow down the list to start the initial contact.

Different criteria were needed to balance out the numbers and have a wide range of co-researchers who could bring different views to the project. Initially, the most important criteria were the level of "experience", considering that expertise would be one of the variables I wanted to study. I could have opted for the creativity criterion for choosing co-researchers and searching for inherently creative programmers, but I would have had to devise a method to measure that subjective concept. The concept of expertise is also subjective in its application but concrete in its definition if you choose the right methodology. I took this task into my own hands and opted to take co-researchers who were experts based on the SFIA model – Skills Framework for the Information Age.

I want to point out that all the people chosen were programmers with whom I had experience collaborating on past projects. This means that I would be the evaluator of their expertise, the quality of their code as programmers, and the intrinsic characteristics of levels 5, 6 and 7 of the SFIA model. The different criteria produced a list of forty-one co-researchers to whom an invitation email was sent (section "16.2 Invitation Email" on the "Appendices"), with another sixteen reserve participants on a second list. The email was a simple formal invitation to participate in the project. The people who received the email were already acquaintances and work colleagues in one way or another. It is important to note that beforehand, the working methods, the code produced by the co-researchers, and their seniority level were known to the researcher, with an objective assessment already produced. Alongside the email, there was an attachment with the "Participant Information Sheet" (section 16.3 on the "Appendices"), with the Research Objectives, the Procedures, information about the processing of information and Confidentiality and the Consent Form, where the co-researchers signed the permission to the anonymous collection and use of their information in the research project. The same attachment accompanied a "Privacy Notice for Participants" (section 16.4 on the "Appendices") with standard university information on why we are collecting data, co-researchers rights under GDPR, information sharing, storage and security and retention. This document also had the contact details of the University's Data Protection Officer.

Twenty-five candidates responded positively to the email (61%), and of these, sixteen participated in the semi-structured interviews.

Each candidate is a case or unit of analysis in this study. This is relevant because I can associate characteristics to each case, facilitating the analysis and interpretation of the data and allowing the search and categorization (Bazeley, 2013). In section 5.5 ("Preparing the data for analysis"), I explain how I inserted the cases into the project.

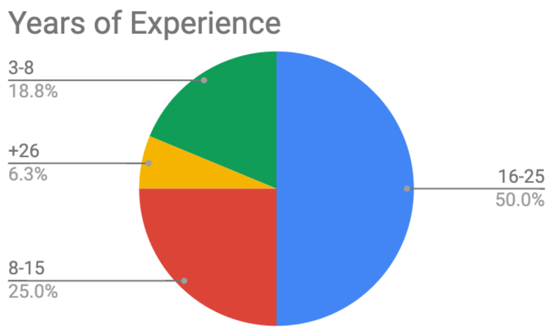
5.2.1. Demographics

I obtained the demographic information of each candidate through Linkedin[3] in the first phase. If any piece of information was missing, I filled it in with data from the personal conversation itself. The importance of this classification lies in the fact that it is possible to cross-check the data collected in the conversation with the different classifications, individually or in a mixture of the various elements. This insight will further enhance the richness of the data and its relationships.

[3] https://www.linkedin.com/in/loback/

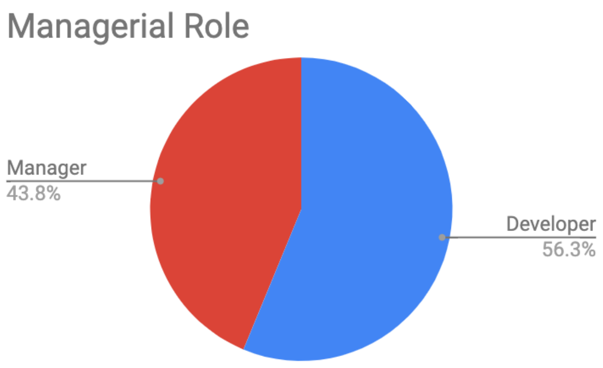
Other demographic or qualifying data, such as age, could have been asked of the co-researchers. However, the understanding was that it would be much more important to verify the years of experience in the profession than the date on which the person was born. I have worked with programming teams with a varied cultural and ethnic mix, and I mentioned this earlier. This statement is tacit knowledge and something I did not follow up on in the research. Moreover, the diversity may happen due to a lack of local labour or specific immigration in the IT area. Although I interviewed co-researchers who are immigrants and from diverse ethnic backgrounds, I did not design this study to deal with these variables, so I will not report on the topics. The same happened with ethnic minorities in the United Kingdom. It is left for future research.

5.2.1.1. Years of Experience



One focus of this research is the concept of expertise. It is the understanding that one should create an objective criterion in the selection that would allow one to count the years of experience in the candidate's field of expertise. Throughout the conversation itself, this demographic information was obtained for the co-researchers that were not well acquainted, but mainly from the candidate's Linkedin profile. It takes into account the number of years since the co-researchers graduated from university or college. As a result, 81% of respondents have more than eight years of experience as programmers counted from before the individual conversations started.

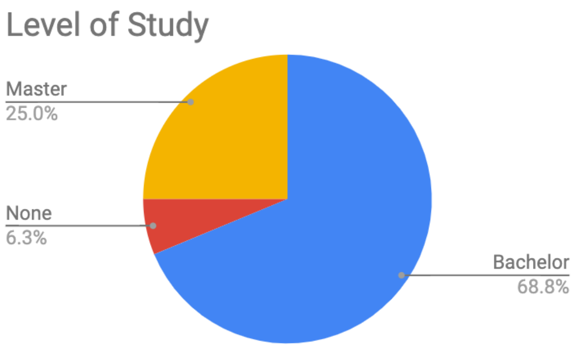
5.2.1.2. Organisational Role



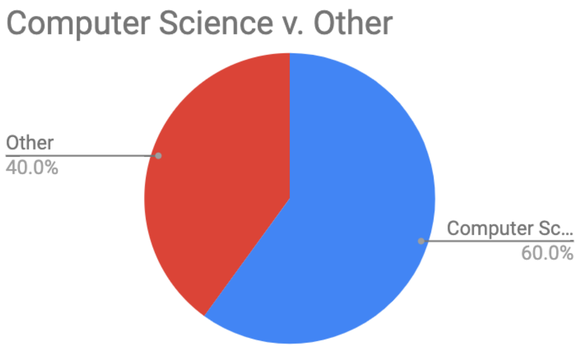
The distribution here is simple and binary: some are just developers and others are developers and managers, or they transitioned from being developers to managers. Some developers, at some point in their careers, had to manage other developers or people. The idea was to know whether there were any changes in how they program from the moment they stopped being simple programmers to start being team managers. The numbers are balanced.

5.2.1.3. Level of Study

Initially, the hypothesis was that the level of study would not affect the individual's level or quality of programming. However, that information would have to be backed up by data. By classifying the level of study, we create a new dimension of this study. It should highlight that 6% of the co-researchers did not go to university.



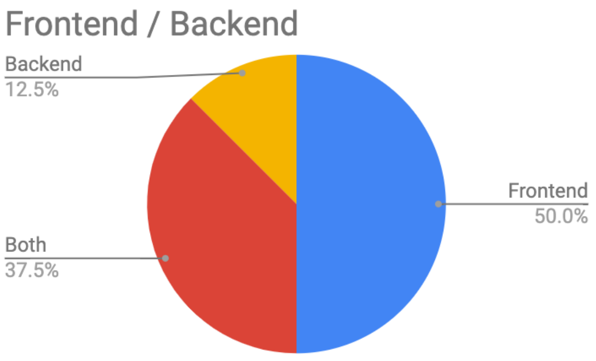
5.2.1.4. Area of Study



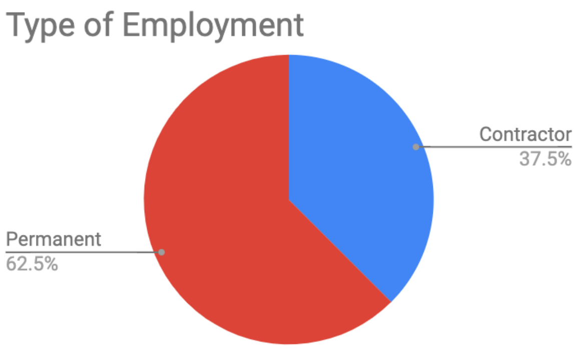
The computer programming profession is open to all people regardless of their background or their chosen courses at the university. Although the sample for this project is small to divide it into several types of courses, the decision was to make a binary division between the main course of programmers (Computer Science) and other courses not related to programming, such as Art or similar. In the small sample of this study, 60% of the co-researchers who went to university chose Computer Science as their course for their training.

5.2.1.5. Type of Developer

There are different paths that developers can take in the profession of programmers. However, for this study, the division was based only between the Frontend (more related with the user interface and the user experience) and Backend in a broader group (development of server side logic, such as APIs, and hidden interfaces with the frontend), still having the possibility of separation as a Full-Stack (Frontend and Backend). Half of the respondents were only Frontend programmers.

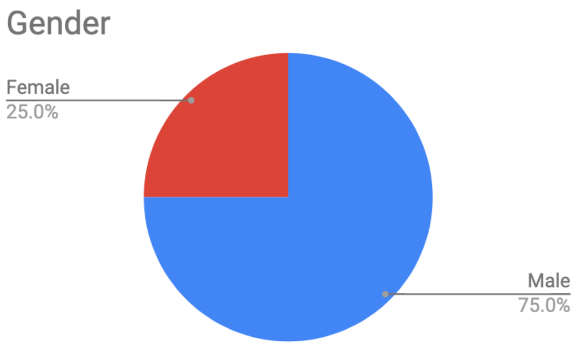


5.2.1.6. Employment Type



Permanent employees and contractors have a different view of the work when they are part of multidisciplinary teams. There is an ephemeral aspect to contract work, and companies hire them as specialists in certain areas. Therefore, there is expertise associated with the work of contractors, who are almost 40% of the co-researchers.

5.2.1.7. Gender



There is an unbalanced gender among programmers. Recruiting women was the most difficult among those selected for this project. Furthermore, recruiting is not easy because the pool of women candidates is smaller than men. Women entered the programming profession later and are still fewer in number. It will still take some time to catch up. Some women rejected the invitation to participate in the project making it more difficult. At the end of the day, and despite all the difficulties, a quarter of the co-researchers were women. It is an excellent ratio, but it does not reflect the reality of the programming teams.

5.3. Data collection

The creation and data collection process was thought out from the beginning of the project. The data was transformed into a document format to be used as an evidence source and be analysed in the project (Bazeley, 2013; Corbin and Strauss, 1998). Knowledge will be generated through data, which will be systematically gathered and analysed through the research process. There is a very close relationship between the method, the data collection, the analysis, and, eventually, the theory that will be created.

5.3.1. Semi-structured interviews

The primary data collection process was carried out through semi-structured interviews. Semi-structured interviews were deemed the most appropriate method for collecting data to answer the research questions outlined in the project. Additionally, it afforded co-researchers the flexibility to respond in a more significant latitude, in their own style and words.

> "... when we interview people in an open and flexible way, they provide us with short stories embedded within the larger narrative of their lives." (Bazeley, 2013)

The sixteen interviews occurred between March 2019 and June 2020. Initially, the interviews took place in a neutral setting in the work offices where it was possible to meet face-to-face with the co-researchers. I recorded the conversations with the mobile phone for audio-only. However, 50% of the conversations were virtual with video via Skype[4] and Zoom[5], and both programs enabled audio and video recording for later analysis. The audio and video recordings were made with the full knowledge and consent of the co-researchers. This was noted in the section 4.6.3 above when I mentioned "Informed Consent". The advantage of having recorded the conversations was that the whole structure of the conversation was recorded, and it was possible to see how the co-researchers comprehended and answered the questions, sometimes without thinking, and at other times their thought process could be understood (Bazeley, 2013).

When we talk about semi-structured interviews, it means that I previously identified the areas I wanted to cover in the conversation and that, at the same time, I would be open to making adjustments to the order and even the topics of our conversations. This implies control of the conversation on my part as I decided the direction of the conversation, what I wanted to ask, and the subjects that would be discussed. There is some flexibility, but I always have the last word about digressions or matters that I do not think will bring anything new to the research. The process is driven and controlled by me, the interviewer, who decides what to ask and what is discussed.

The initial interviews were face-to-face. The benefit of conducting the interviews in person is the smooth interaction between the researcher and the co-researcher. Interjections are perceived differently, and there are nuances in the voice and gestures that make the whole process more dynamic and pleasing. Nevertheless, there was no video captured for further analysis.

With virtual conversations, scheduling was the easiest part. It is more uncomplicated to check if the co-researchers are available or not at the time. It is also possible to stop and return at another time. Besides that, the later analysis is more straightforward because the conversations also contain video and audio. During virtual interviews, the silence was not well tolerated. It is more challenging to leave moments of silence and pause in the conversation for the co-researcher to reflect on and later on dwell on the response. Technical problems were also a constant on the part of the co-researchers, with poor internet connection being the most common. The COVID-19 pandemic was also another factor in making virtual interviews happen.

[4] https://www.skype.com/en/
[5] https://zoom.us/

The conversations lasted for one hour. The first ones took a little longer as I got used to the methods, processes, questions, and topics covered. As the interviews progressed, time and flow control became more internalised.

I stored all the conversations anonymously with a case number and a key master list stored in another place. The suggestion was to anonymize all data from the moment you start recording files on the computer (Bazeley, 2013). Everything was backed up in the cloud and at different computer locations to preserve its integrity.

The entire interview process was a healthy exchange of knowledge between what I was doing at the time as a programmer and my future as a researcher. I was already used to meetings for gathering software requirements; on the other hand, I also strictly followed written requirements set in stone. I learned from the interviews that the interaction with the co-researchers leads to an iteration over meaning that is only possible in a conversation and subsequent recording. The richness and nuances of the conversations would not be the same if the answers were given in writing since conversations around the questions are as important. Semi-structured interviews allow the researcher to ask the co-researcher to better elaborate the answer in certain aspects in order to take away the meaning that the co-researcher wanted to pass on, rather than a pre or post-conceived idea by the researcher. This knowledge I brought back to my practice, preferring meetings/interviews to gather requirements instead of just following previously written documentation as a basis for requirements. The recordings made it possible to go back to verify the meaning given to an answer to a given question. When many people started working remotely because of covid, between 2020 and 2022, this lesson became more relevant.

Another idea that has been assimilated both ways is iteration. This concept is widely used in Scrum / Agile development (Cohn, 2010) and served as the basis for creating the initial questions, taking what was done in previous conversations, which were then improved along the way in later interviews. Here, as in my practice, adaptability was key to knowing what to change and always taking into account the lessons learned in previous conversations.

What did not go so well was the timing of the initial interviews. I was indeed learning, but it took me two or three interviews to synchronise with the pre-set time of one hour. I also had some difficulty getting the appointments right for some of the conversations with which there was always a clash on the agenda. I was very disappointed for not being able to make all the conversations in person, which I am sure would have been more pleasant, but in a pandemic season, I also recognize that the threat of COVID helped me have more slots available to interview remotely.

Looking at the final interviews compared to the first ones, I can say that I went through a learning process. Initially reluctant to start because I thought there was a lack of technique or knowledge until I felt completely comfortable with the co-researchers and the direction given to the questions and my project. I thought I would be much more incisive in my questions trying to get absolute answers, but in reality, the answers were much more subjective and dependent on individual perspectives, and the experiences lived by each one. The interviews were much more unstructured than I was initially prepared for, and challenging my assumptions was part of this learning process. In terms of evidence, I thought that experts' words were more valuable than non-experts', but what happened is that I started to value personal experience much more and the way in which each person approached situations within the context of their work than those considered to be the most experts.

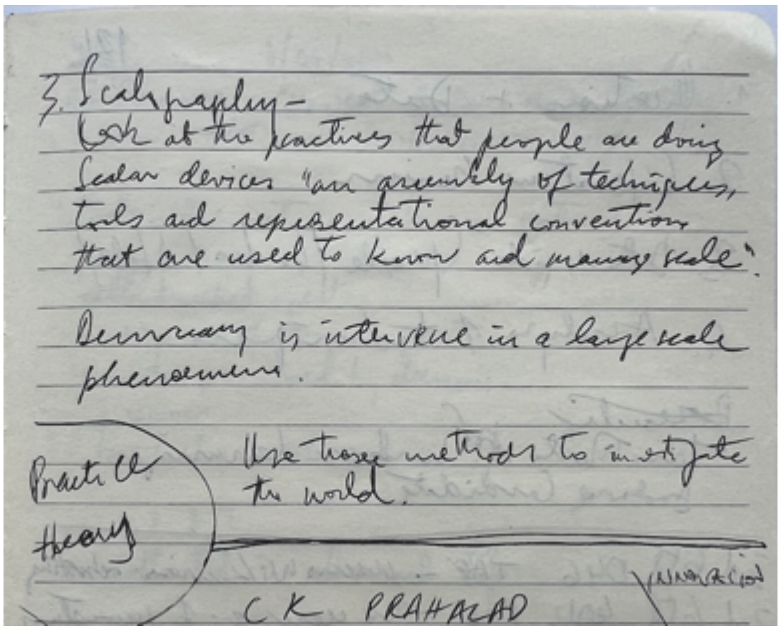
5.3.2. Other Data Sources

The interviews were not the only source of data for this project. Before starting, I already knew that I would use other materials as resources for my final project. The way I organised the whole process helped in the way it could be retrieved and analysed in the future. This follows the best practices on creating a data storage system enumerated by Bazeley (2013).

5.3.2.1. Cloud storage[6]

From the beginning of the project, I put in practice a system in the cloud that allowed the storage of data with differentiated access. I tried to create a system that would allow me to take notes or write on different devices and in distinct situations. I wanted to record audio notes on my phone while driving; take notes of my readings on the phone or tablet, whether they were books, articles, newspaper articles or thoughts or ideas that came to my mind; increase the content of my literature review while sitting at the computer; taking notes of meetings with my advisor; write my thoughts in the seminars I attended. In short, I wanted to create different types of documents that could be accessed, changed and augmented on different platforms and places in an organised, always available and usable instant data record. Besides availability, an essential factor for me was "search". Concentrating on digital files made it possible to search to find words, phrases, and authors in all the digital sources produced. When I took notes on paper, I scanned the image, added keywords and created a new document.

[6] "Cloud storage allows you to save data and files in an off-site location that you access either through the public internet or a dedicated private network connection. Data that you transfer off-site for storage becomes the responsibility of a third-party cloud provider. The provider hosts, secures, manages, and maintains the servers and associated infrastructure and ensures you have access to the data whenever you need it" (IBM, no date)



3. Scalography –
Look at the practices that people are doing
Scalar devices "an assembly of techniques,
tools and representational conventions
that are used to know and manage scale".

Democracy is intervene in a large scale
phenomena.

Practice
theory

Use those methods to investigate
the world.

C K PRAHALAD
[illegible]

Figure 4: Excerpt of a handwritten note

Transcript of a handwritten note:

"Scalography —
Look at the practices that people are doing
Scalar devices "an assembly of techniques,
tools and representational conventions
that are used to know and manage scale".

Democracy is intervene in a large scale
phenomena.

Use those methods to investigate
the world." — Loback (2017)

Microsoft OneNote[7]

Among the many note-taking software I have tried, Microsoft OneNote was the one I liked the most because it was possible to organise notes, highlight text, write, record audio, and insert other files using mixed media. The fact that the notes are searchable makes the search easier since the amount of information started to grow, and, due to my work, I would have to write down my ideas in order to be able to continue following that line of thought after hours or days later. The information was stored locally and synchronised in the cloud, meaning you could open the software on different devices and distribute the same information in different places. The versatility of Microsoft OneNote lies in its resemblance to a physical notebook, and visually this helps to understand how to organise and find information after it has been stored.

Google Docs[8]

Half of my notes were created using Google Docs. The best features are the extraordinary ability of the search engine (which is the same as Google Search) and better formatting of the text. Furthermore, it also works in the cloud and offline. What I was looking for when I started using Google Docs was the ability to open and edit the same document on different devices. Opening the document in different media allowed me to start a document on my computer at home; edit and add information on my commute to the office from my mobile phone or tablet; open the same document on my work computer, add extra information;

[7] The Microsoft OneNote is a digital notebook developed by Microsoft and is available free of charge. Allows the addition of fluid content linking conversations, texts, images, etc. (Microsoft, no date)
[8] Google Docs allow the creation and online collaboration on documents in real-time from any device (Google, no date b)

review information on mobile phone in my commute home; and finish writing or editing again on my home computer. One of the most popular features of Google Docs is the ability of several individuals to collaborate on the same document. I was not cooperating with other people but with myself on different devices and locations. In terms of organisation, it is not as good as OneNote, but it exceeds the competition for text notes. I used more text formatting and wrote down my ideas as they came up.

Along with the conversations, I ended up having hundreds of electronic notes in various formats on the various subjects and ideas that made up my research. Maintaining discipline in this field helped save a lot of time in the future, as the work was adequately catalogued and organised (Bazeley, 2013). I have always taken into account data security and confidentiality concerns as attained from Middlesex Online Research Ethics approval of my research (https://moreform.mdx.ac.uk/).

I will summarise the things these software did for my research:

Collaborative writing with myself: These tools allowed me to edit documents in real-time, in different places and devices. This meant that I could work on the same document simultaneously in different places and devices, increasing productivity and the speed with which I could gather information in a centralised document.

Cloud-based storage: These tools are cloud-based tools, meaning that all documents are stored online. I had the advantage of being able to access documents from anywhere, anytime, as long as there is an internet connection. They also allow for local storage for when there is no internet connection, for example when travelling by plane.

Multi-device syncing: Because they are cloud-based, they can sync across multiple devices, allowing you to work on your documents from your desktop, laptop, tablet, or phone, and always have access to the most up-to-date version of the files.

Integration with other tools: allows integration with other editing tools such as Microsoft Office or Google Workspace. This feature allowed creating documents that could easily be imported into other tools and include various types of media, such as images, videos, or graphics.

Search functionality: This functionality is central to research, since there is a need to find specific texts stored in hundreds of documents, and these tools offer powerful search functionality, allowing you to quickly locate specific information within your notes or documents, even if you have a large number of them.

These tools allowed me to see things that I would not otherwise be able to make the same connections with. The way I usually work and organise information was crucial for me. Here, there is also the interconnection between my work and my research, in which knowledge is fluid, moving from one side to the other. From my work, I bring to the research my methodical thinking and the ability to organise information that was useful to me in the research project. From my research, I bring back to the workspace all critical thinking that includes reflection and these tools that were used and tested in a research environment, which will last in my future career.

5.4. Transcribing

The conversations produced 163,806 words that all need transcribing into hard-copied text. There are, on average, 10,237 words per conversation. I went through several phases of discovery until finding the ideal setup for the transcription. Both video and audio recordings produced high-quality outputs that made it possible to understand everything the co-researchers said clearly. I initially decided that I would do my own transcribing without using a third party since there is real value when transcribing it yourself because it is another way of knowing the data in a more intimate way (Bazeley, 2013). The first phase was to try to listen and transcribe simultaneously in a straightforward way. Soon the realisation comes that there are not enough hands to type and press the button to start and go back at the same time. It was starting to take a ridiculous amount of time to do this job. The second phase involved purchasing software that would assist in the listening and writing process, automating some features, such as hotkeys for play, stop, go back five seconds or automatic stop. At this stage, the software used was the “Express Scribe Transcription” of the Australian company NCH Software (NCH Software, no date). Even with some automation, the transcription was still taking a long time. The third phase was the discovery of artificial intelligence and transcription software called Otter (Otter.ai, no date). Otter helped move the project faster. The process involved importing the audio file into the online software, which instantly transcribed the audio to text. After the work of the artificial intelligence, a manual text revision was performed since there were words that the software could not catch the specific meaning and phrases that needed adjustment. As an example of this, we have the word “code”, which most of the time it appears as “cope” or something similar.

> “The goal in transcribing is to be as true to the conversation as possible, yet pragmatic in dealing with the data.” — Pat Bazeley (2013, p. 73)

In short, the entire text needed revision to make sure that the co-researchers said what they wanted to say, in the way of their thinking, and without any uncertainty about their intention.

Some critical decisions I made in the transcription process were: I left all the interjections; I did not correct incomplete sentences; I included all the digressions, and the entire conversation was included from start to finish; I also included notes and pauses.

The transcription took the entire interview period (between March 2019 and June 2020) plus another three months.

5.5. Preparing the data for analysis

Before moving on to the thematic analysis of the conversations and other sources, it was necessary to format the text for insertion in the software I chose to help me with the thematic analysis. I understand that in this time and age, we should embrace the technology available to us, but at the same time, recognize that the computer is not doing the work for us. Maybe it is helping, but the heavy lifting is on the researcher. I used the software that was available at my university, called nVivo – a qualitative data analysis software that helps researchers organise, classify, analyse and find relationships in unstructured data. Linking, shaping, searching, and modelling can be added to the analysis (QRS International, no date).

Each candidate was entered as a single case in the nVivo software going from "Case#1" to "Case#16". This process served to add the classifications, as explained in the section 5.2 and also allowed the evaluation, interpretation, and reliability of the criteria that I chose for the sample. As a bonus, we have the possibility of triangulating data in the analysis, and this information is critical when used to interpret the results and attest to the meaning and relevance of what was found throughout this project (Bazeley, 2013).

5.6. Thematic Analysis

I found it necessary to become familiar with the texts to be analysed to start the thematic analysis process (Braun and Clarke, 2022, p. 42). The familiarity with the texts started right after the conversations. After each conversation, a reflection for each co-researcher was written, containing the first impressions about the candidate, the answers given, and the interview process. Reading the data "is one essential, basic analytic technique" (Guest, MacQueen and Namey, 2012, p. 21). I read, re-read, and listened to all the conversations, taking notes when what I saw or heard caught my attention. (Bazeley, 2013). From these notes and reflections, I draw inferences about the evolution of conversations and the feedback obtained from the co-researchers, and my evolution as a doctoral researcher. The transcription process itself also served as support for the initial thematic analysis and the familiarisation with the data.

"At a very basic level, Thematic Analysis is a method for developing, analysing and interpreting patterns across a qualitative dataset, which involves systematic processes of data coding to develop themes – themes are your ultimate analytic purpose." — Virginia Braun and Victoria Clarke (2022, p. 4)

Although the process is systematic, each person analysing the data will come to different conclusions regarding the themes, as the result is subjective. What I intend to do here is to give an explanation of what the thematic analysis process means to me and the way I interpreted the data.

5.6.1. Coding

After inserting the conversations into the support software, I started the coding process, whose main objective was to prepare and search for stimuli that generate themes for further data analysis. The code is a representation of the data. For me, as a computer programmer, code means that but also something completely different related to a set of computer program instructions that will render some effect in a given situation. My understanding of the code in the Thematic Analysis setting is like a "label" that marks a given text, paragraph, word, image or other media and allows later access to the information that has been placed on that specific label throughout the dataset. Braun and Clarke, for example, use "code" to define the code label and the code itself (Braun and Clarke, 2022). Costley (2010) talks about identifying "key categories" that summarise what was said or what was observed.

"Coding provides a means of purposefully managing, locating, identifying, sifting, sorting, and querying data." (Bazeley, 2013)

My focus was on entire sentences and paragraphs, with the aim of preserving the integrity of the text and the flow of the narrative (Bazeley, 2013). In this sense, the code is a representation or a reference of the paragraph, but at the same time, an access point to the information contained in that same label.
Coding is not an objective in itself, but it helps define themes in the following steps for further analysis. The aim is to identify themes and patterns of meaning between datasets in relation to the study questions as described in the "Aim and Objectives" in section 1.3 above (Braun and Clarke, 2013).

In the literature review, I noticed that there is an overlap between coding and theme generating, and it seems to me that the authors use the two terms interchangeably mainly because the labels that are produced with the creation of code will be organised in themes and subthemes. But it is important to note that codes are not themes, as Braun and Clarke pointed out in their recent book (2022). Coding is a systematic process and code labels are the visible part of this process (Braun and Clarke, 2022, p. 53).

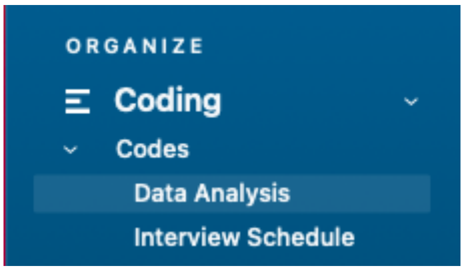


Figure 5: Print-screen from nVivo with the top-level folder for coding on this research project

As I was using software to help create code for further analysis, I wanted to organise my labels into two groups.

5.6.1.1. Data Analysis

This is the main node of my research. It contains all the coding that was done in all the conversations. My initial coding process revealed 168 nodes of labels and sub-labels. Thirty-six labels were in the root, and the others were sub-labels. Initially, I only created one sub-level in the nodes. I wanted to finish all the conversations before creating more specific node clusters and, perhaps, start the organisation by themes. In a way, it was necessary for me to abstract myself from thinking about creating themes while I was still in the coding process (Braun and Clarke, 2022, p. 54).

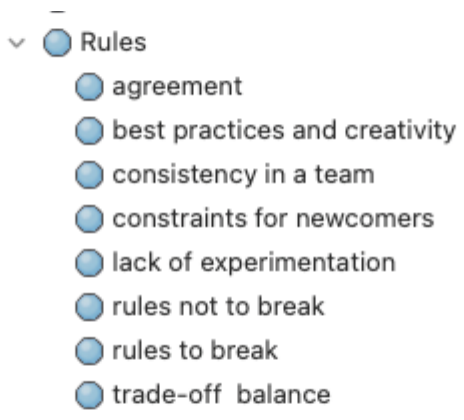


Figure 6: Image showing the top-level label "Rules" with the sublevels in nVivo

The coding process evolved as I moved on to coding the conversations. Sometimes I had to go back to previous conversations to change or add code I created in the recent conversation. Braun and Clarke noticed that it is not a bad practice to change, modify or expand the code during the process because, as the project progresses, it is necessary to expand the meaning of a label or notice a new meaning that you want to give to a specific code. In a way, I am interested in not only demarcating differences but finding insights in common with all the co-researchers (Braun and Clarke, 2022, p. 55).

As I was creating the labels, I was writing the idea behind this code in the description of the label properties. Writing a description helped me make the links between the codes and, later, do the grouping in themes and sub-themes. Several times I went back, revised, changed and added more information to the labels. Other times, I split, proofread the texts, and split a code in two. Writing the description helped not to repeat labels and decide if the new text to be coded would fit an existing label or if it would even be necessary to add a new label.

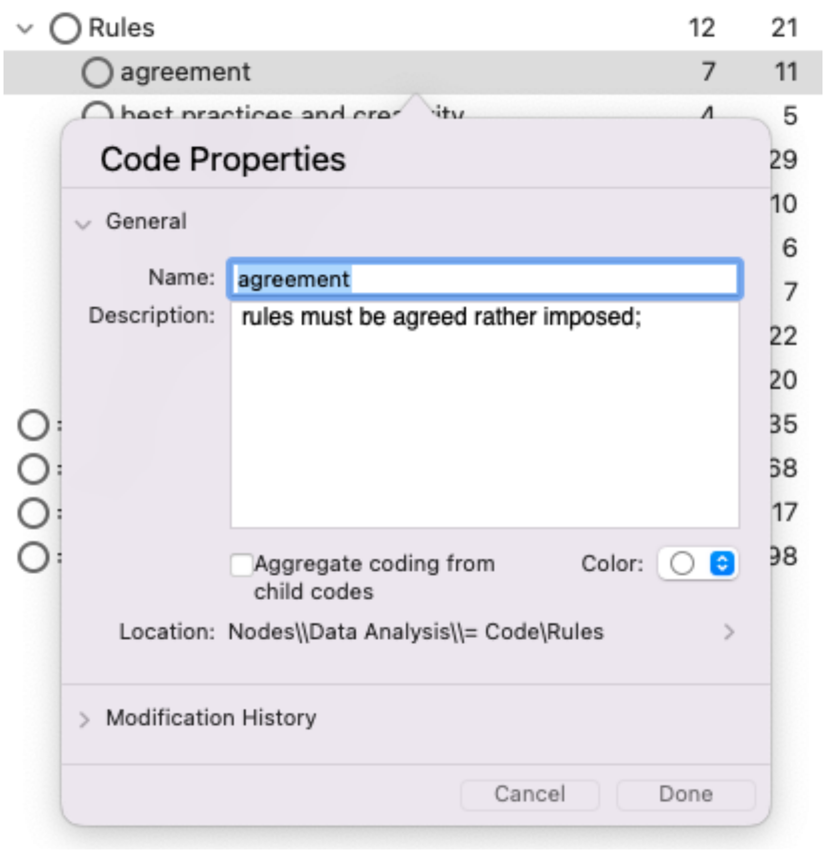

Rules 12 21
agreement 7 11
Code Properties
General
Name: agreement
Description: rules must be agreed rather imposed;
Aggregate coding from child codes
Color:
Location: Nodes\\Data Analysis\\= Code\Rules
Modification History
Cancel
Done

Figure 7: Image showing the label description on nVivo

5.6.1.2. Inductive Analysis

The generation of labels was created in an inductive way using the data as a starting point of the analysis. Basically, I was looking for perspectives, experiences, and understanding of what the co-researchers wanted to say. From the moment the co-researchers articulated their experiences in relation to the questions that were addressed in the conversations, I tried to capture these concepts in themes for my research. Braun and Clarke (2022, p. 56) argue that it is impossible to engage with the data in a purely inductive way because, as researchers, we bring the baggage of concepts, ideas, and perspectives that affect the way we develop our code. Also, it is worth remembering that I was the one who organised and conducted the interviews, so I already knew what they were about.

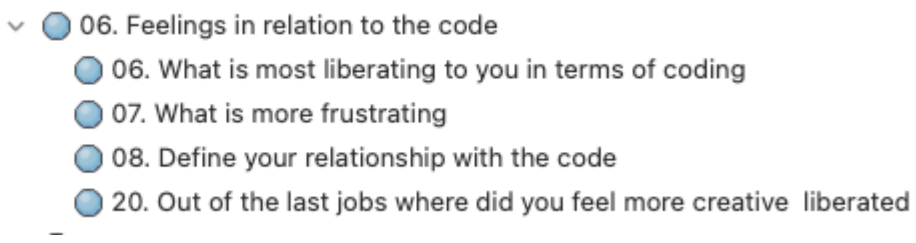
As an example, we have the "arts & crafts" label that was created to encapsulate the idea of the intersection between technology and art that the co-researchers mentioned. This label was accompanied by a note with a personal reflection on photography as a technique and art and a quotation from Schön (among other paragraphs):

> "If it is true that there is an irreducible element of art in pro-fessional practice, it is also true that gifted engineers, teachers, scientists, architects, and managers sometimes display artistry in their

day-to-day practice. If the art is not invariant, known, and teachable, it appears nonetheless, at least for some individuals, to be learnable." — Donald Schön (1994, p. 18)

5.6.1.3. Interviews Schedule

The code in this node was an exercise to retrieve all the answers that the co-researchers gave about a certain question or subject. The question script was not rigid, but everyone answered more or less all the topics. Please see section "16.1 Questionnaire" in the "Appendices" to view the guide. As an example of the type of organisation we have:



06. Feelings in relation to the code
    06. What is most liberating to you in terms of coding
    07. What is more frustrating
    08. Define your relationship with the code
    20. Out of the last jobs where did you feel more creative liberated

Figure 8: Group of questions numbered as 6, 7, 8 and 20 within this idea of "Feelings in relation to the code"

Clicking on each of the nodes shows the set of all responses from all co-researchers to a given question. In this specific context, clicking on the top-level "06. Feelings in relation to the code" gets the result of all other "children" questions of this node.

5.6.1.4. Code creation

In the next paragraph, I will explore in detail how a code originated, how it was developed and how it was revised during the project.

Initially, I started by exploring Case#1 in detail. I added several initial codes as I read on until I came across the following excerpt from the conversation:

> "When you've been building the same thing over and over again – and even though you may have to write new code, sometimes it just bores me, and sometimes – what happens is, when I am bored – if I get bored and I complete things quickly, what happens is my mind almost like it – I am not interested in it anymore, because I don't have … I feel like in some ways, sometimes I need that kind of pressure to keep me going, and because you have these 2 weeks sprints, the work that you are working on may not be that interesting, all of a sudden I just get frustrated, and I am not interested. It is … it happens." (Case#1, 2019) Answering a variation of Question#7

There were several things here to analyse in this answer. In general, one can see that the co-researcher is trying to express feelings, and starts a reflection on the situation, explaining why it happens, when it happens, and, in a way, even presents what would be the ideal situation. The co-researcher is not talking about something external or a pseudo-problem but about their practical reality, lived on a day-to-day basis as a programmer. We can clearly perceive the frustration felt by the programmer. I realised at this moment that the entire text should be included in a “frustration” code. When creating the new code, I took the opportunity to add the description “frustrations felt by programmers”. At this stage, I still did not know what to do with this code, if I would split it, promote it, or delete it in the future. With the first code closed, I continued my analysis. In fact, the text begins with the cause of the programmer’s frustration: the repetition of the same tasks and the reality of having to do the same thing over and over again. “Code repetition” would be a good candidate for a new code, but I felt it did not capture the essence of what the person wanted to say at the time. I understood that the frustration came from the fact that the person was “bored” with the situation. I created a code that I called “boring” because I realised that the most significant criticism that the person was making was towards the agile method and the two-week sprints, where the tasks are assigned to all the programmers, and they have two weeks to complete the tasks. Our programmer did the tasks, which were already repetitive, and managed to complete them, and later lost interest in the work. The solution to the frustration is to increase pressure in terms of work and bring in new challenges.

5.6.1.5. Code change

I would like to demonstrate below what was the process used to change this code created with the text of Case#1 compared to the analysis of an excerpt from Case#2. That way, you will have an idea of what judgments I used to create my codes, how I generated my code and themes and organised everything in the end.

> “Hmm, so, often I got some opinions about how things should work and, if there is an impediment, or if there is a manager who says, just ‘Shut that down’ without any guidance around it, you know, I don’t like that part. But I do like that part of exploring that, and if they don’t give me any avenue to explore my own personal ideas then it is frustrating and I just feel … boring (laughs).” (Case#2, 2019) Answering a follow-up question from Question#4

In the text excerpt from Case#2 I have again the “frustration” code that I added to the whole text. The reason is that there is frustration with managers and their lack of emotional intelligence and not knowing how to deal with programmers. This is associated with a “bad system”, the opposite of “code for fun” referred to elsewhere. There is also the desire to “create something for myself”, and in the end, we return to the issue of being “bored”. Having a “bad system” as one of the reasons for frustration, at this stage, I understood that “boring” was also one of the causes of “frustration” and decided to promote the "frustration" code to the parent of the "boring" and "bad system" code. This was in line with what I had already found in Case#1 above. The "frustration" code would be an aggregator of several reasons why programmers feel frustrated in developing their work. At some point, I changed the code from "boring" to "monotonous – boring" in order to better portray the reality that I found throughout the interviews.

My coding process is appropriate for my research because I consistently pursued the co-researchers experiences regarding creativity in programming. There are several ways to see the data. My code reflects my insider-research approach and perspective. All observations involve interpretation. On the one hand, my interpretation, and on the other hand, the interpretation of the co-researchers, who also provided the data.

5.6.2. Themes

There is a general idea that themes will appear as if by magic. The 'emergence' of themes and theory depends entirely on the research's work with the available data, whether on paper or software (Bazeley, 2013; Braun and Clarke, 2021, p. 230). Thematic Analysis is more than counting words, making cognitive maps or creating semantic networks and focuses on identifying and describing both implicit and explicit ideas contained in the text (Guest, MacQueen and Namey, 2012, p. 10).

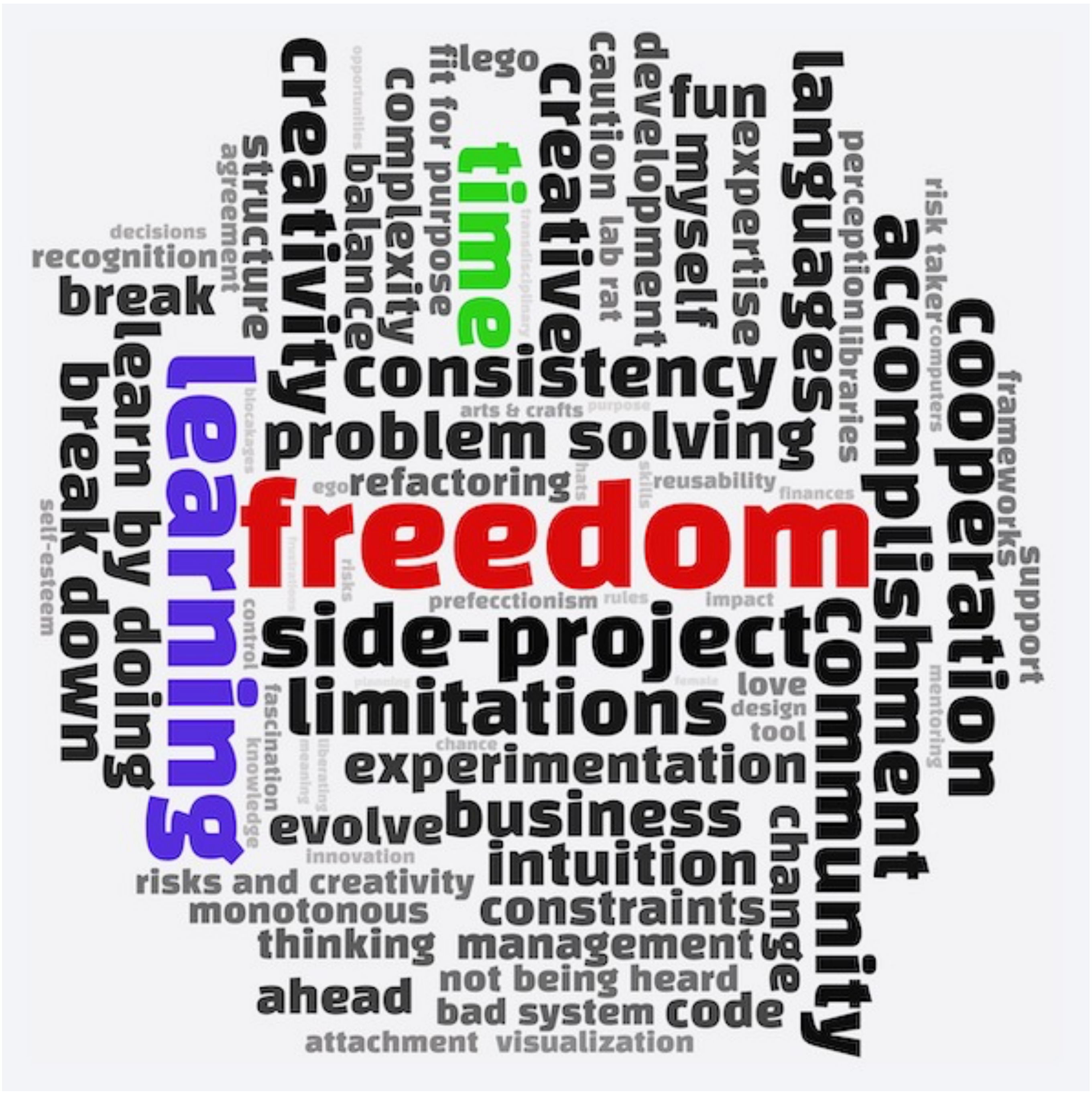


Figure 9: Word frequency cloud matrix

5.6.2.1. Themes Development

Initially, I had an idea of the themes I wanted to analyse, and these were studied in the preliminary literature review. Together with the preliminary bibliography and the doctoral research project, these themes were presented to the PAP – Programme Approval Panel in April 2018. At this stage, the areas I thought would be interesting to investigate, even before doing the interviews, were: Complexity, Creativity, and Innovation. These themes followed my line of thought at that time and formed the basis of my project. For the themes to begin to appear, it was necessary to take notes as I reread the conversations and ideas emerged. This is a lengthy process in which patience and perseverance are required on the part of the researcher. The process of creating the themes takes time because they are the result of the researcher's creation of working on the data, immersing for a long time. With a large amount of text in the conversations, it became necessary to read and reread all the data several times, and each time I looked at the text, I could see and find something different. New interpretations were created that gave rise to new concepts. Concepts are the building blocks of theory. By creating code, I was able to identify these concepts. New concepts have properties that give them meaning and also several dimensions that explore the range of variation of properties.

In the search for themes, I looked with particular attention to the work developed by Guest, MacQueen and Namey (2012, p. 66) with some thematic or linguistic tips for identifying themes. Among them, we have "repetition" as one of the most recognized techniques, which alone is not enough for coding, but which, together with the relevance and objectives of the research, can help. For example, we have the concept of "creativity" mentioned several times by the co-researchers, which was coded as such. We have several properties such as "creative side", "code is creative", or "experimentation", which may vary depending on the situation or co-researcher. This variation, which allows having a range of dimensions from the properties, is what produces the building blocks of my research . Another idea for generating themes comes from the "metaphors and analogies" in which co-researchers refer, for example, to "Lego building blocks" when talking about the current state of programming. Overall, I was looking for patterns, storylines, plots, causality and relationships, and this is usually achieved by asking, "What is the co-researcher saying here?" and directly connecting with the objectives of the analysis in this research and the research question. In this way, I defined the themes based on the data (Guest, MacQueen and Namey, 2012).

> "The really interesting data are in the variance and not the mean." — Kate MacQueen (2012, p. xvii)

5.6.2.2. Themes Review

The development of the themes took place in an iterative way. While ideas and concepts were being developed and clustered, there was always a review process. A revision that either made small arrangements and rearrangements or changed everything completely to create an entirely new version. The themes were grouped and analysed in the next chapter among the various iterations.

The section "16.5 Codebook" on the "Appendices" has a complete list of the codebook and how the codes and themes are organised. Here is a list of all the initial nodes (some were sub-nodes).

Table 3: List of all codes generated in the project

1 : achievements	56 : Intuition	111 : most liberating
2 : code impact	57 : lack of time	112 : create something for myself
3 : just did my job	58 : limitations to expertise	113 : create something new

4 : national impact	59 : perfectionism	114 : freedom
5 : no impact	60 : pointing towards being an expert	115 : good communication
6 : personal lasting impact	61 : wear different hats	116 : lab rat
7 : teaching	62 : fascination about the first lines of code	117 : love the code
8 : worldwide impact	63 : female developers	118 : mix of personal interest and work
9 : arts & crafts	64 : formal education	119 : recognition
10 : balance	65 : frameworks and libraries	120 : variety of tasks
11 : code	66 : add complexity	121 : other related skills
12 : attached to the code	67 : change and evolve	122 : practical skills from university
13 : code as a tool	68 : fit for purpose	123 : purpose and meaning
14 : code community - cooperation	69 : limitation	124 : Reason for becoming a developer
15 : code for fun	70 : saves time	125 : accomplishment
16 : code is not creative	71 : structure	126 : Become a developer by chance
17 : code perception	72 : time consuming to learn and keep up	127 : coded before university
18 : confused	73 : frustrations	128 : Control the Outcome
19 : detached from the code	74 : bad code or language	129 : design background
20 : learn by doing	75 : bad system	130 : financial reward
21 : lego	76 : blockages	131 : influence
22 : reusability	77 : copy paste code	132 : interest in computers
23 : structured	78 : Dogmatic	133 : interest in design
24 : supported	79 : management	134 : solve problems
25 : complexity	80 : monotonous - boring	135 : video games
26 : amount of knowledge	81 : new things and no knowledge	136 : Risks
27 : refactoring	82 : no coding	137 : caution
28 : speed of change	83 : people	138 : risk taker
29 : Continuous Professional Development	84 : stuck on permanent job	139 : riska as a way to learn
30 : Creativity	85 : time	140 : risks = bad
31 : code is creative	86 : wasted time	141 : risks = good
32 : code is not creative	87 : future	142 : risks and creativity
33 : Create space for creativity	88 : grab opportunities	143 : risks at the research phase
34 : creative side	89 : Innovation	144 : time constraints
35 : experimentation	90 : Innovation as a choice	145 : Rules
36 : lack of creativity	91 : inspiring	146 : agreement
37 : Limitations to Creativity	92 : positivity	147 : best practices and creativity
38 : need time to be creative	93 : Learning	148 : consistency in a team
39 : decisions	94 : limitations - constraints	149 : constraints for newcomers
40 : ask other people - counselling	95 : challenge limitations	150 : lack of experimentation
41 : break down to smaller parts	96 : hardware	151 : rules not to break
42 : business value	97 : limitations foster creativity	152 : rules to break
43 : deep thought	98 : management	153 : trade-off balance
44 : gather information	99 : own limitations	154 : self-learning
45 : planning	100 : working with others	155 : side project for learning
46 : try different approaches	101 : low self-esteem	156 : think ahead of the time
47 : understand the code	102 : manager	157 : Transdisciplinary
48 : Visualization - Imagination	103 : become a better developer	158 : women
49 : Expertise	104 : change in terms of code	159 : advantage

50 : Expertise - Drawbacks	105 : dictator - big ego	160 : fair treatment
51 : not being heard	106 : interested in people	161 : fit in
52 : don't believe they are an expert	107 : mentoring	162 : no difference
53 : experts take the leadership	108 : no change in terms of code	163 : opportunity to prove themselves
54 : exposure to different programming languages	109 : organisational skills	164 : work for free to get experience
55 : exposure to other people	110 : troubles transitioning to manager	

While doing this thematic applied analysis, I reflected on my thoughts, feelings and experiences related to this process. The first aspect is related to my personal biases. As I became more aware of the data, I became more mindful of my assumptions and preconceptions, leading me to better understand how my worldview influences my thoughts and behaviour when choosing topics. My thematic decisions have biases and are also subjective. Moreover, I had to learn to live with it. By better analysing the information shared by my co-researchers, I became more aware of the shared and identified experiences with each of them that I have. I found myself more involved in the role of inside researcher and active participant. I understood their dilemmas and shared their difficulties, which opened my eyes to solutions I had not considered before.

5.7. Writing up

Writing down thoughts solidifies ideas and clears thinking. Writing is part of the research, and it took me a long time to realise that and start writing. It was not for lack of incentives from the supervisor but due to insecurity and feeling that I had to have something concrete to be able to start writing. I was wrong. The hard part was getting started.

Gillie Bolton and Russell Delderfield (2018, p. 135) explain the importance of reflective writing and how it can be used to expand professional development as well as personal development. This reflective process helps bring our thoughts, feelings and actions to light. As a result, writing becomes an essential tool to increase reflection and as a way to learn more about ourselves and our practice (p. 138).

> "Reflective writing is the reflective process, rather than recording what has been thought." — Gillie Bolton and Russell Delderfield (2018, p. 135)

The more I write, the more my horizons open to new paths and the more information I have to write about. It is a constant exercise that I had to learn to use and persevere. This perseverance would only come with the regular practice of writing in order to take the most significant benefits from the reflective process (Bolton and Delderfield, 2018, p. 140). The way to finish the project was to write a little bit every day. The next day, erase, change, twist, squeeze, but continue until the final project is finished. The text changes as you write. The ideas were already inside my head. I had to force them out, and as I wrote more, more and more ideas came. Writing helps the mind to think. Furthermore, the mind thinks better while one is writing. Getting into flow in writing means spending time putting words on paper, and they take their own shape, bringing to light what is on my mind.

It took some time to come up with all the arguments I would like to use to discuss my ideas in this project, and the writing process was crucial in clarifying, consolidating, expanding and organising all the ideas.

5.8. Summary

In this chapter, I wrote about the activities carried out in this project, expressed as a form of reflection, where it was possible to see how this project affected my global vision and how I was changing as I engaged more with the data and with the interpretation of the same. This was also the chapter in which I justified in practical terms the decisions I made throughout the project, from the alignment of questions to the choice of co-researchers (with demographic data), data collection, transcription, data analysis, thematic analysis with coding, and the writing of the thesis.

In the next chapter, I will analyse the data collected.

Part 3: Analysis

6. Analysis Overview

In this part, you will find how I make sense of the data I have collected throughout this project. You have the discussion and analysis here, as well as the findings. In the chronology of this project, this part was developed at the same time as the 'Knowledge Landscape' (chapter 3 above) because as the themes emerged here and took shape in this analysis, I would investigate what was the knowledge landscape of these broader themes and bring this information back to my analysis, in the chapters on this part.

The data analysis was done with creativity and programming in mind. I will analyse six aspects that were identified based on the conversations with the participants: creativity, code and programming, expertise, innovation, learning and development and finally, the impact.

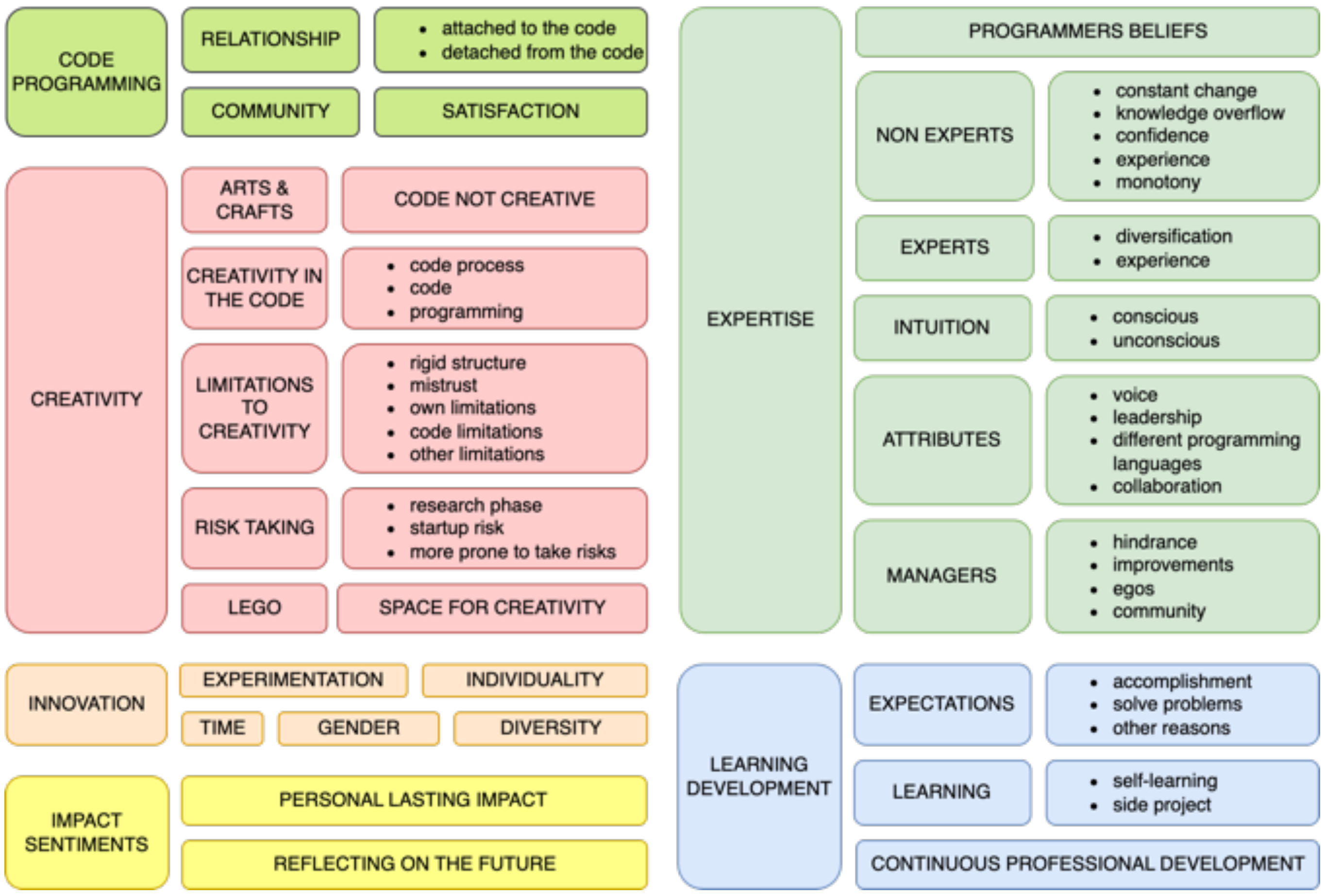


Figure 10: Summary of the themes with sub-coding

7. Creativity

This project's main idea is centred on programmers' creative ability, especially when placed in parallel with expertise. Creativity is one of the themes that emerged from the conversations and from the collected data. It is interesting to know to what extent there is the possibility of being creative in producing lines of code, to understand the limitations and impact creativity has on the programming result – the software. This theme is part of the research question that I want to solve with this project.

7.1. Arts & Crafts

The first question that needs to be explored is whether programming is an art or a craft and why this is relevant to this research. This question was developed with the co-researchers of this project, which I will answer later with the co-researchers' insights.

Arts & Crafts was a movement that emerged at the time of the industrial revolution in the United Kingdom and was based on strong criticism of artistic decadence and the lack of quality that came from industrialization. The movement and its practitioners had a negative view of machines that led them to put hand work as the epitome of quality as opposed to machines that were soulless, repetitive and inhuman (Crawford, 2002).

Defining what art is has been a philosophical question that has been debated in recent years with no apparent consensus. Even the question of whether art can be defined has generated controversy, according to Nishimuta and Layng (2021). In a recent article published in the journal *The Psychological Record*, they argue that art and craft are defined by the creative process and not by the audience. Those who see the objects can say whether it is art or not, but what they are determining is a characteristic: the "universality" or "popularity", which is a variable attribute of art. Artists are organisers of stimuli that may or may not be perceived by the audience. The audience may also recognize certain stimuli but not engage with them; that is, they may consider that the stimuli do not have an aesthetic appeal that they like. In more extreme cases, they may think that the art is poorly made or even think that what they have in front of them is not art. What distinguishes art from craft is the interaction with the creation – with craft, there is a consequence, function or utility. The attributes of art will be the (1) creative process of the work, (2) maintained through the arrangement of the work's stimuli, and which becomes craft when it begins to (3) have a utility, which serves a function when complete. The term craft goes further than art because it assumes that the art was not created just to be enjoyed by itself.

Adamson (2007) associates craft with a process that culminates in skill and technique and art as a place where free play of ideas is completely dissociated from processes and materials.

Schön (1994, p. 18) also suggests that art exists in the professional practice of engineers, teachers, scientists, architects, and sometimes managers. Within their daily practice, they reveal some artistry in the way they find problems and how they solve them. The creative process can be considered art, and the way it is developed is related to the craft.

One of the curiosities I had was to understand why I found several programmers who had artistic training and who had embarked on another career in the area of programming. The change was only possible because they understood that there is a relationship between the two areas in the creative process. A painting, for example, starts with a white canvas, and little by little, the artist adds brushstrokes, colours and shapes. The lines of code also go through the same process – it starts as a white canvas and gradually adds to it. Code is also a transformative process in the sense that when you add numbers or data, you get an output that may be different from the input, hence, the transformation. As a programmer, I get positive feedback from my quest for the right code. There is art in the creative process, at least in the eyes of the creator (Sullivan, 2009). I would not want to say that all code is an art or craft, but several features of the code are linked to features of the art. Using Nishimuta and Layng's (2021) definition of the variable attributes of art, that is, those that can change according to different perspectives, we then have an analogy with software development.

Form or Medium

The software is made up of lines of code generated on the computer, and as such, it is ephemeral, usually in bits and bytes, but it is created in different programming languages. There are a wide variety of languages, and the creator has the possibility to choose the one that best suits the task he wants to perform or the specialisation of his craft.

Subject

The software can be created for different purposes, including computer programming, web development, algorithms, data structure, human-computer interaction, artificial intelligence, and although programming is a universal language, it adapts to different purposes.

Complexity of the work

The complexity of the work is one of the characteristics of art, which, from my point of view, allows for assessing the potential of a work of art. This is one of the reasons why we do not have children's art displayed in art galleries worldwide. There is a range of complexity in what we produce as software engineers that allows us to assess the complexity of the project and the ability to assess the effort put into the software so that we can consider it an art or craft.

Time taken to complete / perform

Along with the previous point about complexity and following the principle of the art & craft movement, something created in series on a production line, without a soul, has some difficulty imposing itself as an art object. On the other hand, objects that were thought out, studied with analytical rigour, and took time in the creative and execution process are more easily inserted in the artistic category.

Price

Price is an art variable, and in terms of software development, I can only associate it with the time it takes for the creation to be produced. The longer, the more expensive, and although it has value in artistic terms, it is never just the intrinsic value, as is the case with software.

Audience's perspective

As we will see in the following paragraphs, this variable is subjective in relation to programming, and co-researchers in this research have given their opinions (please see the following paragraphs for an answer).

Permanence

This characteristic is more difficult to manifest in software development since the software is hidden from most of the public. Also, the software is typically evolutionary, meaning it is not one piece that's set in stone forever.

Universality (Popularity)

This variable characteristic of art is perhaps one of the most important in valuing an artistic object. Popularity makes it more accessible to a more significant number of people, which leads to a larger number of people considering a particular object as art. The popularity of computer code is low, and maybe there is not a serious discussion about it, which, here, in this project, I want to demystify.

In the previous paragraphs, I explained the reasons why I can see a relationship between computer code and the arts & crafts movement. The relevance of this discussion within this research focuses on the fact that our view of computer code within the Arts & Crafts movement changes our perception of creativity or lack thereof within what I am looking for. If programming is neither an art nor a craft, then creativity may not make any sense. Next, I will explain the connection that the co-researchers of this project see in software development as an art.

The first feeling related to craft is that programming changes the concept of reality around us. Through programming, software developers are the creators of a new reality. Being able to position specific data, manipulate them within the software, and extract something completely different is one of the stimuli that pleases those who produce lines of code. Furthermore, being the creator of this process is exciting. Some co-researchers consider that software developers are painting with numbers and letters instead of painting with paints and brushes, and in that sense, they should be considered artists. There is a correlation.

> "I found that it's not just about image, imagery, or like some kind of UI [User Interface] for me, it was also numbers like, I really enjoy putting numbers in you getting numbers out. There's something in that that it really excites me [laughs]. You do like calculations you just putting stuff into a function and then when you can have like complex layers, its functions and that and they will [get something out of it]" (Case#14, 2019) answering a question about frustrations (a follow up from Question#7)

Viewing programming as an art depends on the creator's point of view (Nishimuta and Layng, 2021; Sullivan, 2009) but also includes skill and technique (the craft). Unless we claim that programming is an automatic process, which does not lead to a process of creative reflection, then we can say that there is no craft. Nevertheless, setting a thought process plus skills acquired over the years, I say that it is a craft due to the amount of knowledge and years of experience needed to master this craft.

> "There are quite a few people out there who see it as some kind of art form (as if) they are creating a Michelangelo, or you know, painting like Rembrandt when they are writing their programs and it really should be seeing in the same kind of light. This is a derivative of mathematical processing. It is nothing more than that." (Case#6, 2019) answering a question about the code (Question#8)

Programming code is creative. Within what are the rules and the concepts themselves, there is ample space for creativity. However, it is necessary to learn first to create later. We always create in a context – that context is the domain and field in which we are creating. Understanding this context is important because the domain and the field impact how creative we can be. Knowing the rules and content of your domain and knowing the criteria of selection and preferences of the field is essential (Loback, 2021). As a student in my first university degree, I had the opportunity to learn photography — the classic one. Before I could create through the camera lens, I had to learn the technical-technological aspects of the medium I would be using in order to become familiar with the technical aspects of the medium. My photographs are average. I know the rules of photography. I know the difference between the depth of field, the effect of the aperture and how to choose paper sensitivity (or digital sensor in modern times). In technical terms, I can understand the quality of a photograph, but the most important thing, in my opinion, is the aesthetics, the choice of subjects and the framing. These are things that exceed me and do not come naturally. Photography is the primary model of the intersection of art and technology. Moreover, before owning fully automatic cameras, I had to get to know the intricacies of this craft, which has become popular nowadays thanks to its automation and its availability on smartphones. Millions of people take photographs these days; most are neither art nor craft. In the same way that the art & craft movement rebelled against automation and mass production of objects, I believe we must keep our best software products optimised by programmers who know what they are doing and who strive to keep themselves updated in their craft.

7.2. Creativity in the code

The second question I would like to explore is whether there is creativity in the computer code. Nothing better than asking the co-researchers of this project and hearing what they have to say about it. The question was asked directly; in general terms, 81% of the co-researchers gave a positive answer, meaning that they have the notion that they are creating something unique, and they will have to put some creativity into solving that specific problem. Moreover, the solutions are typically unique because they are different from other things that have been done. In terms of programming, you can do something that already exists but in a different context, and therefore there is a need to create or adapt to that different context, and for that, creativity is needed.

> "If you have to come up with a specific solution to a specific problem, it's always creative, so I wouldn't make a distinction, I guess. I thought about that for a while, because people often think, 'Oh, that's so creative, that's not creative.' I think whatever – I mean, creative means making up something that wasn't there before, right? The definition of being creative;" (Case#3, 2019) answering a question about expertise (variation of Question#4)

Programming code is creative because it solves problems. Furthermore, because each programming project tends to be different from the others, the solution to the problem differs from other solutions because it was created specifically for this purpose.

7.2.1. Code process

The evolution of the way programmers work is entirely the responsibility of the best creative thinkers that programming has to offer. Thanks to their creativity, we have new ways of programming, and the profession has advanced towards a beautiful future. There is creativity in the programming process. A lot of thought and dedication is put into the process of creating code so that we have new libraries and new programming paradigms. This process includes the architecture and how the code is structured – there is much room for creativity there.

> “So, everything changed. We got rid of the tables, we started using more semantical markup and everything, and then, with react and all the libraries, VUE, now, nowadays that we have everything is a lot more interactive and user friendly and fast. Hmm, so this is, all of this is available because of creative people. Because of people who like breaking the rules and like doing things differently.” (Case#16, 2020) answering a question about creativity in the code process (a variation of Question#15)

To some, on the surface, the code might look like it is not creative, but underneath there is creativity. In the same way that a painter takes a blank canvas and starts adding colours and shapes, programming starts with no form and then develops as new ideas and thoughts are added in the form of lines of code. The process is creative in that a person has to choose from a limited set of text and keywords and form code that will produce something extraordinary. Initially, the process is straightforward with the printing of text and loops. Nevertheless, as programming becomes more advanced than before, where you know the desired outcome, there is no way to get there unless one uses creativity to move from input to output by filling in all the blanks. There is no way to do this other than through creativity.

> “If I’m building something, for me, that’s, that’s it. I mean, you can’t get more creative than that.” (Case#10, 2020) answering a question about creativity in coding (a variation of Question#15)

7.2.2. Code

Apart from the processes and how we approach tasks, you can also see creativity in the code itself or in the programming language itself. We can give the same task to ten different programmers and then compare the code; a high percentage of the code will be different from the others. There were those who set the bar at 99%, but I dare to say that the code is different because the variable names are different, the order and organisation are different, and lastly, you can use native functions that are different too. There is already differentiation by the simple fact that there is a choice. Furthermore, the greater the challenge and the greater the task, the more differences we will find.

> “Well, I guess if it's a one line request, then probably will be the same but ... yeah, there's definitely room for creativity and just because of the variation. So many ways to do it.” (Case#4, 2019) answering a question about creativity in the code (a variation of Question#15)

I used the code below to exemplify the different ways to produce the same effect in a javascript code snippet. In this case, it prints the name of the four generated colours “Red”, “Green”, “Blue”, and “Grey”.

Table 4: Code Variance

```
var getColorNames = function() {
    var colors = ["Red", "Green", "Blue", "Grey"];
    for (var i = 0; colors[i]; i++) {
        console.log(colors[i]);
    }
};
getColorNames();
```

```
function getColorNames() {
    var colors = ["Red", "Green", "Blue", "Grey"];
    var i = 0;
    while (colors[i]) {
        console.log(colors[i]);
        i++;
    }
}
getColorNames();
```

```
const getColorNames = () => {
    ["Red", "Green", "Blue", "Grey"].map(color => {
        console.log(color);
    });
};
getColorNames();
```

```
(() => {
    ["Red", "Green", "Blue", "Grey"].filter(color => {
        console.log(color);
    });
})();
```

You can also look at everything that we can do with programming and the variety of uses and expressions and see the creativity there. On the one hand, we are restricted in a language with a limited number of commands, but on the other hand, within these restrictions, we can do whatever we want in terms of programming. We have freedom within constraints. And, although the programmer’s work is often devoid of any happiness, sometimes one finds pleasure in crafting software.

> “But obviously I can’t help enjoying writing code every once in a while, er, so I might indulge in making things elegant and succinct and nice, so that can be – that can be fun. (pauses)” (Case#3, 2019) answering a question about the relationship with the code (variation of Question#8)

7.2.3. Programming Language

One consideration regarding creativity in programming is the creation of the language itself. There are several programming languages, some generic and others with specific purposes, and creativity is needed in everything that is created.

> “ I mean, we can talk about creativity in programming for people who actually, you know, develop languages and create, like, libraries or something like that. Then we can talk about creativity.” (Case#13, 2019) answering a question about creativity in the code (variation of Question#15)

The way the language and code are presented constitutes a form of expression for the programmer, and it speaks volumes about what they value and the way they think. The code can be presented in different patterns, classes, separation of concerns, and a range of different concepts that reveal the programmer's creativity.

7.3. Code is not Creative

On the other hand, the co-researchers also felt that the code was not creative. Interestingly, it was not necessarily saying that the code is not creative, but recognizing that they did not have a "feeling" that they were being creative or that it was not creative one hundred per cent of the time. I have already analysed the fact above that it is not because we think there is no creativity that the process itself is not creative. What they mean is that, compared to other media in which the use of creativity is more evident than programming, the creative process seems not to be creative.

> "I built 27,000 lines of code and the component library that I am working on here, this huge data component, hmm, and I've been hard pressed to feel like any part that was particularly creative process. It was not to say (that) it wasn't satisfying. It was great. Solving a problem. I love solving problems and I enjoy doing it, but it doesn't feel creative to me in the way that taking a photograph feels creative or drawing something feels creative or writing something feels creative. It doesn't … it feels more like following a set of processes into it comes to a solution. So, certainly less than 10%, I think, if you try to put a number on it." (Case#6, 2019) answering a question about creativity (variation of Question#15)

The feeling is that programming uses a different part of the brain that is not associated with creativity. Even realising that the co-researchers have a creative mind and, like some, have a background in fine arts, and having used other creative means in the past, they cannot associate the creation of lines of code with the art produced in the past. Furthermore, I assume it is just a feeling. The analogy I can see here is that of a graphite drawer on coloured paper looking at an oil painter on canvas, using a tripod and fancy oils, and thinking that his work is less creative simply because of the medium he is using.

> "A lot of it ... mostly it's about ideation and coming up with cool ideas and trying things, but, actually sitting down in front of a screen and writing stuff ... I've never felt creative when I've been doing it. I never felt like …" (Case#6, 2019) answering a question about creativity (variation of Question#15)

Another exciting analogy of creativity in programming is the analogue and digital signal. The analogue signal is a wave of infinite magnitude drawn beautifully on the screen as opposed to the digital, which are a kind of zeros and ones that go up and down abruptly. In comparison, programming seems a poor relative within the different crafts.

Another reason why they think the code they produce is not creative has to do with what we will discuss in the next section.

7.4. Lego Building

I want to explore a concept that came up during the conversations comparing modern programming to using Lego bricks. Lego is known worldwide as the toy of building blocks, where different pieces are fitted precisely, and other objects are built from there, such as cars, houses, castles, bridges, etc. I tend to appreciate the coherence and consistency in the different pieces that come from the modular nature of the pieces and their infinite number of combinations. A few blog entries scratch the surface when it comes to building blocks and computer programming, but no academic studies relate Lego to building software. Right now, there are thousands of libraries and codes scattered across the internet. The proliferation of the internet has allowed this to happen, and there is almost an online solution for every problem in terms of programming since most things have already been done, especially if we think of the web, where the rate of code reuse is higher than in other areas. Most websites, for example, use the same type of navigation, follow the same patterns, and try to follow more or less the same line in structural terms. There are several sites where programmers look for their building blocks – the functionalities or self-contained objects or functions that will be added to their projects. One of the most famous is stackoverflow.com (Stack Overflow, no date), which works as a "Questions & Answers" space where programmers can ask their colleagues questions about the errors and problems they are facing and get various responses from the community. Similar but complementary, we have github.com (GitHub, no date), which is a code repository where developers can search, use and add to their projects' codebase code collaboratively produced by other people. Other sites/forums include w3schools.com (W3Schools, no date) and codeproject.com (Code Project, no date). In addition to these websites, any programming question can be asked directly on the google.com (Google, no date a) website, and the result will be hundreds of pages with answers to the most varied problems that programmers face in their daily work.

This introduction takes us to the concept of programming done nowadays as if it were a set of legos.

> “Hmm I say, web developer these days is like Lego. You're just sticking together things that other people [have] made.” (Case#7, 2019) answering a question about creativity (variation of Question#15)

In conceptual terms, it works with the union of code from different places created by different people to produce the desired software through these different building blocks. The programmer is literally building on top of other people’s code. The feeling is that this is not programming, much less that there is creativity in this process.

> “I mean, there’s nothing … you're not doing anything. The building blocks are all there you know, and they are quite big building blocks. You’re not putting together things …” (Case#6, 2019) answering a question about experimentation (variation of Question#17)

One of the advantages of using this concept is that we can experiment with different pieces until we find one that fits what we want to produce. Moreover, if you do not find any parts, you always have the more time-consuming option of making your own code. Writing your own code is perhaps too much to ask for some programmers, as the Lego-like approach to programming may have been started due to a lack of in-depth knowledge of the programming language, and looking for other people’s code snippets may be the only solution possible. However, it is a reasonable solution since it can produce more significant learning, and an evolution in knowledge. I can extrapolate and see the creativity in this vision of programming, but it also has its concerns.

One valid view of this lego-like programming is that we are building things rather than actually programming. Furthermore, considering this thought, programming in blocks makes perfect sense. We are putting together a product that is being built piece by piece, brick by brick, lego by lego. These blocks contain all sorts of features that you can plug into your project.

> “... you could argue it in different ways. There are, there are many boxes – many – rather than one large box. And you can be creative with getting all these many boxes and configuring in a way that makes this final product. But forgive my English phrase, is the, at the end of the day, you end up with a box full of lots of many boxes rather than a painting of CSS and unique animation and interactivity.” (Case#9, 2020) answering a question about javascript libraries (variation of Question#22)

From a product perspective, there is no problem using the different parts to produce a completely different final product from each individual part. The combinations are endless, and each piece serves its unique purpose within a completely different whole. As an example, we have a specific piece of programming to format the date in the localization of a particular country – the United Kingdom, where the normalisation uses a "day-month-year" format producing something like "31st October 2022" or, in short, "10/31/2022". There are several other variable parameters that can be added, such as the name of the day of the week ("Monday"), or a short version of the year ("22"), or the week number ("44"), among others. Nevertheless, the analogy with building blocks works because this specific function of formatting and displaying dates can be applied to any software or application that needs to format dates. Not just for the application/software you would be building right now but any software that a programmer is making now or in the future that uses a similar date format. This touches on specific programming concepts like maintainability and reusability but highlights the importance of collaboration between programmers (something I will get into later).

The first question the co-researchers faced was: am I a software developer just because I am assembling the different pieces of programming from different places and people? When we look at the creativity needed to combine the different proposals, we can see that creativity has to be present in the unique combination of the different pieces, even if there is a feeling that we are half-programmers. The programmer's role in this context is to bring all the parts together. This is a valid and acceptable way of programming where the programmer picks and sticks blocks together until it ends with a solution for a problem. Moreover, this can happen in many different ways, and for that, you need to know what you want to get as a final result, as well as previous experiences that proved you can get there. There is also the purist programmer who presumes that everything he uses must be done from scratch without reusing or copying codes from other programmers. I get the impression that this trend has been disappearing since the programming process is increasingly more collaborative, especially after the advent of the internet.

> "Hmm I say, web developers these days are like Lego. You're just sticking together things that other people have made. But I think you can be quite creative in how you stick those things together." (Case#7, 2019) answering a question about creativity (variation of Question#15)

The second question that the co-researchers answered was: is there creativity in assembling programming code like lego pieces? The answer came with some ambiguity in the sense that there is a recognition that the final product is the result of creativity and a creative process but excludes some tasks from the process. The ambiguity comes from a self-reflection and a less orthodox view of creativity in which the programmers think that in the simple selection of different pieces, the choice of order and the manipulation of the pieces, there is no creativity.

> “Like, If you get a box of Legos with a series of instructions in it and you assemble something from the box with the instructions into an assembled piece there are thousands of boxes that are exactly the same piece… same set of pieces with the same set of instructions but everything that comes out is a unique version of that, even though everyone follows the same process to do it. To me programming feels like the assembly of a bunch of pieces of Lego and that doesn’t feel creative, but the end piece feels like the result of a creative process.” (Case#6, 2019) answering a question about code experimentation (variation of Question#17)

Nowadays, programmers’ capabilities are somewhat limited. We must add time constraints to the limitations and gaps. Using pre-defined programming blocks allows you to do things very quickly because the ideal features are already in the building blocks. It is only necessary to connect with the other blocks that already exist in the system being created.

> “I mean, development has become so, you know, what developers do nowadays is they want to deliver a piece of code and if they don’t know how to do it, they Google it. How was it done in the past? It's the fastest way of learning nowadays.” (Case#11, 2020) answering a question about code experimentation (variation of Question#17)

With the programming building blocks, the programmer becomes an architect concerning the creation of the software because he makes the connection between the different elements of the system.

In the paragraphs above, we saw an interesting aspect raised by the co-researchers of this project that references the profession of programmer nowadays as being a copy and paste of functionalities through building blocks – code snippets created by other programmers that can be used and reused by different people. We also saw that there is creativity in this process of choosing, experimenting and fitting together the different pieces.

7.5. Space for creativity

I wrote an introduction to this theme in the Literature Review (section 3.3.1) in the part that refers to “Environment”, which can be summarised in the fact that creativity has to be stimulated and nurtured. There is also a consensus that there is a need to create a flexible work environment with the right equipment so that teams can experiment with different solutions. This solid emphasis on experimental components means that people can work and take risks without fearing the worst consequences (Smith, 2010; Catmull, 2014). Not being afraid to take risks stems from the fact that they have the necessary support, there is an environment of trust, and they feel good and comfortable with the work they are developing. With the facilitation of an environment focused on creativity, experts are able to obtain and optimise their impact within organisations (Pikkarainen, 2011).

> "And, it was, yeah, very, very good company. It was a big company. It wasn't a startup as such. And, yeah, I think creativity comes from the way that the people around you (how to say?), is this feeling and they give you the space to think and they give you the time, so you're not pressured with deadlines." (Case#16, 2020) answering a question about freedom (variation of Question#20)

Programmers notice and appreciate when they are given an opportunity within a space to use their creativity and come up with new ideas. There is a need to research and verify what can or cannot be developed, and as a consequence, having an environment conducive to taking risks is an advantage. You can not expect the environment to be perfect all the time. Typically, limited experimentation exists in which the boundaries of what would be acceptable are not exceeded. What we need is genuine new approaches to programming, and for that, we need a space conducive to this development in the most varied areas.

Looking at creating spaces to foster creativity, at least two concepts emerged in my approach to the collected data: some limitations and a lot of diversity.

> "Well, I think, erm, to be creative there are two things that usually generate or promote creativeness. It's, its diversity and some limitations." (Case#4, 2019) answering a variation of a question about creativity (Question#15)

In the next section, I will explore limitations as a catalyst for creativity and further down I will address the diversity issue when discussing innovation (section 10.4 "Diversity").

7.6. Limitations to Creativity

After seeing the role of creativity in programming, in the production of lines of code and the entire software development process, and the importance of creating an environment conducive to creativity, we arrive at the situation of the limitations to creativity that was mentioned in the conversations for this project.

One of the issues I want to address is finding the ideal balance between innovative ideas and the limitations imposed by organisations on work, code, teams and projects because this can affect how programmers see creativity applied to their lines of code. There is the other side of the coin when it comes to creativity: the boundaries and limitations that can also spark creativity. From the scarcity of possible solutions, an innovative idea can emerge through the absolute need to find a solution. From this point of view, a limited set of rules and constraints can be beneficial for casting creativity. Some of these limitations arise from the programmers' frustrations and the honest conversations we had during our chats. Catmull (2014, p. 66, 167, 189) expresses the idea that some constraints can help foster creativity in certain circumstances. He believes that providing some boundaries helps guide the team to think differently and generate new creative solutions. As a result, constraints help focus energy and effort on a specific goal, allowing some options to be explored simply because there is a particular constraint. Well, not all limitations are beneficial, and it becomes essential to find a balance between providing a specific restrictive framework while leaving room for creative exploration.

I address the topics "diversity" and some "limitations" in particular later on, but I wanted to look at them with a thought on the building blocks I mentioned a little bit above ("Lego Building"). When you have the building blocks in the programming, you can find, out of the box, everything you need to create your software or to produce code. Just search for the blocks/code you need and insert them into the project. From this point of view, there are no limits — there is no limitation. I have already discussed the question of creativity in this context, but basically, you take one block, add another one, and another one, and it is all done by adding different codes, coming from different places, with different characteristics and functions. Now, if the programmer does not find exactly what he needs within this diversity of building blocks and an infinity of code, he starts to encounter limitations and roadblocks. These blockers become a creativity catalyst because the programmer will have to create something new from blocks that had not been combined until then to form new blocks. That is, some limitations are not harmful to creativity.

> "Somebody once said to me that web development was coding in the most hostile possible environment. And I quite like that, I quite like that we were expected to operate under heavy limits and we are expected to optimise and we're expected to make sure that [the] code runs even in extremes. It makes you a better developer, I think, to be able to do that. Hmm, yeah, I like operating under a limit." (Case#6, 2019) answering a follow-up question from Question#29

This is the realisation that some constraints can be healthy. The aim is to encounter the right balance between innovative concepts and the constraints imposed on the work, the team, the project and the code.

In the opposite direction, we also have the constraints that undermine the creative capacity. In the following paragraphs, I will elaborate on some limitations observed during this project's development. There may be many others, but the list I have developed here is by no means exhaustive. These responses from the participants came as a result of the answers given to entirely different questions from each other.

7.6.1. Rigid Structure

Many organisations create their own processes, dictate their best practices, and implement code structures that are rigid in their approach. A rigid structure hinders the entire creative process and prevents the search for innovative solutions that do not fit the standards that the organisation expects.

> I find rigid structure isn't good for that creative flow, so, so, say, I'd have guidelines like, this is what's best, but, you know, feel free to push those boundaries, because that's how we learn, right? I don't believe in strong strictness, rigidity unless, unless there's a reason, unless it's absolutely necessary. (Case#10, 2020) answering a question about rules and constraints (a variation of Question#11)

7.6.2. Mistrust

The best companies hire the best programmers that are available in the job market. They choose the best experts because they want them to bring all their expertise, personality and leadership to the new company, but on the first day of work, they put on the table all the programming rules that they have to comply with, all the best practices, and all processes that must be followed strictly! They do not give the slightest bit of autonomy or opportunity to use their creativity.

> "If they really believe what the team is doing and give them autonomy and the right guidance, it can be incredibly efficient and productive, whereas if there is poor communication, if the leading managers don't give the right autonomy, it can be very degrading and I think at the moment I am finding things quite frustrating in the team that I am in." (Case#2, 2019) answering a variation of a question about frustration (Question#7)

7.6.3. Own Limitations

Personal flaws can also be perceived as limitations to creativity or the use of one's maximum capabilities. There needs to be a self-awareness of your own situation within the organisation.

> "... if someone has no fundamental knowledge of how something works, I mean in, naturally going to be at a disadvantage. ... But as I said, I mean I don't see that I am I don't feel that I am. I know, like, given the opportunity to work on something that I would put my time into it and get the result that I want." (Case#14, 2019) talking about random things (Question#28)

Regarding its own limitations, there is also the cultural aspect where people coming from a more confrontational culture have no problem questioning the more senior ones, while others have a more contrite attitude. They forget that we are usually on the same team trying to solve the same problems in a way that is acceptable to everyone. When you just dutifully accept everything, the end result is always a worse solution. In my professional experience, I have seen this difference in cultures in the different environments in which I have worked.

7.6.4. Code Limitations

The code itself can limit creativity. Javascript, for example, is a programming language that is interpreted – it is not a language that needs to be compiled like Java. But that does not mean it does not take your time since normally, in programming environments, we run automatic programs on top of the code to check for errors in addition to minification[9], optimization, compression and code obfuscation[10].

> "I would say it limits everybody's creativity. And people don't realise how much a compiler slows their production down, as much as having just one screen or tied to the laptop, one figured." (Case#9, 2020) answering a question about experimentation (Question#17)

7.6.5. Other Limitations

In this project, in addition to these limitations mentioned above, there are others that have been mentioned throughout this thesis in their own space, such as time constraints (section "10.3 Time") in which programmers refer to the fact that they do not have enough time to develop their ideas; and the lack of experimentation (section "10.1 Experimentation") where they are not given the opportunity to experience novel solutions.

[9] "Minification, also known as minimization, is the process of removing all unnecessary characters from JavaScript source code without altering its functionality. This includes the removal of whitespace, comments, and semicolons, along with the use of shorter variable names and functions. Minification of JavaScript code results in compact file size." (Cloudfare, no date)

[10] "... obfuscation of code is a technique used to transform plain, easy-to-read code into a new version that is deliberately hard to understand and reverse-engineer. The most common JavaScript obfuscation techniques are reordering, encoding, splitting, renaming, and logic concealing techniques." (Jscrambler, 2022)

When analysing the limitations mentioned in this project, we can clearly see that although there are limitations that are beneficial for solving a specific problem, I also looked at the limitations that are imposed on programmers and that undermine their creative capacity, as is the case of rigid structures, the limitations in terms of code, the co-researchers own limitations and other types of limitations that were mentioned during our conversations.

7.7. Creativity and risk taking

In the following paragraphs, I will discuss the relationship between creativity and risk-taking.

After the first few conversations and, in one of my many interactions with my work colleagues, I created a graph on the back of a piece of paper to try to explain a possible dichotomy between the "art" and "science" of programming (Figure 11: Dichotomy between "art" and "science"). It was a simple exercise to get a set position from work colleagues about how they thought about their work as software developers.

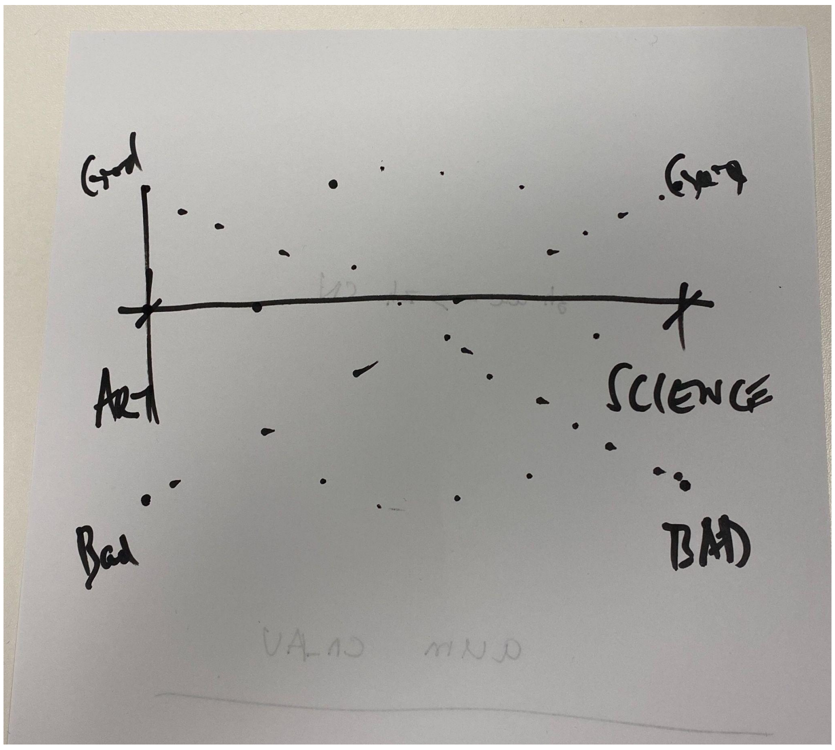


Figure 11: Dichotomy between "art" and "science"

The idea expressed was that people would find a balance between the process that culminates in skill and technique (science) and art, where ideas flow freely without significant concerns about processes or techniques (Adamson, 2007). The by-product would be a greater openness to risk-taking on the "artistic" side instead of an aversion to creativity and risk on the "science" side. Carla Willig (2008, p. 149) argues that this conceptual opposition between "science" and "art" does not reflect a proper division between two mutually exclusive ways of working.

I did not show the graphic to the co-researchers in this project because I did not think it would be relevant at the time, but for some, I gave the definition of this concept at some point while we were talking about creativity in the programming process.

> "I am slap bang in the middle where you should [laughs] I think I'd probably, again, I'm probably more or less sat on that third dimension of, of like the act of writing code can be both. But in the end, the art / science ... sort of paradigm doesn't matter because it's, it's the value that's created in the industry." (Case#12, 2020) answering a question about code frustrations (a variation of Question#7)

The approach to risk attempts to be balanced, and this risk aversion also comes through in the programming. It is understood that some areas have more critical software that requires a much more conservative approach, but there are also some situations where the risk is more acceptable than others.

Hamel (2007, p. 59) argues that "rule-following employees are worth zip in terms of the competitive advantage they generate", alluding to the fact that it is easy to find employees willing to be "biddable, hardworking" but much more challenging to find those with passion and creativity – maybe those who want to break some rules? Hamel himself suggests that "obedience" in the relationship with employers is worthless, and the most valuable human capabilities would be passion (35%), creativity (25%), initiative (20%), intellect (15%) and diligence (5%). In order to obtain a competitive advantage, our search has to be for employees who have passion and are more creative than average (Hamel, 2007, p. 60). Glass (2006, n.p.) explores this dichotomy naming them "discipline" and "flexibility". By placing software development in modern companies' Research and Development area, he argues that flexibility, associated with creativity, has to play a much more important role in modern software development, moving away from "factory-workers" and doing craft and research puzzling over intellectual challenges all day long.

I will explore in more detail the situation of experts as it relates to creativity and code programming, but it is essential to realise that there is room to take some risks and that experts will be called to the forefront to use all their creativity to bring a competitive advantage to organisations.

The situations experienced by the participants in this project are mentioned below.

7.7.1. Risk in the research phase

At the beginning of the project, when you are researching and discovering the limits of what can and cannot be done, it is a time when you can be riskier. The beginning of a project is also when the most creative decisions are made, and most issues are resolved. A programmer takes their chances, and even if they have to give up what was created in that spark of creativity, they always have the lessons learned from the risks they took, and that will allow them to learn for new future situations. If you are in business and have to deliver something, a programmer starts to take fewer risks as you get closer to the delivery deadlines.

> "But I think it's very important to take risks in the beginning stages, because that's where the creative problem solving comes from, as you take a chance and you may have to throw that away, but you learn something and you really do it on again. So yeah, take, take risks." (Case#8, 2020) answering a question about risks (Question#10)

7.7.2. Startup risk

Another time when you can take risks is at the beginning of a startup company. Usually, the teams are smaller, and there is less stress to get things done, with an eye on the completion of the projects and less on the way in which they were developed. This freedom makes you rely more on the creative process in order to gain an advantage over the competition.

7.7.3. Personality

Personality may play a part in the risk-taking dilemma. Some people are more adventurous due to their background or stance in life. A person that is not a "purist" developer may think that code is just a means to something bigger. Coming from a different background in terms of studying programming may affect this perception too.

> "And I'm probably much more willing to break rules than others are, because to me matters more what's on the screen rather than code for the sake of it. [clears throat] Yeah, that has been an issue in the past, it can be — I mean, it depends a lot on the people you work with, it's just...." (Case#3, 2019) talking about rules (version of Question#16)

From the concept of radical engineers mentioned in the knowledge landscape chapter to the fact that computer software is averse to taking risks, all this acts as a limitation to creative development and the realisation of a full innovative potential.

7.8. Summary

In this chapter, I looked at the first theme that emerged from the data analysis: creativity. Initially, I investigated the Arts & Crafts movement in the United Kingdom during the industrial revolution and its connection with creativity to help define what lines of code are in this context. I then explored creativity in the code as seen by co-participants because some could see creativity in their relationship with the code in many aspects of their work. Everyone did not share this perception. I then explored the concept of "Lego building" (joining and collaborating with code from different sources), which emerged in interactions with co-participants and tried to understand the different creative processes. With this, I realised the need to create spaces conducive to creativity. So, I looked at the limitations placed before programmers that limit the use of creativity and innovation, including rigid structures, lack of trust, limitations of programmers themselves and limitations inherent to the code itself, among others. An essential part of the study of creativity has to do with risk-taking, trying to understand what balance can be found between creativity and a controlled ability to tolerate risks. In this field, we have the personality of programmers who may be more prone to taking risks, and I also looked at which phases of the programming project are more acceptable for taking risks. In the next chapter, I will look at the connection between creativity and programming, specifically the lines of code produced.

8. Code / Programming

This dissertation is about creativity and code. It is important to think about creativity within the code because the generalised idea in relation to code and creativity is that with creativity and innovation, we have the ability to generate new ideas, and programming allows us to transform these ideas and bring them to a physical and tangible world that comprises the digital products. Code is the language used to give instructions to computers, but above all, it is written in such a way that it can be understood by other programmers with whom projects are shared. Unless you are working alone on a programming project, ideas and code are shared, and code is always integrated back into a larger repository alongside with code from other developers. This idea that the code is written to be read by people, and not just by the machine, becomes relevant because a more significant effort is needed to communicate by coding that goes beyond technical knowledge to transmit a message capable of convincing the most fervent code readers.

In most of the teams where I worked, one of the key quality control measures is to take the code to a peer review session. Like university dissertations, theses and papers that must be peer-reviewed before publication, everything in programming must be validated by colleagues and ultimately by programmers that are designated as seniors in the project. The pinnacle of communication between programmers happens during “Pull Requests” or “Code Review Sessions”. Pull Requests or Code Reviews happen when the team uses a central code repository, and each developer works with a copy of the central code on their computers. This is the most common setup in programming environments. When the time comes to integrate this local code into the central repository, a virtual session is created where the differences between the local code and the code that is stored centrally are highlighted and open to comments and peer review. Code is only brought into the central repository after the peers and designated approvers have reviewed and approved it. Depending on the amount of code that has been changed and the number of revisions requested by the reviewers, the process can be more or less time-consuming. This process becomes more straightforward if the programmer considers beforehand that it is writing the code to convey its ideas through the code to those who will review it and not just to be interpreted by the machine.

Consequently, creativity plays a key role in the software development process. In traditional terms, I can say that creativity allows the programmer to visualise the final product and understand the user’s needs in order to create a solution that meets that need innovatively and creatively. This is the role everyone thinks of when talking about creativity and programming. Nevertheless, programming code is also about writing readable text that communicates to other human beings in a clean, efficient way that is easy for colleagues to update and maintain. And for that, you need a good dose of creativity.

It is not surprising that good programmers have come from other disparate areas in relation to programming, as mentioned in section 10.4 below in the area of "Diversity". These other areas bring with them a different portion of creativity that, perhaps, traditionally, we do not find in individuals who only take programming courses.

8.1. Type of Relationship with the code

The relationship that programmers have with their code also affects the way they write their lines of code. From the data collected, we have three types of documented cases: the programmers that have an attachment to the code, the programmers that are detached from their code, and those that are both attached and detached simultaneously. This means there is a division in opinion regarding the relationship with the code. Also, the way programmers see their profession and their code affects the way they work and develop their careers. Professionals who view their work as enjoyable or a passion will experience greater job satisfaction than those who see it as a boring routine. Furthermore, that also affects the quality with which they develop their code. This, in turn, can also affect their approach to learning, especially if they keep in mind that they do not want to keep doing the same things for too long. If they do not feel that their code is challenging enough, they may feel that there is no need to evolve in their knowledge. On the other hand, those who only see their lines of code as the predecessor to the paycheck may have a more immediate view of their work, and ultimately, if they do not see their profession as a community of peers, they will not be interested in building relationships or in sharing their discoveries and the way they solve their programming problems. In particular, how you see your code can impact the engagement, commitment and motivation level for the work.

I want to mention that, in general, programmers like programming, and they like writing lines of code. An emotional response is associated with the act of programming that makes an impact and brings inspiration and enthusiasm to the developers. On the other hand, the experience of programming alone already brings satisfaction.

> " I loved programming. I enjoyed it. It really, I was really, really inspired by by it." (Case#10, 2020) answering a question about achievements (variation of Question#26)
> "And I finished school and also I spent quite a lot of my time programming for no reason. I just, I just loved it." (Case#10, 2020) talking about programming (version of Question#1)

This is in line with the thinking of Csikszentmihalyi (1996, p. 5), who, speaking about creativity and creative people, advocates that "creative persons find joy in a job well done".

I will now analyse in detail the types of "relationships" that the co-researchers develop with the creation of their lines of code.

8.1.1. Attached to the code

Even with the code normalizations that can be created nowadays regarding formatting code in teams, there is a feeling of ownership of the code, as if one could analyse the DNA of the code to match it with a person. Nevertheless, for some, the relationship with the code is more like a love-hate relationship but much more balanced. The relationship becomes more substantial the more significant the individual contribution to the code – if a programmer writes an entire component or program alone, the level of attachment is much higher.

> "I say, you know, when I said it's my baby, that's not code that I love. I'm not attached to the code, but it's, I'm a bit more precious about it." (Case#5, 2020) talking about code (version of Question#8)
> "... there's not that much of a relationship other than it just just makes you feel smart that you've done it." (Case#8, 2020) talking about code (version of Question#8)

The attachment can be linked to copyright – the right over who wrote it and the fact of being recognized as the author, but that does not affect the alteration or continuity of the work.

8.1.2. Detached from the code

There are those who think that they have never had a relationship with the code – and never thought one could have a relationship with the code they write – but can notice when the code was written without due care and have no empathy for the code or with the programmer who wrote it. Detachment can also come from the fact that after the code is written, the code becomes open to criticism and suggestions, and the developer wants an emotional distance from it.

> "It's just a mechanism to achieve some result." (Case#4, 2019) talking about code (version of Question#8)

The code does not respond back and has no opinion. The code just does what a developer or programmer tells it to do. As a consequence, there is nothing not to like about the code that is produced. Fancying the code that is produced does not mean that one likes all and any code produced. In the love-hate analogy, the hate part has more to do with the quality of someone else's code when the time arrives to make corrections or changes to it.

> "Otherwise, I don't give a shit. ... Yeah, so if I have to jump in there and fix somebody else's piece of shit, I don't care. I will fix it the best way that I can, but it tends to be as lighter touches I can make, if that makes sense, until it reaches a point where I can't make a light touch – I have to refactor it. Do you see what I'm saying?" (Case#5, 2020) talking about code (version of Question#8)

Moreover, finally, there is an opinion that code is just a means to an end. Giving too much importance to the code and thinking a lot about what to do is unnecessary because it is irrelevant in the end. After all, what really matters, in the end, is the final product.

> "And a lot of nerdy, nerdy, big people, I mean, I say I'm a geek, but to some degree, but people talk about code like it's sexy. It's not sexy, code is code. You know, sexy is obviously what you're attracted to. This is nothing. People use words to make it sound glamorous. It's not glamorous." (Case#9, 2020) free flow talking (version of Question#28)

The perception of the relationship with the created code affects how they use their creativity to develop their code and, ultimately, their entire career in programming.

8.2. Code Community

There is an established community among programmers. In many occasions, it is not a formal community, but they gather together to provide online help for problems posed to programmers (see information, for example, in the section "7.4" on "Lego Building"). The fact that there is a community of help among programmers was something that attracted some of the participants to become programmers. Moreover, that collaborative spirit endures over time and is part of the DNA of most programmers. There is a tendency for programmers to join a support network that becomes a positive factor in their personal and collective development.

> "And so when you are looking at it so much code everywhere, read other people code, ... helped by a whole community of developers it makes a whole lot of difference … Like when you go into a forum and you'd ask questions, and people actually helped a few on this, ... you get to understand fundamentals, sometimes, that you will missing or they'll just write the code for you and tell you

> why it is going to end the way you worked it off. It was an amazing community at the time." (Case#10, 2020) talking about reasons to be a developer (version of Question#3)
>
> "... the big thing always is to collaborate with people … if it's not too many, obviously, if it's a small group of people...that's where the best things come out of, and that's the most fun, usually, to bounce ideas around and then improve on other people's things...What was the exact question again?" (Case#3, 2019) talking about creativity (version of Question#20)

One good thing that developers have in common is a will to share problems and solutions. They know that there are several ways to solve the same problem and people will also find different solutions for common problems. And, this is the great advantage of cooperation, in addition to improving team relationships. Depending on the programming niche and its popularity, there will be a lot of knowledge shared on the internet about that topic, as mentioned in the Lego building part (section 7.4 above). Moreover, it is advantageous to participate in this community since by imitating other programmers, one will increase the arsenal of tools, techniques, and code that one can use in the most varied situations. This learning is important and leads us to the discussion that the code that is written is to be read by humans and not just by the computer.

> "... or ignoring conventions for variable naming, functional remaining, weird spacing. All that kind of stuff is stuff that makes code hard to read and if it is hard to read and people aren't gonna want to format, mess around with it, and if they can't do that then new code is essentially sealed to you and will never be used by anybody else. And this is supposed to be a you know a job where we hand stuff around, we are collaborative, hmm, if you are not collaborating then you probably are not doing it right." (Case#6, 2019) talking about

8.3. Code Satisfaction

I mentioned earlier that programming is a source of joy for programmers. That joy comes from the insights and experiences the co-researchers have had with code in particular, and with creativity in general. There is a sense of admiration and respect for the code and also for the potential they can see once they start producing lines of code, which extends to the satisfaction and pride that comes with creating something new generated from the code.

> "Yeah, I think you need to get down to what creativity means, I suppose, but there is an immense satisfaction that comes from code." (Case#2, 2019) talking about creativity (version of Question#15)

> “When the code does what I want the code to do, that's it [laughs]. Sometimes it's the way you write it. Sometimes it's like, okay, that looks nice. [Okay] So you're proud. It's like your own baby.” (Case#16, 2020) talking about freedom (version of Question#26)

Lines of code are a means of unlocking the creative potential, and different programming languages allow different types of creativity to be used. There is a relationship between creativity and code.

> “And that's what I like about development is that once you know how to code I mean, there are similarities in each different language but each different language has its own tools, its own benefits and appeal.” (Case#14, 2019) talking about creativity and code (version of Question#15)

In general terms, there is a perception that the code is creative and simultaneously a process that brings personal satisfaction. Furthermore, programmers place value on the feeling of ownership and achievement because they manage to create something new and unexpected through the computer code.

8.4. Summary

In this chapter, I highlighted the importance of creativity within software development, more specifically in the code produced, exemplified through peer-reviewed sessions. I looked at the relationship developed between the co-researchers and their code, as this reflects the importance they place on collaborative relationships, their engagement with the community and their work. There is a well-established community of programmers who like to share their problems and solutions, and from this collaboration come the most creative ideas that everyone shares. After analysing creativity in the production of lines of code, I will look at expertise and its connection with the community of practice and the execution of creative ideas.

9. Expertise

In the Literature Review chapter (section 3 above – "Knowledge Landscape"), I analysed various aspects related to expertise, its connection with professional practice and some models developed on the subject, without forgetting the role of intuition in the expert's characterisation. The experiences were built by the co-researchers and led each one to understand better their position concerning their own expertise and the idea they had of themselves concerning their practice.

It is important to look at expertise because experts have a privileged position when it comes to creativity, and I am trying to understand what experts perceive as creativity in programming.

Before deconstructing my idea of expertise, in the following paragraphs, I will check what programmers think of their own expertise, analyse some assumptions and draw some conclusions about their self-analysis. Then, I will identify some expertise-related characteristics and link them to the literature review. In between, I will address intuition and how it is used by experts, ending with the importance of expert programmers becoming managers to shift the perceptions concerning risk-taking, creativity and innovation.

9.1. Programmer beliefs

In my opinion, all the people I had conversations with for this research were experts. This is a statement I made at the beginning of my research and which, in principle, is difficult to substantiate if my actual reasoning is not understood. Initially, I thought expertise was associated with an accumulation of knowledge and cases inside your head that would help the person solve problems with new solutions. This was grounded in the work of Dreyfus (Dreyfus and Dreyfus, 2005) and the various stages of knowledge acquisition. Over time, and through the analysis of the conversations, I realised that only my external opinion was not enough to define the expertise of my co-researchers. It was necessary to hear what each person had to say to me regarding their own concept of expertise since this was intrinsically linked to the way each person saw their own practice (Love and Guthrie, 1999; Dall'Alba, 2006). In defence of my position that all the people I had conversations with pertaining to this project are experts, I have another point mentioned earlier: the social aspect of expertise related to reputation in which a layperson may think that another is an expert based on the very interactions that they had with that person. That is, based on my practice and on the knowledge I have of my profession, I could eventually classify my colleagues as experts based on my experience, my vision of my work, and based on the position in which I currently find myself in relation to my own development in terms of skills and career.

Taking these concepts further, it was imperative to ask the co-researchers if they considered themselves experts and how they came to these conclusions.

> "If I need to do that, like, I have to have a very good foundation, foundational knowledge and then I will always approach with those questions. Like, what are the … what is the best way to approach this? What is ... I mean it, also when you've done this for a while, sometimes things become automatic, aren't they? You just know not to do this because it's gonna lead to a lot of pain, eventually …" (Case#10, 2020) Answering a variant of Question#5

In this research, when I asked the co-researchers if they considered themselves experts (Question#4 and its variants), opinions were evenly divided: half of the co-researchers thought they were not experts while the other half had no problems in assuming their expertise. In the first answer that the co-researchers provided about the question, I did not explain what expertise was or was not. Moreover, the answer provided was about their "understanding of, and in, practice" (Dall'Alba, 2006). This was their definition of expert or expertise in the practice, according to their own perception and embodied knowledge that they had when the question was asked. The answer, in part, depends on your perception of what it is to be a programmer (collectively), the skills needed to be and be in this profession (individually), and the ontological view of the act of programming itself (Costley, 2010).

9.2. The "Non Experts"

According to the description based on the work of Dreyfus mentioned above, all co-researchers chosen for this research are experts, even if they do not consider themselves so. It may be that they do not consider themselves experts because they do not know the definition of expertise that is portrayed in this research, but it could also be due to other reasons.

> "I begin with the assumption that competent practitioners usually know more than they can say. They ex-hibit a kind of knowing-in-practice, most of which is tacit." (Schön, 1994, p. viii)

I agree with Schön's (1994, p. viii) statement regarding some modesty of the co-researchers positioning in relation to their own expertise, and I reaffirm that their expertise is understood or implied without being stated. It would be interesting to understand whether or not there are other factors that could influence the perception of those who do not consider themselves experts.

9.2.1. Constant Change

It is increasingly difficult for a person to say that they are an expert in programming. This is a feeling shared in this research because programming has constantly been changing its paradigms in the last fifteen or twenty years. Technologies have been evolving, and there is a constant need to learn, develop new skills and catch up with something. The constant change has created extra anxiety and pressure on co-researchers because they feel their expertise is constantly slipping away, and they are playing catch-up all the time. There is this feeling that compared to the best practitioners, there is still a long way to go.

> "I think I have expertise. I don't think I am an expert. I think it's, especially in development, it's so hard to call you – yourself that, because the goalposts are moving all the time." (Case#12, 2020) Answering a version of Question#4

When a programmer manages to learn a new language, or a new framework, or a new way of doing things, other new questions appear, and there is a feeling that everything has gone back to square one and they have suddenly ceased to be an expert. This idea arises because their perception of their practice makes them think that only horizontal knowledge is valid as explained in section 3 above ("Knowledge Landscape"). There is a distinction between explicit knowledge (propositional) and tacit knowledge, which is experiential (sometimes cannot be explained) and in which the co-researchers struggle to grasp the difference. Furthermore, what is important is that knowing how to do things is the only way to demonstrate their expertise. They see themselves as task-makers (Savignano, 1989; Savignano, 1996). This task-makers analogy comes from a role-playing game for Macintosh from the late 1980s and early 1990s, where the player travels across the realm performing tasks for his mentor. Along the way, you level up as you perform tasks and face difficulties. Eventually, the player will be proclaimed "Protector of the Land" and hailed by its citizens. In the same way, programmers see themselves as doing a quest to become experts, and this only comes as they encounter difficulties and win to be respected and recognised as experts in the field. The other aspect of task-makers is having a factory worker role, in which tasks are decided at a higher level, and programmers are mere executors of these tasks. In this view, programmers are simply technocrats looking for technical solutions to problems without regard for the bigger picture or human, social and creative aspects – they are simply machines sitting in some dark basement concerned only with the technical aspects of the lines of code that they eventually write. This can become an ethical problem when the code they are writing will be for decision-making applications with real consequences for real people in the physical world. For example, we can have software that decides whether or not a person should receive a mortgage or be recruited to do a particular job. Code has consequences, and programmers must be aware of this.

9.2.2. Knowledge overflow

Programmers have a piece of pretty extensive knowledge. The amount of things they know and what they can create with it is substantial. However, whenever I pay attention to what my peers say at conferences, read a programming blog, buy a new book, or listen to a podcast, I quickly conclude that there is a lot that I still do not know. Moreover, what I still do not know is overwhelming.

> “So, I mean, I, I know enough to do my job and I try to keep up and add to my skills set. But it's just massive, massive amount of knowledge out there. [...] you were used to be able, like, to build a website with one or two guys, maybe, who would be an expert in everything, but now, it's really, really narrow. And there's a lot of specialisation in particular subsets of skills. So, yeah, you can be proficient in one area, but it's really difficult to be, you know, to know everything ... if that's what you call an expert too.” (Case#4, 2019) Answering Question#3

I still have plenty to learn. In a way, this is scary, and this self-realisation that I still have many things to learn leads us, with a certain degree of humility, to decide that I am not yet at the level of expertise I would like to be. The feeling is that I may never reach this level because, in the meantime, knowledge will also expand, and the “goalpost” will move again. This feeling is in consonance with what Ericsson (1993, p. 366) defends, in which to be a world-renowned expert, in addition to mastering what your peers are proficient in, it is necessary to have a deliberate practice in specific areas to advance to the next level (Sawyer, 2013). In the same way that I still have a lot to learn, I also have some things to unlearn, and I must be open for that to happen. Perhaps the way I deal with problems via heuristics needs to be rethought, or I need to recognise that I will not be able to be an expert in different areas of expertise and narrow down my path.

9.2.3. Confidence

Confidence or lack of confidence can affect the way a developer sees themselves within the profession, for the reasons mentioned above or simply because the answer was pragmatic.

> “I think I have expertise in areas and I know how to approach problems and solve problems for the business and for teams, but I wouldn't feel confident saying I am an expert.” (Case#12, 2020) Answering Question#4

For a programmer who is a contractor, confidence in his abilities is much more critical, as one needs to prove its “worth” at the next gig (and constantly). Nevertheless, this confidence can also be undermined by the assigned tasks themselves and the apparent inability to find expert solutions.

> "Like, like today, I could feel like I am the best programmer in the world because everything is just working and then tomorrow, I can have a bug and I feel like: why are you even doing this? You're a fraud [laughs]. So, so, depends on … I feel like I can be relied upon to deliver things. But I think to say that I'm an expert, I mean to say that I'm an expert is a little bit arrogant. Let someone else decide what to expect or not [laughs]." (Case#10, 2020) Answering Question#4

Here, I can see that the concept of expertise is also linked to the perception one has of what others think about the profession. Goldman (2018, p. 8) discusses this social aspect which is also relevant to the co-researchers of this research. While it does not make sense for programming to be judged by a layperson in the field, the result of the programmer's work can be judged by people outside the programming, as it is a means to an end: having a working software.

9.2.4. Practice

The effect of prolonged practice can better explain the ability to develop more accurate intuitions than by the simple fact of using heuristics – experimentation, solving problems on your own, evaluation, or trial and error (Kahneman, 2011). It takes time, and time cannot fast-track. Although I can think that the simple fact of practising for ten years in a specific area to become an expert, I can also take into account the deliberate practice, which in the case of programming probably means covering some deficiencies through the specific study of some area or the exposure to certain characteristics of languages (Ericsson et al., 1993).

> "... but I wouldn't say I am an expert. I think I need a lot more time, I need to see a lot more problems and solve a lot more problems to even be near that." (Case#12, 2020) Answering Question#4

There is a clear recognition that exposure to a certain number of problems and solving them using a variety of techniques over a long period of time leads to professional maturity and experience.

9.2.5. Monotony

While you are learning and the technology is new, it is worth continuing to invest your time and skills, but once you have mastered a particular area, you begin to question whether it is meriting continuing to learn. What is the purpose of the expertise? While you are sure you do not know everything, there is the incentive to keep learning, and it is still fun to do it, but it starts to get tedious as time goes by.

> “… you know what? I, I never become an expert. Well, one of the reasons I say it is because It's something I noticed all through my life is that when I become good at something, I will then try something else.” (Case#8, 2020) Answering a variation of Question#20

In summary, some experts play down their expertise out of humility, distrust concerning their own abilities, or comparison with other developers at a more advanced level of expertise, or for a real lack of knowledge. On the other hand, there is the recognition that after acquiring a certain amount of knowledge, using it becomes boring.

9.3. The “Experts”

The other half of the co-researchers had no problem considering themselves experts by their own standards. Moreover, there is nothing to be done about it since the evaluation that counts is the one that the co-researchers make of themselves. Furthermore, half of the co-researchers thought they were experts with some constraints.

9.3.1. You cannot be an expert in everything

Even pointing towards being experts, there is a recognition that they cannot be experts in everything but still manage to have the ability to have expertise in what they do. Furthermore, what they do, they do it well. Doing certain things well brings confidence to regard themselves as experts.

> “I don’t feel like I am the most gifted computer programmer in the world or even close to being in the top 50% of people in the world that do the job that I do, but I feel like I’m competent and I know enough about my industry or my job now to feel confident with any task that is thrown to me.” (Case#6, 2019) Answering Question#4

It seems that understanding the environment and the practice in which one is inserted brings confidence in their own skills to succeed in the environment. Having a clear perception of what is known or not helps in positioning in relation to the level of expertise that one believes to have.

9.3.2. Experience

The longer they work in the profession, the more confidence they show, making it easier to acknowledge their own abilities. Malcolm Gladwell’s famous idea of the ten thousand hours of practice rule for becoming an expert comes across in the answers about their definition of expertise (Gladwell, 2008, p. 38). To become an expert, it is necessary to have practice correlated with time. I was not the one who brought this author to the answers, but it was an idea conveyed by the co-researchers during their answers. This is the widespread perception of expertise. It is contested as a marketing plot, but Anders Ericsson – a professor of psychology who studies expert performance in different domains, such as music, chess, medicine and sport – advocates that although the acquisition of knowledge and skill is important to achieve expert performance what really makes the difference is the deliberate efforts to improve. It is not enough just to have experience and build knowledge on top of other professionals but to exercise professional practice for at least ten years and use deliberate efforts to improve (Ericsson et al., 1993). This deliberate effort to improve includes highly structured activities that are designed to improve current performance.

> "I feel I've got certain skills, I've reached a certain level … over time, and with enough experience, I suddenly I notice that I do have strong opinions on how to do things" (Case#2, 2019) Answering Question#4

Even though they have the notion that they do not know everything and that, comparatively, recognise that other people know more and are at a higher level of expertise, they understand that time at the highest-level favours experts.

> "Yeah. I mean, I wouldn't say I'm the best at it. I mean, it's probably people out there much better than me, but I've been doing it a long time. So very experienced, definitely an expert." (Case#9, 2020) Answering Question#4

In short, it is recognised that there is a process to reach a level of excellence. It is not a one-off thing or something you can order or buy from a till. There is no shortcut to the time and amount of hours needed to exercise your expertise effectively. The longer you spend at the highest level, the greater the possibility of becoming a true expert. This is valid for an activity that requires a high degree of skill, of which programming is part but excludes intuition. There is an expert judgement on the analytical level, but also on a moral and aesthetic level. The development of these characteristics is more related to different cultures and ways of perceiving knowledge than to a rigid discipline of learning associated more with physical activities. Acquiring the skill is not enough to be considered an expert. Deliberate exposure to various forms of acquiring knowledge is required resulting in different patterns gained through experience that can be used in similar situations in the future.

9.4. Intuition

I could not talk about expertise without mentioning intuition. Initially, I had in mind that experts rely almost entirely on intuition and that intuition was the main external signal of expertise. In the "Knowledge Landscape" in section 3.4.2 above, I presented the three currents of thought in relation to intuition – the Naturalistic Decision Making community, the Fast and Frugal Heuristics researchers and the Heuristics & Bias community (Klein, 2013). It is not hard to see that we find people within the programming community that fit within each of these models.

9.4.1. Conscious Intuition

For some programmers, there is an awareness that intuition is being used, and the basis of the decision is consciously built on past experiences.

> "I think, I don't know whether, you know, I feel that I have particular good intuition towards these kind of stuff, but I do feel that some decision I've made were intuitive decisions, and sometimes I surprise myself how we made a good decision based on nothing other than the gut feeling – but that does happen more often than I would be comfortable with, but it is certainly something that I do." (Case#6, 2019) talking about decision making (a variant of Question#5)

The perception is that one should not act on intuition. Even those who do it recognise some discomfort. The general idea is that all answers should be based on the scientific method. Still, because one can get answers in the "gut feeling", one acts with intuition more often than one thinks would be acceptable in a project. You may not feel comfortable with the decisions but are surprised at the result. Although it does not have a metric associated with good decisions, they stick. Moreover, variables and tools can be added to intuitions to increase the degree of reliability and reinforce their position in the decision-making process. We are not talking about a magic process but a process where decisions are made based on past experiences and looking to the future.

One of the arguments found in my data in favour of using intuition in programming is the fact that there are always new things in programming. Programs are rarely repeated since if that were the case, it would be enough to sell the same thing over and over again. Years of experience make decisions faster. It is the same as learning to discern choices because you make decisions in a way you do not like in some projects. Furthermore, the more you use intuition, the easier it becomes to make the decisions you believe are the right ones. Nevertheless, there is always a risk of the unknown. Furthermore, it is new. You never know if it will work at 100%. Perhaps this risk fuels the desire to create new things and go down paths never before navigated, and intuition becomes part of the adventurous process.

> "Hmm, I think, again, drawing from experience is helpful." (Case#2, 2019), answering a question about solving problems (variation of Question#9)

Experience is something that always comes up when the subject is intuition. I already mentioned in two paragraphs above the fact that over time programmers learn what good practices are and to differentiate for themselves what may and may not work in certain situations. This set of experiences is what makes programmers learn new things in each situation that they face in their day-to-day work. Moreover, this whole set of new experiences leads to a refinement of decisions that informs the way programmers produce code and how the decisions are made when they are programming.

> "Yeah, seeing all the puzzle having seen as many puzzle pieces before, as possible, allows you to put them in different orders ... I guess …" (Case#12, 2020), answering a question about the way it makes decisions (variation of Question#5)

The puzzle analogy is not the best, but if you know as many types of puzzle pieces (patterns) as possible, then it will be possible to arrange the puzzle in the best way to obtain the expected result. In a way, Klein's thinking, in a naturalistic proposal, appreciates the fact that the greater the number of patterns achieved through experience, the greater the knowledge and the possibility of creating new knowledge (Klein, 2013).

9.4.2. Unconscious Intuition

Regarding intuition, the processing of the ideas of the people I talked to was not always very clear. On the one hand, they had no theoretical knowledge of intuition, and on the other hand, it is not easy to articulate something that is not understood perfectly. Gigerenzer (2007, digital page not available) articulates that in the decision-making process, unconscious intuition can be a valuable tool, especially when there is no reliable and complete information or when there is a time constraint. Although he recognizes that intuition does not always work, it should be used based on experience and training so that it can be more effective and faster.

> "I found, I think with programming, I found that I can get into a state where the problem is so complex and I've invested so much time in it, I can so, I can almost type, I can code without having to think about what I'm typing and I can almost distance myself from it and I can just be, in some kind of meta-level, just watching myself solving. I almost don't understand what I'm doing and yet, when I run it, it's kind of solved the problem. I found that quite an interesting thing. I think, maybe, somehow when you have a problem so … when you spend so much time doing something that

certain patterns that … are reproducible, that you don't need to be fully conscious of. I found that, I found interesting (laughs).” (Case#2, 2019) answering a question about other themes that were not covered in the conversation (Question#29)

My perception is that there is a state of flow, almost a trance, when the programmer is completely immersed in the programming task. This comes from years of experience discovering patterns, and the programming functions almost automatically, causing the code to flow in a transcendental way. What is automatic are particular analysis and decision tools in the brain. What catches my attention in this text excerpt is the fact that it appears near the end of the conversation in response to a variation of Question#29: “What are the things that we did not cover and you want to tell me?”. In other words, it was not something I would have expected to hear. Still, it indicates that this concept is of vital importance for the programmer in question, as it did not want to let the conversation end without making it clear that this is an intriguing subject. For me, it is important because it requires an analysis of the context where the text is inserted. This is an interesting definition of intuition as far as expertise is concerned. The analogy of the soul leaving the body and observing from the outside, the body working alone and solving the problem, is mystical, but in a way, it defines intuition. It is a search for patterns and automatisms and the application of the same to new situations and problems.

I can see here a direct relationship with my initial academic experience when I studied Art & Design. When I was in the creative process of drawing or painting, I was not looking for logical or expected moves. The goal was to be spontaneous, to get into a flow and to have the freedom to create something original and to physically represent what was created inside my head as a new vision – the presentation of a new simulated reality created within another world.

9.5. Attributes

There are some lessons I can take from our experts. Some are practical suggestions that are part of common sense, and others are examples taken from a lifetime of experience exercising the professional activity of programmers in the category of experts. Other ideas reinforce what characterises an expert in relation to his activity as a programmer. These are pieces of advice coming from experts.

9.5.1. A voice

Since the beginning of this project, I have been concerned with understanding the limitations that were placed on expert programmers and understanding the different forms of negotiation between the various agents within the programming branch. The question “What is the competitive advantage that a better programmer can bring to an organisation when all projects already have experts?" resonated in my head from the beginning (Loback, 2018). The reason companies hire excellent and qualified professionals is so that they tackle challenges in unconventional ways, offering solutions that transcend conventional thinking and unlocking the potential of code and coders. Good professionals are hired to have a voice – to bring their knowledge and expertise to the table. Well, in theory, this is the idea. However, after hiring an expert, will the companies that hire new developers be willing to listen?

> "Sometimes, people don't listen (laughs). I think, sometimes people don't listen … I don't like to chase people too much, so, it is like, I just thought that there should be a mutual respect – sometimes I am saying like, you know, the way you've coded this, you know, you need to code it this way, or you need to look at, you know, you need to fix your LINT[11] errors, and sometimes people don't listen. You have to keep – I don't want to keep chasing someone, I am thinking 'ok, even they don't know how to fix it – or they don't want to fix it', kind of thing. Because, I feel like, because I've worked with different levels of developers, kind of thing, I felt like, it is just like – primary example …" (Case#1, 2019) Answer to a variation of the Question#19

Some of my co-researchers feel that their voice is not heard the way they presume it should be. The projects are different and fall into two categories as far as developers' opinions are concerned. Either the opinion of the programmers is taken into account, or it is not. On the one hand, we have a waterfall model – a sequential development process that flows like a waterfall through all the phases of the project, in which the programming work is literally defined in the specifications, and all that is asked of the programmer is to create the lines of code to produce what is already outlined. There is no feedback or interaction with the requirements. The expert's ideas do not count in this situation. This is treating programmers like factory workers in a warehouse pipeline. In my professional experience, I have heard this expression several times, and it limits my intellectual and professional abilities. Some programmers were trained in this way, to think only in technical terms and to produce their code, and they got used to having Project Managers or Business Analysts who "supposedly" translate the business language for the programmers (and vice versa). Not everyone thinks that way, and for some, that mindset from the 80s and 90s is obsolete these days.

> "But I still prefer the job, obviously, where I've got a bit more saying in the way that things are defined. I am happy to sitting there and code things for people. I do the job, but it is nicer if you [are] defining what it is that you are building as well." (Case#6, 2019) Answer to a variation of Question#21

[11] LINT is a software that examines source code from programs, detecting a number of bugs and obscurities, based on the computer language "C" (Johnson, 1978)

The problem with factory workers is that they are only waiting to receive their salary at the end of the day. There is nothing wrong with doing your job well, within the expected parameters, and getting paid for it. Programmers are intelligent people. They are capable of much more than that. They have an understanding of how things are built, and that would surprise stakeholders if they allowed programmers to be part of non-technical conversations concerning corporate businesses. There is added value to the business when developers also focus on the expected end results, on interactions that result in changes for users, or changes in metrics, or changes in the way business is done.

Programmers are also a part of the lack of communication or effective communication. Moreover, it was important for some participants to realise very early in their careers the importance of dealing with people. And, to deal with people, it is necessary to communicate and be trusted – to be reliable.

> “And it's about how you present yourself and how you talk to people. So his words really resonated with me. And about, you know, dealing with people is the most important thing that you do. So that's what I think it does give you a voice. I think it gives you even if you're less technical than Somebody else around you. If you get on with your peers and superiors, then you can still have the loudest voice. Does that make sense?” (Case#5, 2020) talking about competitive advantage (version of Question#24)

Of course, there are consequences for organisations and teams when the voice of developers is not heard. This phrase is not a threat to managers or other stakeholders in any way. Professionals do their job to the best of their abilities, but when their voice is not heard, they do it at a high cost to themselves.

> “I think the last time I really felt lost ... was ... [company name removed] ... with that Angular project! It was so overwhelming because the decision to use Angular was not mine ... and ... because it was so cutting edge, I always having to immerse myself in the, in the ... in the folklore around the ... around the language at the time. And, that's not a nice feeling. You feel like you're sinking constantly.” (Case#5, 2020) Answer to a variation of Question#21

In this day and age, talking about mental health in the workplace is already more normalised, and it is essential to recognize that the well-being of professionals should also be part of the culture of organisations. It is evident that this is valid for other areas, not exclusive to programming. There is pressure not only to make yourself heard but also to convince others of your ideas. Furthermore, not all the time one can present one’s point of view convincingly. When talking about innovation, Wallace and Gruber (1989) explained that the biggest challenge of innovation is convincing others that your idea is worthwhile or changing your idea entirely so that it becomes acceptable to others. It is necessary to balance the points of view not to jeopardise the argument and ultimately close the door to dialogue.

> "And then I spoke up about them, but the managers, I don't know, it fall on deaf ears. They didn't want it, and if they … they, they involve me because I was the most vocal really. Hmm, but after a point I think they, they tried not to involve (me) in some of those discussions because I would have criticised them (laughs)." (Case#2, 2019) Answer to a version of Question#25

As mentioned in Chapter 5, in the area about demographics (section 5.2.1 "Demographics"), a quarter of the co-researchers were female developers. This number does not reflect the reality of the developing teams since the number of women programmers is much lower. One of the co-researchers mentioned the fact that, in some companies, we are faced with a male culture of boys' clubs, which prevents women from expressing themselves in the best way. It is something we have to take into account as people who want to see a change in the workplace, allowing the voice of our fellow programmers to be heard. Moreover, we have an obligation to make that happen. This kind of toxic culture is amply documented, and one of the ways it reveals itself is by diminishing the voice of women or polluting their ideas so that they disappear. This is not what we want at all.

> "Okay, so it would be then, the role before that, so remember, we ... I was talking to the guys ... and they're like, no, we're not gonna change the way we're doing it. Yeah. So in that team, it was quite difficult for me to get my point across. On an emotional side, I would want to say that there was a crick on the boys club, and whatever that was established before I joined the team, and it was an uphill battle for me to be trusted and heard, because there was an assumption that I was a lesser developer. This is the emotional aspect of it. But if I'm, if I think about it a bit more, but shortly, I would say that it's, It's just the way that the culture of the team was. The lead technical was quite rigid in the way that he did things. And he wanted things done the way he wanted." (Case#10, 2020) Answer to a variation of Question#21

Communicating is a challenge and communicating effectively is important for the engagement of programmers and the general perception of the various stakeholders. This effectiveness will only happen when there is mutual respect – the ability to listen to the opinion of someone who was hired to exercise their expertise. When it fails to consider the other person's level of expertise and the other person's range of experience, it is no longer possible to benefit from this same knowledge. Rigid hierarchies and the inability to generate and accept feedback lead to inefficient communication channels. This happens when communication only works in one direction.

There is a voice, and it is a factor that was mentioned as inhibiting the exercise of expertise: "... sometimes not being heard or listened to …" (Case#8, 2020). Having the notion that one has the experience, knowledge or wisdom, making a critical judgement about a situation and not being heard is a risk one should consider in business. It may come from the programmer not being heard, but it may also come from not having a sense of gravitas in the business world where things connect.

9.5.2. Leadership

It may seem obvious, but experts should voice their opinion in a way that moves things forward. They take the initiative. They lead. When a problem or opportunity arises, they are able to look at it from a different perspective and find a way to transform the circumstance. By their initiative, they take responsibility and risk without fear of failure. Where others fail, experts shine.

> "So, I immediately, because of my experience I could see very concrete things that were going wrong and problems that would arise from the way that they were working on." (Case#2, 2019) Answering Question#25

Experts have the ability to see the future based on the data from the present. Some may call it magic; I would say, instead, that it is intuition, and it is based on past experiences and careful evaluation of variables and possible outcomes. Over time, your voice begins to be respected. Possibly the intuitions worked, the results appeared, and the confidence levels were high.

> "By ... where I work? ... Ha, Okay. All right ... [long pause] Yeah, I think I've reached a level where, if I say something - I sound like such a dick. I don't meant this, right? I don't mean to sound like a dick. But you reach a ... reach a point where, you know, people begin to respect your opinion." (Case#5, 2020) Answering Question#16

Opinions are also respected because they have been put up for consideration. There was the knowledge, the evaluation of it, and the courage to voice an opinion. It is not just about knowing that you know something but also how to express and be open to having your opinions judged by others and eventually followed. Polanyi (2009, p. 4) wrote a book based on the premise that "we can know more than we can tell" and introduced the idea of tacit knowledge – a knowledge that is understood or implied without being stated. There are things that I know that I know and that help me make decisions but that I usually cannot explain in a rational way why I choose one path over another. I introduced the concept of intuition discussed in the literature review, and that follows the thinking of Dreyfus (2005) when he describes higher levels of expertise. Other voices explain intuition, including Gary Klein (2015, p. 164), who highlights the various decision-making study communities explained in the literature review.

Ultimately, you officially take the lead on a project. That is not to say that you did not have ownership before, but it simply means that someone recognized the work you had done and made official what was already happening in practice. Again, I repeat that it is a process, and you cannot rush the time necessary to demonstrate the expert's capabilities.

> "So, [company name removed] was great because, when I moved on to the monetate team — this part of the A/B testing — I was the only developer. I was talking to their developers and their development, development team and there was a mutual respect that went on but anything that [company name removed] needed, I did it. It went through me. So that was nice." (Case#5, 2020) Answering Question#20

The ownership of a task or project brings a degree of satisfaction for the expert programmer that should not be overlooked. It can be a connection with the business world, where what they produce as lines of code (text) will become an application with multiple uses that will make all the difference in the real world. It is a way of not living in an abstract world and far from reality. Not only does it enjoy taking the initiative, but it also has the ability to be reliable and accountable. I will address this at another location, but it is worth noting here.

> "You're not there to be the best. Just because you're managing and leading, you're there to sort of promote a way of everybody levelling up and getting better, including yourself doing better in the ... in leadership." (Case#12, 2020) Answering Question#18

Taking the lead and initiative in a project should not be a form of self-promotion but a way to help the team as a whole to perform better. Several studies on leadership point to the "servant leader", and our experts also believe in their mission and their ability to become better people and programmers by promoting their team members and giving them all necessary conditions to become more productive and the best versions of themselves. Robert Greenleaf (2002, p. 19) , the founder of the servant leadership movement, defines it as a philosophy that puts the needs of others first and emphasises the importance of serving others to achieve greatness. In his book "The Servant Leader", James Autry (2004, p. 44) highlights the importance of putting employees' needs first and how this way of looking at leadership can lead to improvements in organisational performance.

Taking the lead also means taking more risks. Taking risks means choosing the most complicated tasks and making the most challenging decisions.

> “I think it is also – I think an expert will take more risks.” (Case#1, 2019) Answering a variant of Question#4

Usually, this involves sticking with tasks that no other programmer has wanted to do. It can be the one that takes the longest time or the one that is the most complicated beforehand. The expert knows that, even if another programmer chooses this task, the task will eventually do a full circle and end up with the expert. An expert is confident they have the answers to the most challenging questions others are asking:

> “What do they think if I can’t do it”; “What happens if I can’t deliver?” (Case#1, 2019) Answering a variant of Question#4

An expert is willing to take that risk. Moreover, that is the significant advantage of being an expert. Nevertheless, being an expert who takes the leadership does not mean taking the opportunity away from someone else. If, on the one hand, you do not turn your back on a challenge, on the other hand, there should be no barrier to what people can or cannot do in a team.

> “I think everyone should be able to wear or get involved in any role and there shouldn't really be any hard constraints around what people do.” (Case#12, 2020) Answering Question#13

Assuming that everyone wants to self-develop, grow, and become experts, the advice is that, if you are still growing in your path, take ownership of tasks you do not yet have the whole experience. Learn along the way. Look for the experts, and have a word with them. Learn and do it.

Furthermore, lastly, if you are a manager who has brought in an expert to solve a problem, be prepared for the expert to take the lead and cause disruption. If someone is an expert in their field, and you have the honour of having that person with their experience in your team, avoid creating rules that limit their performance. There is no point in bringing the expert and not letting them take ownership. No matter how ingrained the team is in the project, accept that the changes will benefit different areas, especially the area of the person who was hired as an expert.

> “... what happens is the big organisations sometimes they spend more money, they bring in all these guys that have very .. that are experts in their fields and sometimes what they ... what they say is:

> we know that we need to change stuff, and we know that we need to adapt to some changes. And sometimes they don't allow you to change [those] rules ...” (Case#11, 2020) Answering Question#11

Taking ownership is a characteristic of experts, but it should also be a goal for anyone who wants to become one. It involves taking responsibility for your choices but also looking for solutions amongst others. It involves getting ahead, taking risks, and allowing others to have their say and grow. It is pulling the string for change, but it is also willing to change to accommodate others.

9.5.3. Knowledge of Different Programming Languages

In order to be considered a programming expert, you must have exposure to different programming languages. You may not be an expert in every language, but the knowledge opens up new horizons. Firstly, it feeds curiosity, and each one will bring some versatility and help with future intuition.

> “I think exposure to different programming languages is very interesting as well. I think I’ve, I am interested in more weirder programming languages and, I think, my experience having played with these different languages has informed how I write my code a bit.” (Case#2, 2019) Answering Question#5

In the same way that the idioms I speak influence how I think and talk, my creative process adapts to different languages. I can communicate well in at least three different languages: Portuguese, English and Spanish. In my introductory text, I speak about my different personalities in each language and the behaviours associated with different languages. It seems like I am three different individuals adapting to different lexicons and forms of communication. The same happens with programming languages. By knowing various programming languages, I am increasing the potential for future use of intuition. I am storing “cases” that may come in handy in the near future.

> “And that's what I like about development is that once you know how to code I mean, there are similarities in each different language but each different language has its own tools, its own benefits and appeal.” (Case#14, 2019) talking about creativity (version of Question#15)

Knowing different programming languages could open doors in the future. What has been learned does not need to be forgotten – perhaps it will be incorporated into a new library or language in the future. Opening the horizons for new ways to program, since each language has its own intricacies, can help an expert solve different future problems.

Furthermore, I am convinced: I do not know what the future of programming will be. I do not know the languages that will be used or the frameworks or libraries that will replace the current ones. I do not even know how many of us will be needed to program in the future, as Artificial Intelligence will replace some or all programmers.

> “Oh, God …. (...) being a compiler developer is the hardest thing of all, I think. I really don’t want to do that kind of engineering that just requires a lot of knowledge that I can’t possibly ever have, but yeah, sometimes I wish that you could take the good bits from every language and coalesce them into one that did everything that you wanted.” (Case#6, 2019) Answering a variant of Question#22

9.5.4. Collaboration

Specialisation is important, and the job market is looking for experts. However, at the same time, it is essential to take a step back and look at what we do as a system linked to other interdisciplinary systems and have the ability to make the necessary connections. In the same way that there are several disciplines within medicine, there is also General Practice, which makes the connection between the different specialities. In terms of programming, it is crucial to know the programming language you are working with and know how everything connects and how everything is interconnected.

> “I think some of the cleverest things I've done as a coder and things where I have come up with a solution and where I'm applying something from a completely different field of programming (going) that solve this problem here. And, so yeah, I think it's my knowledge of whole systems and my pragmatism I guess …” (Case#7, 2019) Answering Question#24

One could argue that the best ideas come from collaborating with others. There are countless cases, which are already part of innovation folklore, of cross-pollination of ideas that became successful. In conversations with other programmers, I found solutions to ongoing problems in areas I was working on completely different from the subjects in question. It is no surprise, then, that collaboration has enormous power. Programmers are not indifferent to this phenomenon. Contrary to the popular idea of programmers being “lab rats”, there is an interest in collaborating with art directors, creatives, and also with other programmers experts within their own field or outside. The interest is not only in getting ideas for oneself but in helping to develop other people’s thinking – bounce ideas around and improve how people think. The development of the World Wide Web (the Internet, as we know it today) was the result of cross-pollination between different areas of knowledge involving different people, as recognized by its author

Tim Berners-Lee (2000, p. 2). Berners-Lee was working at CERN, a physics research laboratory in Switzerland, when he developed the idea of the World Wide Web, which at the time, would enable scientists to share information and collaborate more easily:

> “The Web resulted from many influences on my mind, half-formed thoughts, disparate conversations, and seemingly disconnected experiments. I pieced it together as I pursued my regular work and personal life. I articulated the vision, wrote the first Web programs, and came up with the now pervasive acronyms URL (then UDI), HTTP, HTML, and, of course, World Wide Web. But many other people, most of them unknown, contributed essential ingredients, in much the same almost random fashion.” — Tim Berners-Lee (2000, p. 2)

The fact that at some point, I started to wear different hats and have different experiences even more helps to continue having these experiences. That is why it is essential to get started. In this search for new experiences, it is worth mentioning that programming, for some, was also starting to use different hats, coming from other areas. Interestingly, 40% of the co-researchers did not choose computer science as their primary degree or education (see section 5.2.1.4 above to compare statistics). This means that programming appeared in their lives when they opened their minds to a new opportunity and seized it with force. After a particular moment, they started to like it and thought about developing a career in this new area. And it may also be that they did not choose computer science as the first area of university study because their way of learning is different from the conventional, and perhaps it would reduce interest in that subject and would be a motivation killer in terms of pursuing a career in computing.

> “Things became less interesting to me as soon as someone was teaching them to me at school. Erm, so...if I’d done computer science at university, I think I probably would’ve been less interested about doing things in my spare time related to it, I don’t know, but I would’ve found that annoying.” (Case#7, 2019) talking about skills acquisition (version of Question#3)

I want to open a parenthesis here to clarify that the use of the expression “different hats” came from the analysis of the data and the responses that were given by the co-researchers and should not be confused with the concept of parallel thinking developed by De Bono (2000, p. 4, 10) in his “Thinking Hats” book. According to this method, the participants of a group analyse all sides of a problem in parallel, using the six thinking hats as a metaphor for the different “positions” or “functions” for analysing a problem: facts (white – neutral objective), emotion (red – emotional view), benefit (yellow – logical positive), ideas (green – creativity), planning (blue – process control) and judgement (black – the logical negative) (De Bono, 2000).

What the co-researchers referred to has more to do with trying out different roles in their work and being open to new opportunities related to what they are developing at a given moment. It seems clear to me that there is an enrichment of your position as a programmer, the fact that you are open to having different competing assumptions in relation to your work with the aim of broadening your horizons to different areas. Our worldview determines the path we are going to follow in our career, and having a more comprehensive worldview means that the programmer can see himself doing other things in his career and be available to experiment and be more creative.

Another aspect that comes from the fact that programmers have varied experiences is all their understanding of the digital world. Programming is creating things in a digital world. Moreover, programmers know this. They are aware not only because they are the ones who create them but also because, for the most part, they are technology users and use digital products. They have an inherent knowledge of how things work and bring all that expertise to the world of work and the projects they are involved in. Hence the importance of using different hats in technological terms and constantly searching for varied experiences.

> “I can make things look beautiful. I can dynamically ... I can make ... write them from scratch and I can use libraries. And I've got a whole range of experience with all these different technology skills. I can touch type - that helps. I've got an appreciation, a good experience at management and I'm happy to work in a team - happy to work on my own. I can ... I can do all those things. ” (Case#9, 2020) answering a variation of Question#24

Versatility helps solve problems. Although having the notion that it is impossible to be an expert in numerous things, as we saw earlier, having a good knowledge of several areas and professional experience in some, makes a professional more complete. It adds value to the career and importance to the expertise that I believe they already have, as more insights have been established that will turn into patterns, which, in turn, will be stored as cases for later use in the professional area.

If programmers want to create a greater wealth of insights (Klein, 2013), they must be open to new experiences, not just following the paths already traced and known but genuinely wanting to try new hats.

9.6. From Experts to Managers

Most people want to advance in their careers and get to the top. Anyone with any ambition thinks about their future and what they want to do in a few years. A career as a programmer opens up countless possibilities. Most programmers like what they do and want to continue doing it for as many years as possible. In more than twenty years of working as a programmer, I had the opportunity to practise my profession in the most varied companies and different sectors. Over time I adapted a career plan model based on SFIA – Skills Framework for the Information Age (explained on the "Co-researchers" section 4.4.2). The SFIA is a model based on the process of knowledge accumulation, established on a conventional understanding of professional skills as a set of assignments, such as practical knowledge, skills and attitudes (Dall'Alba and Sandberg, 2006; Dreyfus and Dreyfus, 2005), labelled Stage model, where the character advances from one level to the next.

They all start with the title of Software Developer/Programmer. It is the beginning of the programming career. It can start with an internship or adding junior to the title, but everyone has to commence somewhere. What every developer wants and loves to do is get lost for hours on end, solving problems and creating new programmatic solutions. A person can climb the stairs of the levels and reach the top in a few years, and then what? Here is where some are faced with a crossroads where it is difficult to choose the best path.

It is essential to understand what the co-researchers think about becoming a manager because this is an important career step where they can make a difference to the developers that they lead.

To stay in the same area, there are two different paths. On the one hand, we have management, which will typically be focused on technology, programming and mentoring, which may or may not involve managing people, or, on the other hand, continuing only to develop software without adding more responsibilities. It is evident that to choose one of these areas, several factors come into account, with ambition and temperament playing a fundamental role.

9.6.1. Hindrance

The transition to management involved some hardships for some of the co-researchers. The most significant deficit came from managing people or being dragged into an unwanted role. A developer's mind works in a way that it is constantly solving problems as they are used to a linear world of cause and effect. A programmer produces an action and manages to obtain a certain result, which is predictable at the outset. This predictability did not happen when the subject became people manager. Moreover, starting to manage people brought an increase in problems without an apparent solution, which worried the co-researchers. The difficulty arises from the fact that it is impossible to apply linear causes when it comes to human behaviour. It is impossible to predict accurately what human behaviour will be like, and it is not because we act in a certain way that we will obtain a specific result, as far as people are concerned. People do not have a certain fixed number of solutions that can be applied to their behaviour. Usually, human behaviours are dictated by their hopes, fears, desires, longing and belonging. Humanity is not a machine that can be programmed to obtain the results expected by the programmer. Social relationships and interactions are very important; nothing is simple regarding human systems. This complexity is what was noticed in the co-researchers conversations. Furthermore, we will notice in the following paragraphs how accentuated this inability to deal with people is. There was difficulty in adapting to the new role and especially in learning to delegate responsibilities.

> "I was very bad when I started managing people, when I first starting doing it. It was 10 years ago when I became a Lead Developer for the first time. Hmm, and I have a trouble letting go, hmm, I didn't like ... I felt like people would taking too long to do something that I could do much quicker. I wasn't very good at delegating." (Case#6, 2019) answering a question about management (variation of Question#18)

Looking at the data, I get the impression that the co-researchers, deep down, have the desire to control and firmly believe that people can be controlled through remote control. This leads to a host of frustrations, including impatience and difficulty delegating responsibilities.

This probably happens in other areas as well, and the difficulty comes from the need to manage people and the lack of emotional intelligence and patience. Nevertheless, over time, the skills have improved and enhanced.

> "I wasn't that great at managing people at the beginning because I was impatient. I wanted results. (...) I've got better at it right now." (Case#9, 2020) answering a question about management (variation of Question#28)

There is a tendency to compare the code produced by subordinates with the time it would take the manager to produce the same lines of code. In the beginning, it seems to be the focus of some frustration that, with time, disappeared as the understanding was assimilated that managing is different from programming. One is tempted to want to change code written by other people because you are used to writing your own code. Moreover, letting go of a code different from yours and your way of thinking and doing was a learning process. The evolution went through realising that there are different learning curves and that the people being managed will take their own time to develop their skills. Furthermore, there are different ways of thinking about problems and solutions. The recognition that the support, the identification of their skills and their proper use were more important than the time it takes to perform the tasks.

When you start to manage a team, in most cases, you do not have time to focus on programming anymore. From the co-researchers' experience, it is almost impossible to have a leadership position and, at the same time, get lost in lines of code and produce some good development as it used to be. Moreover, it is a choice between the two areas. They soon realised that they could no longer have both. As much as they like to code, some feel they have to put programming aside to progress in their careers. Furthermore, they miss it because they are stepping away from development.

> "So I miss putting my head down and doing a task but at the same time if you, if you're leading a number of developers, you can't do the same thing. So I miss that. Well, I also, if you're managing something bigger than that task, multiple tasks, then you can't ... you have to, you have to, you know, you have to just step away from development and just look at the bigger picture." (Case#11, 2020) answering a variation of Question#6 about code liberation

The most significant advantage of having a former programmer as a tech lead or manager is that the person has been a programmer and understands everything the developers go through. There is this facility to understand what developers are feeling or going through at a given moment to solve a problem and help them deliver what they need to deliver.

9.6.2. General Improvement

In my sample, 43.8% of the co-researchers had experience as managers (see section 5.2.1 above about "Demographics"). This means they already had experience leading people, projects and codes. When asked if the way they produced code has changed since becoming managers, the unanimous answer was that they had become better developers since they changed their position. The transition from expert to manager transformed the way co-researchers began to code. Moreover, the change was for the better.

> "But I started looking at the code differently, more so. And sort of getting the sense that I got out of it, trying to make sure other people have that. So the sense of pride and ownership in the code that I once had. I wanted the other people to do that, even if they even if they've inherited some code." (Case#8, 2020) answering a question about changes in code (Question#18)

Becoming a manager involved handing over the code to other people as if it were a valuable legacy that, from now on, they would have to nurture and change. From here on, they will have to understand it and do what they want with the code as if they were adoptive parents. This also demonstrates, in a way, the kind of relationship that developers have with their code and the way they understand that code should be treated when passed on to others (section 8.1 "Type of Relationship with the code").

> "But it so, to sort of answer to your question is: Yes, it changes the way that I write code, … So it made me a better developer. And that's how I write code now." (Case#2019) answering a question about changes in code (Question#18)

From the moment they started to use a wider lens, when they became managers, they were able to look at the code differently. In a way, they mirror their own mistakes they used to make and, in doing so, help the new code holders. They began to put aside personal preferences and individual style, in favour of a more elaborate approach, beneficial to the group and project, without an intrinsic personal agenda.

The only manager who thought that there was no change in relation to the code since he started managing people and teams concluded that the fact of sharing his code with other people works as an extension of what he was doing, and "you got an extra pair of hands that can help so that we can deliver it faster" (Case#11, 2020), alluding to divide and conquer!

In general, there is a sense of motivation to improve and be an excellent example for future generations.

> "Hmm, so it, yes, this also in terms of code, it changed me as well. I try harder, I tried harder." (Case#16, 2020) answering a question about change in code (Question#18)

This motivation comes from the desire to create processes and templates that work to elevate and help the people who are now leading – experimenting with what works and what does not in the quest for constant improvement, listening to what others have to say in constant feedback. I can say that programmers feel that, when they became managers, this change improved their perception of the world of programming and that there were significant improvements in their performance and in the way they produced code and see the code produced by others.

9.6.3. Clash of the Egos

A people manager must have a very high degree of humility. This humility is one that comes with time and that is discovered with the hindsight of years of experience. It leads you to put the interests of others ahead of your own and look to the community, your peers, and what you can create together. Instead of imposing its will and way of doing things, it takes a co-author attitude, helping others achieve shared successes.

> "The other thing is that while I do enjoy working, connecting and helping others succeed, a bit like a coach-like situation, the other angle is that I don't need to be at my ego fed, like, maybe I was in a younger self so much by having the need for direct reports. So I don't measure my worth by how

> many direct reports I have." (Case#8, 2020) talking about competitive advantage (version of Question#24)

Experience and maturity come with time. There is no shortcut. Moreover, it is clear that when the co-researcher reflects on their own trajectory until reaching the level of expertise and evolves towards the leadership of people, there is no need to show their ego.

Conceivably losing control over code is the biggest concern when transitioning to manager. Moreover, developers complain about this fact when managers demand that the code to be produced in their image and likeness. This is not the best way to work when we think about creativity and the will and ability of each developer. Following a particular way of creating code just because it is the boss's "right" way is demotivating. Furthermore, programmers want to have their voices heard.

Looking back, some co-researchers were able to see their own transformation in controlling their own code and could look beyond the code and develop personal skills that would help them to be more effective as managers.

> "So I'm ... I've changed, I've probably changed since we worked together. And when I'll sit with somebody I can, I can be a bit abrasive at times, and I don't realise it and I can be a bit bossy and say: No, do this, do that, do that, and I realised this now, and like, I can pick up on it much more easily. I think it's my soft skills more than my relationship with the code. But I'm at a point where I can say: maybe try this. Have you thought about trying that?" (Case#5, 2020) answering a question about the perspective on the code (a variation of Question#18)

This perception that, as managers, your relationship with people will produce a better result in terms of programming leads to more attention being paid to interpersonal relationships. Moreover, the lines of code go down in history, and anyone could write good lines of code, but the relationships last. This requires an openness to change, accepting different opinions and ways of seeing the code and the community in a different way, leading to collaboration between the different entities. Looking back and understanding patterns of behaviour and reasons for performing certain activities is the beginning of a redemptive path that is initiated through reflexive practice. Interestingly, Case#5 above shows an inability to delegate and a propensity to tell others what to do in a kind but ineffective manner. "Have you thought about trying that?" it is a way of imposing your ideas without really listening to what others have to say. An approach that would demonstrate a transforming attitude on the co-researcher's part would be to ask, "What do you feel are our options for this task?". This question would allow the subordinate to offer an alternative solution, which the manager is not imposing. In a deeper analysis, the question could be posed: "What have you tried so far and why do you think it is not optimal?”. This text sample reveals an inability to lead and a lack of preparation that became notorious in this study.

> “I did some terrible things like, I've had bosses who done this to me in the past as well. I can't believe I started doing this but, you know, you do bad things, you check people’s code and rewrite it and resubmit it, and, I mean, you know, when that used to happen to me I would be mortified and then I was doing myself within 3 years, and that was an awful thing to do.” (Case#6, 2019) answering a question about code perspective (variation of Question#18)

In summary, we can see that one of the aspects documented in this study regarding developers becoming managers is the clash of egos – a willingness to control others and impose one’s will. It starts from an assumption that there are ways to control other people. In fact, they see themselves as master puppeteers who can pull the strings and make things happen. In the real world, things do not happen that way. It is an aspect that must be acknowledged, internalised and changed so that there is a more effective collaboration in a work team.

9.6.4. Community Development

There is a human side to programmers, especially in those who become managers. Perhaps out of self-fulfilment or out of necessity, but the soft skill of knowing and enjoying working collaboratively with others is part of the journey of becoming a manager from a programming career. Furthermore, whoever took the leap seems to appreciate it or at least positively tolerate it.

> “I really, I got satisfaction from working with people and making sure that people on the team that I was managing were happy. I really, I went home feeling very rewarded and that was probably, touching something human, you know that, that's probably, probably I felt good about that.” (case#2, 2019) answering a question about achievements (a variation of Question#26)

The fact that some programmers have made the leap to managers and genuinely care about their peers debunks the myth that all programmers are cold and calculating. No, there are some programmers with a heart who can work well with others.

> “And about, you know, dealing with people is the most important thing that you do. So that's what I think it does give you a voice. I think it gives you even if you're less technical than somebody else around you. If you get on with your peers and superiors, then you can still have the loudest voice.

> Does that make sense?" (Case#5, 2020) answering a question about competitive advantage (a variation of Question#24)

The relevance of the programmer and manager is in the relationships and validation by the community. In organisations, work is always done in the community. It can be created individually, but at a certain point, it will interact with other processes and people. Moreover, the most important relationship is with people. It is what defines the voice and the way it is heard.

I want to expose three factors that bounce out at me when I look at the data exposing the fact that programmers-turned-managers show when they are interested in people: the acceptance that there are different perspectives; the ability to persuade and convince others of the best ideas; and later involvement in the desires and goals of the other people you are leading.

9.6.4.1. Acceptance

The first realisation is that everyone thinks differently when we deal with people. This means that, in terms of code, the solutions to the problems will emerge in different ways, as they will come from different heads. Moreover, every mind thinks differently. It is essential and necessary for us to accept that people approach problems differently than I, or you, approach them so that we can accept different proposals for the same type of problem.

> "With being a manager, manager in this, you always like ... when you deal with people, then there's always different perspective." (Case#16, 2020) answering a question about achievements (a variation of Question#26)

Programmers have a voice, and they want to be heard and acknowledged, as mentioned a few paragraphs above (please see chapter 9.5.1: "A voice").

9.6.4.2. Persuasion

As each person thinks and acts differently, and there is a need to accept, on the one hand, and on the other hand, communicate ideas effectively for the different levels of skillsets of each one. This ability to convince others of different ideas takes many forms, including the gravitation toward the people around and the interlocutor's ability to communicate. It has already been mentioned in the previous paragraphs about managers' own egos putting themselves ahead of project and code interests and how they try to convince others that their ideas have an intrinsic value. There is always a need to communicate points of view well, convincingly, and fairly. Furthermore, in the case of people, it is necessary to resist the urge to impose your own will.

Persuasion travels both ways: convincing others of your ideas or convincing yourself that others' ideas are better. Some have demonstrated that they can find this balance point and have made the right decision, not without first going through a learning process that is only possible with time.

> "Everybody's strengths are different and actually, I think, as being a leader for longer and longer I had less of a flattering view of my own abilities and more, a lot more respect for other people's abilities, and these days I am far more likely to accept somebody else's approaches to a problem than use my own solution to a problem, largely because often making the right decision. I am a better manager now but it is taken a long time to get here." (Case#6, 2019) answering a question about code perspective (a variation of Question#18)

As time passes – and time here is a good counsellor – and new experiences are added to the programmer's and manager's history in this journey, maturity and experience come to the fore, showing the positive evolution that is possible to have in interpersonal relationships and in persuasiveness.

9.6.4.3. Involvement

Managers are responsible for involving everybody on the teams. The responsibility to create an environment where everyone feels included is everyone's responsibility. I, in particular, have a philosophy that, if I have to interact with a larger team, I want all the interactions to be positive with everyone. This requires the extra work of understanding everyone, a little emotional intelligence to "read" other people, and treating everyone the way they deserve. To treat others in a dignified manner is to be an inspiration to others and to be a helping hand in time of need. Getting involved with people is also thinking about their future and planting in them the desire to be better and achieve their passions.

> "And so there were some things where, you know, that's too complex for that person, they won't do a good job of that. Or so yeah, finding the right things. But then what you don't want to do is always just give the really simple things to the least skilled team member, you want to be finding things that are going to push them and so that became a big part of was like, trying to think about the problem. How do I find the right ones for the right people? So I'm always pushing people forwards." (Case#7, 2019) answering a question about code perspective (a variation of Question#18)

This genuine interest in team development bears fruit long-term, keeping the team motivated and united. This is equivalent to mentoring and was mentioned as a factor of personal satisfaction and a legacy that this group values: leaving a legacy of trying to bring out the best in people.

9.6.4.4. Conclusion

After uncovering the three important aspects regarding the people skills of programmers who became managers, I would like to present the connection between creativity in programming and the fact that we look at creativity as a driver of productivity.

Programmers who become managers are a particular cog in defending the interests of programmers in the hierarchical chain of organisations. They have the technical knowledge of how things are done and how to do them and the management skills that can make things happen. These are the ones that took expertise to the next level – literally. When we think about the exercise of expertise, more than ever, they can use intuition as a process accelerator and be the catalyst for creativity and innovation in their sphere of influence, if they are inclined to do so. Because being a manager has these two aspects – catalysts or blockers. If one looks within themself, in genuine introspection, and learns from one's mistake and those of others, one can improve and increase the performance of others, giving them freedom of thought and execution, or else, in the opposite direction, having mere extensions of your thoughts, reducing programming to a pre-defined code factory. Here is the choice that we face today.

9.7. Summary

In this chapter, I looked at expertise and how experts have a privileged position in relation to creativity since they are the ones who have the power to execute the best ideas and take the most significant risks. I looked at the evolution of the concept of expertise as I progressed in this research project by listening to the co-researchers' opinions regarding their own expertise and comparing it with those who thought they did not have expertise in the area of programming. Some factors prevented co-researchers from calling themselves experts, such as the constant changes in programming paradigms, the knowledge overflow, the lack of confidence, the absence of practice and the monotony in some projects. There is also the notion that you cannot be an expert in everything. Another important theme in this chapter was the development of intuition as a sign of conscious and unconscious expertise. Next, I developed some attributes of experts based on interactions with co-researchers, namely the fact that programmers have a voice, are able to take direction and lead, are multifaceted with knowledge of different programming languages and are experts in collaboration. This collaboration often leads to them being "promoted" to managers. I explored some challenges programmers face when placed in the position of people managers. After analysing expertise-related factors, we move on to innovation, the product of the experts' work.

10. Innovation

The word innovation is used interchangeably with creativity and vice versa in this project since, in the interactions with the co-researchers, the differences were not explained to them. Moreover, for most people, they are considered the same thing. Although I try to distinguish between the two in the professional setting, in the "Knowledge Landscape" chapter (3) I explained the distinction between creativity and innovation. Creativity is related to finding new things, more associated with the context of creation (Cropley, 2015), and innovation is associated with the search for new ways to link existing things to get something completely new.

> "You have a choice what to, you know, choose which direction … " (Case#13, 2019) answering a question about creativity (Question#15)

Software developers have the ability and possibility to choose the path they want to follow in terms of creativity and innovation.

> "I think there is still a lot of innovation in language design. It is just that maybe happens on a slower pace now because things are currently tied to mission critical ways of doing stuff that we already got. And people are less likely to want to take risks with totally new ways of approaching language." (Case#6, 2019)

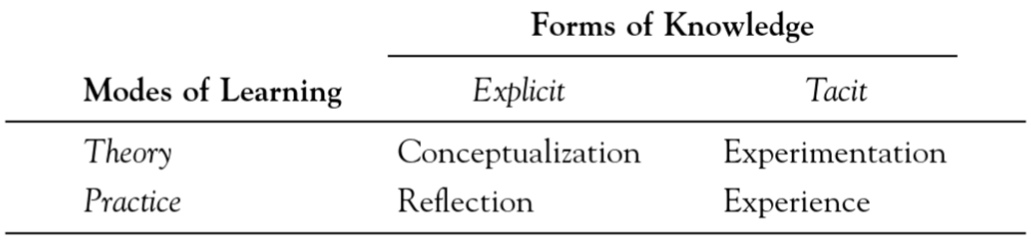
One of the ways to exercise creativity is through the choice of programming languages. There are similar ways of programming within most programming languages, but also different languages appear because they try to solve different problems that were not foreseen when other languages were created. Here I have two properties associated with innovation in terms of programming language design: the time it takes to implement a change and the inability to take risks. These two dimensions are intertwined and reflect the inability to innovate when it comes to code and boils down to an aversion to taking risks. This risk aversion comes from the fact that one wants to avoid change in already established work bases, not to disrupt what is already working – maintain the status quo. Maintaining the status quo is missing an excellent opportunity to innovate.

10.1. Experimentation

Schön (1987, p. 70) explained the definition of experiments as the action to verify what happens next. When that action does not follow a prediction or expectation, then it is an "exploratory" only experiment. This exploration happens when we are faced with something new for the first time, and we want to explore without really knowing what the end result will be. A second type of experimentation occurs when there is a deliberate action to produce a specific change. Schön calls this type of experiment a "move-testing-experiment" with the final result being "affirmed" or "denied", depending on whether or not the desired result was achieved. The third type of experiment is that of "hypothesis testing" in a discriminatory way in relation to competing hypotheses, verifying whether or not the result is confirmed (p. 71). Schön argues that in the context of reflection, a practitioner can perform these three types of experiments in the same action.

> "When the practitioner reflects-in-action in a case he perceives as unique, paying attention to phenomena and surfacing his intuitive understanding of them, his experiment is at once exploratory, move testing, and hypothesis testing." – Schön (1987, p. 72)

Raelin (2008, p. 80), based on modes of learning (theoretical or practical) and forms of knowledge, created a conceptual model of learning or knowledge creation in the work-based setting. What caught my attention in his conceptual model, in addition to reflection and experience, was the importance given to experimentation. Experimentation is one process that makes it possible to bridge the gap between theory and practice. The learning process typically occurs through experience. As a specific experience is reflected upon, learning becomes internalised, reinforcing the tacit knowledge that was acquired through experimentation. Experimentation allows theoretical concepts to be tested in case studies, snippets of code, and other forms of learning and, when associated with experience, produce knowledge that the person may not even realise that he has assimilated something new – he may not know how to explain what was learned.



	Forms of Knowledge	
Modes of Learning	*Explicit*	*Tacit*
Theory	Conceptualization	Experimentation
Practice	Reflection	Experience

Table 5: A conceptual model of Work-Based Learning – Individual Level — Raelin (2008, p. 70)

This conceptual model is attractive because reflection is what transforms tacit knowledge into explicit knowledge, which is the basis of intuitive actions, a concept defined as fundamental when we talk about expertise (Raelin, 2008, p. 74). The co-researchers in this project were able to grasp this concept in their own way, and it is interesting to see how throughout the conversations, they used reflection more and more to understand their own weaknesses and also their strengths, as well as to overcome their own fears.

> “Hmm, so, for example, I recently started learning a new programming language. I started learning GO, and may even 10 years ago I would have felt quite unerved about diving into a new programming language as it is quite hard to do. Hmm, Go is actually quite friendly and is quite easy to work with, but nevertheless, you know, a couple of weekends of work I feel that I am familiar enough with GO now that I can start to apply it to projects and I can start to see how it works in context and in another context. Hmm, and I think that's because the fundamentals of what I do for a living are set inside me and I understand how to do them.” (Case#6, 2019) talking about expertise (version of Question#4)

Knowledge of the new programming language associated with a knowledge of himself led the co-researcher to overcome the barrier of learning a new programming language. Through reflection, the co-researcher opened up new horizons that were indeed registered inside, to be used with the opportunity to learn a new language should that situation arise.

The simple problem is that the co-researchers in this study do not go home at the end of a workday to do experimentation. They can even experiment and use their creativity in other areas, but not in programming.

> “... but again, nowadays, I am still creative, I still experiment. It is just not with code. It is something else. I have, yeah, other side projects.” (Case#16, 2020) talking about experimentation (version of Question#17)

10.2. Individuality as a differentiator

In my experience, reducing programming to an automatic set of rules and pre-defined puzzle-like components is returning to the pre-Industrial Revolution time of the 19th century. As a programmer, I want to use all my knowledge to produce the best code, the most optimised, and the one that best adapts to the function I am developing. In a particular project in which I participated, they wanted me to cut corners and use a pre-designed website in a production environment, whose template would have already been used by other companies and which created a lot of overheads and excessive code that was not of interest to the project. I refused to do it and did not want my name associated with the project. The reason may seem like a rebellion against automatisms, but genuinely, I wanted to create something that I could be proud of, that would have been what we needed for the project, and that would be optimised with the years of experience that I already had. Maybe the pre-designed template would have worked for amateurs. I remember that at the time, I used the chef analogy. Anyone can make spaghetti Bolognese, but we will eat the same dish in a restaurant because the chef will make his own tomato sauce from scratch, choosing the best ingredients and putting all his years of experience into preparing this same meal. I can try to do the same, but I will buy canned tomato sauce from the supermarket and choose the best ingredients I know (with limited knowledge), and in the end, the taste and experience will be different. It is not a mannerism, but when I am creating code, a lot of thoughts, techniques, ideas, and experiences are put into the project, which, despite the amount of artificial intelligence (AI) it has, was not explicitly created for what I am producing, but for a generality of situations. In that regard, and to obtain the best results, companies continue to hire programmers to create bespoke solutions to beat the competition and have a competitive advantage. The AI solution can be an alternative for an average project, considering semi-professionalism. However, at the top level, the solutions will always be bespoke created as they have a differential for users.

10.3. Time

Time is an interesting concept that was mentioned several times in the conversations, with variations in meaning depending on the topic. In my data analysis, there is not much emphasis on the time it takes to do things but on the lack of time to finish things with reasonable quality. Programmers need time to produce quality code.

> "We could have done it better if they have given us more time." (Case#15, 2020) answering a question about limitations (variation of Question#13)

Code development is pulled from sprint to sprint because programmers do not have enough time to properly develop their ideas on the code. This is a constraint on the quality of the project. Perhaps the criticism against the current agile programming model is that programmers seem to be on a conveyor belt where there is no time to stop and think, and no time to look back. The programmer is constantly looking for the next piece of code to produce next, worried about delivering more and more. From the customers or product owners' point of view, these are just advantages, and the agile model has delivered, and that is why it continues to be adopted. There is no doubt that the fact that there is continuous delivery of working software, changes are being welcomed at any time, early feedback is given, time constraints metrics are supported, etc., make this method the most appealing for development nowadays (Beck et al., no date). Perhaps some of the principles are not implemented in teams, creating frustration.

> "... it's the place where they always focused on deliver more and more and more and more and more. It's like you're on a conveyor belt where you don't have time. You don't have time to look back. It's always the next one. next one, next one …" (Case#16, 2020) answering a question about their contribution (Question#21)

Time was also mentioned as a constraint on the exercise of expertise. As the years go by, family responsibilities grow, especially with the arrival of children. You feel that you no longer have the time to stay on top of your game. Moreover, it would help if you had time to research, learn new technologies, read programming articles, and above all, experiment. However, Ericsson (1993, p. 365) argues that time, by itself, is not a guarantee that a person will become an expert in a given subject. Nevertheless, by spending time intentionally on a specific subject, there is the possibility of improving performance and eventually becoming an expert. Moreover, with deliberate practice or not, there is recognition that time is crucial for developing their capacities.

> "... Okay, so, first of all I don't have, nowadays, I don't have time to experiment on … in my own time, in my own project." (Case#16, 2020) answering a question about experimentation (a variation of Question#17)

When thinking about the time required to reach a degree of expertise, there is also the possibility that, in reality, some programmers do not want to spend twenty-four hours of their day dedicated to programming. I do not know if achieving professional success will be possible without extreme dedication to the profession. The co-researchers shared this idea:

> "So, to be experts, I should be able to do this non-stop, like, 24/7. When I go home, I should, you know, play with different things. I don't have this time. To be honest, I don't have this desire as well. So, I'm definitely not an expert. I'm good in some certain area, but I don't believe there is an expert, or if there are some, they very few people." (Case#13, 2019) answering a question about expertise

As mentioned in section 9.2.4 above when discussing programmers' feelings about their expertise, some programmers feel they need more time to develop themselves. The time comes with the possibility of having more insights into the problems they face. And, over time, they will solve many more problems if

they want to become programming experts. A familiar feeling that appears in the data is that programmers want to “be challenged, but not overwhelmed” (Case #8, 2020) – they would like to find a balance between having enough challenges not to be bored, but at the same time not being so overwhelmed that they do not have time to experiment and innovate.
In short, the career of experts is only available to a few, and they are those who are willing to sacrifice more than others.

10.4. Diversity

There is a consensus that one should have more female developers and more women in STEM areas because of the obvious advantages. What is still not well accepted, and I want to change that, is the fact that one also has to look for diversity in terms of different universities and different courses as a background to enter the profession of software developers. I had conversations with several programmers whose higher education was in areas utterly different from Computer Science. One of the factors mentioned in conversations that increases creativity in a team is having people with different backgrounds. The reason for this connection is that they do not see things the same way that other people do. Many of these programmers from other professions approach programming problems in a completely different way without worrying about the courtesy offered by predefined ways of doing things, and eventually come up with solutions that may initially seem stupid but are much more intelligent than “traditional” solutions.

> “ I don't know if it's the most important, but it's a big factor in creativity. So this this randomness, backgrounds. If you want to be creative in a field, if you want to create disruptive in an industry or build new project, you need to have a diversity.” (Case#4, 2019) answering a question about different backgrounds (variation of Question#28)

If you take a doctor, a lawyer and a builder, and teach them how to program, so that they are software developers, the code produced by each of them will be different from what other programmers with different backgrounds will produce. Potentially they will bring different insights and some disruptive ideas for programming.

Usually, when someone transitions from another profession to programming, they will have to learn the programmer’s craft. In the section dedicated to Learning & Development, I address the topic of how programmers reached their current level of expertise and reflect on self-learning and learning by doing (see section 12.2 below “How they reached their level of expertise”).

> “I think self taught programmer they do approach problems in a different way. I think they approach them the less hendy by predefined ways of doing things and they might find stupid ways of doing things, because sometimes they find smarter ways of doing things.” (Case#6) talking about extra things (version of Question#29)

This excerpt from the data reinforces the idea that programmers with different backgrounds approach problems in a different way than those who have had formal education or training in programming. I have a suggestion here that programmers who have taken on the task of training themselves, because they come from a different background than traditional programmers and bring diversity to the team, will be less likely to use pre-established methodologies and will have a greater inclination to emerge with innovative solutions by unconventionally approaching problems. There is also a link to creativity and innovation here.

10.5. Gender

According to the BCS (The Chartered Institute for IT), a recent analysis from the ONS (Office for National Statistics) in the UK in a report published in 2021, women make up only 19% of the IT sector when in reality, they represent 50% of the working population. The representation varies according to the job type, and the rate for web designers/developers is around one in three (35%) (BCS, 2021b). Having a diverse range of perspectives and experiences helps foster creativity and innovation. One of these different perspectives comes from a gender-balanced team, where we look for different ideas to generate creativity and innovation. A gender-balanced team takes advantage of the fact that men and women have different preferences and requirements regarding technology.

10.6. Summary

In this chapter, I explored the topic of innovation by exploring the differences in relation to creativity. Here, I begin to develop the idea of experimentation in the development of creativity in programming, followed by the importance of individual programmers as being a differentiator in relation to Artificial Intelligence or the use of pre-defined code or templates. The concept of time is treated within the theme of innovation, both as a constraint and a catalyst for innovation. To conclude, we have diversity, not only linked to gender imbalance but also to programmers with diverse backgrounds in terms of study areas or life experiences, who see and solve problems differently. Expertise and innovation are the connecting links between creativity and the Programming Code. Together, they produce the impact associated with feelings, which we will examine in the next chapter.

11. Impact / Sentiments

As we will see below that reaching an "Accomplishment" (section 12.1.1 below) was the main reason that the participants of this project gave to start a programming career, one of the important data is the achievements they reached throughout their lives. During the conversations with the co-participants, I always asked questions about their achievements and the impact their work has had on their lives and the wider community.

I want to mention that linguistically there seems to be a slight difference between "accomplishment" versus "achievements". Achievement typically refers to something that was gained through effort and skill, while accomplishment refers to something that was also achieved through personal effort and skill but also denotes something that may have been completed through luck or chance (Collins Dictionary, 2011). In scientific fields such as Psychology or Education, this distinction is also made with the additional nuance of adding other levels of interpretation. On the one hand, achievement is associated with a quantitative, externally imposed measure that benefits oneself, and accomplishment is associated with a qualitative, self-imposed measure that brings personal satisfaction but ultimately benefits society in general (O'Grady, 2012; Prensky, 2015).

11.1. Personal lasting impact

Talking about themselves and the impact they eventually had on their careers as a programmer is not easy. On the one hand, it suggests that they are at the end of their careers, which is not the case for any of the co-researchers, and secondly, because it may seem presumptuous.

> "Making a life out of it [laughs]? I don't know. Hmm, it's, yeah, it's just, sticking around for such a long time and continue to being a developer, maybe" (Case#16, 2020) talking about achievements (variation of Question#26)

Most participants mentioned that their most significant accomplishment is the personal lasting impact programming has had on their lives. Consequently, the word accomplishment takes on a new meaning because of the admiration and disbelief with which they mention that they have been working in the area for so long and feel that they still have a lot to give, which in itself is an outstanding achievement. It can also reflect a sense of dedication and commitment to the profession they have chosen to pursue. Not to mention the fact that they can make money doing something they really enjoy doing. It can be inferred that they still have a long career ahead of them because they also mention that the project they are working on at the moment is their big achievement – they still look to the future as a time when they are still going to do something great.

> "Staying in the game for 20 years and not get moved into a management position. Hmm, I am delighted that at the age of, coming up in my mid forties [age changed], hmm, I haven't stepped into a hands-off position." (Case#6, 2019) talking about achievements (variation of Question#26)

For some, an achievement is being able to remain pure in programming and not being convinced to pursue a management career over the years. For those who like what they do and want to keep doing it, this is a challenge when you are an expert in your field. I can see a desire to maintain hands-on contact with programming and to remain involved in the technical aspects of the work that shines through here.

Furthermore, it is not just longevity that is important to co-researchers. It is also the fact that this longevity is accompanied by experiences in different industries with the possibility of working in different areas.

> "I worked in different business, technologies. I worked in utility, I worked in agencies, like commerce, ecommerce. I worked at the hospital pay business, now I got into the trading business. So, I think that itself is an achievement. I know how it works in different businesses, so that is interesting." (Case#15, 2020) talking about achievements (variation of Question#26)

I find here a broader perspective of knowledge of the industry where it operates as well as an ability to adapt to different environments and challenges.

> "So, I imagine anybody who looks at the player, the [company name removed] player code - all the good players are difficult to write - will remember me, maybe hate me a little bit, but then once they try and do it themselves, they'll understand why it was so difficult." (Case#7, 2019) talking about legacy (version of Question#25)

The most significant impact is leaving a legacy for those who will follow. There is pride in accomplishing a difficult task and a desire to leave a good impression in their field.

Overall, the impact of these achievements is primarily personal, reflecting individual goals, values and desires. However, they had implications for their individual careers, their reputation and the influence they had on their field of work.

11.2. Reflecting on the future

In this project, developed alongside the co-researchers, I realised that they started to use reflection and to see its importance in a personal and professional context and how it changed their lives. The use of reflection by the co-researchers was a pleasant surprise.

> "The world of development was been a little dynamic and there always been new approaches doing particular jobs, but it did feel like 2015/2016 was a particularly bad time for this. It felt like the growing pains of the industry really finally starting to really kick in. Ahh, but actually these days it doesn't feel so bad. It feels more like … if not, we are not standardizing on a particular framework, or a particular library, I don't think. I don't think that. I mean, react is probably a dominant player in the market, but it is not the only way of doing stuff. It feels like that speed of change has started to reduce a little bit, and you can finally catch your breath a bit. It is not quite as bad as it was. Hmm, there are defined good ways of doing stuff at the moment which they weren't a while ago. But to get, to cope with that problem at the time, honestly, it was just a case of read blogs, read blogs, read post, try stuff, talk to people, try stuff..." (Case#6, 2019) talking about changes (version of Question#23)

What we find here is a reflection of my co-researcher on programming development in a given time frame, in a moment of instability in the industry that was caused by the emergence of new technologies and frameworks. The co-researcher reflects on the overwhelming feeling of how quickly changes occur and how he is dealing with that feeling. After this phase of constant changes, there is an improvement, and the speed of changes slowed down a little and found ways to have some security and stability. In a way, this instability brings a unique opportunity for personal development and also a way out for the cognitive entrenchment mentioned in the section on expertise (3.4.1), where the lack of some flexibility in experts is associated with the existence of a high degree of stability in the domain schema of the expertise (Dane, 2010).

From a broader textual perspective, this conversation focuses on the dynamic nature of technology and how programmers navigate and respond to change. It also shows the pathway in terms of solutions for an individual journey of how to remain relevant in a changing world through personal learning strategies. In a deeper and more personal analysis, the answer reflects the emotions experienced in times of change when one feels more stressed and under pressure. We find these feelings in the words "bad", "pain", and "catch your breath”. The way in which he was able to reflect and overcome these feelings is expressed in the terms “not so bad”, “feels more like”, and “defined good ways”.

There is an encouragement to use reflection when working within the “agile” framework that was mentioned above when reflecting on the “Time” (section 10.3). One of the things that is advised is to look at what we are doing well and continue on that path; look at what we are doing wrong and stop doing it; and look at what we are not yet doing and should be doing (Cohn, 2010, p. 283). These concepts are so entrenched in programmers’ minds that when they see something interesting they are doing, they want to create a snapshot of the moment for later analysis – the reflection.

> “And, yeah and I said to the team there ‘we should write down on paper all the things that are going right on his team right now’ because it felt very fleeting and felt like I was not going to last and things were going so well that it should be documented for the sake of humanity whatever ‘why did this feel, why did things work so well on the team’. It wasn't a very large team of, 20, 25 people. But, yeah, I think, I think that team, for some of those reasons, yeah.” (Case#2, 2019) talking about creativity (version of Question#20)

11.3. Summary

This chapter has two facets. On the one hand, we have the personal impact that a career in programming produces, and on the other hand, we have a reflection on the future. Reaching an accomplishment was the main reason that the participants in this project gave for starting a programming career, but the analysis is not only focused on the past but also on the future and everything they still hope to produce in their careers.

12. Learning and Development

One of the topics that engaged me when I embarked on this academic project was trying to understand how interest in programming arose among the participants. I was able to realise throughout the study that there are factors that influence creativity and innovation, and among them was “Diversity” (section 10.4 above), including “Knowledge of Different Programming Languages” (section 9.5.3 above). Each participant had a different path to becoming a programmer, and the career paths of some intersect and form a pattern. In the following paragraphs, I will explain the reasons that led to choosing this profession, analyse the trajectory of the participants in this project, how they reached their level of expertise, and what they do to continue to be relevant in the profession.

12.1. Reasons for choosing this profession

In all the conversations I had with the participants, I started by asking what happened that made them decide to become programmers. I was looking for a definitive answer that could lead to a paradigm shift in terms of training, curriculum and how to teach programming. The response came much more mixed. The interest in the programming profession varied considerably among the participants. The data were crossed with answers to other questions, and of all the reasons taken from the theme analysis, two stand out more than the others: “accomplishment” and “solve problems”.

12.1.1. Accomplishment

The desire for success is the most mentioned among all participants. It is a very personal feeling, bordering on selfishness, but that comes with a job done and an internal reflection that it was worth doing what they like, but also what they probably did not like either. It is a satisfaction accompanied by recognition from others, or perhaps the achievement of having fulfilled a personal objective or having presented a completely different angle that had never been considered before. It can be demonstrated by having your software installed on devices all over the world, or having managed to win an award and have drawn attention, or having worked on a product with a big budget and everything, or simply seeing the outcome of your work.

Accomplishment is intrinsically linked to recognition. The recognition that the job was well done makes you feel intelligent and proud for having managed to accomplish the task. Moreover, recognition is motivational. The accomplishment was what the participants in this study were looking for when they decided to embark on this career. I can say that being the most mentioned reason, they indeed found what they were looking for.

> "I mean, it can be frustrating when you hit a wall, but then it's a lot of fun when you discover ways to do things, in other ways, and it ties in really nicely." (Case#3, 2019) talking about frustrations (variation of Question#7)

In the same way that it is frustrating when you do not find a solution, the opposite is also true, since when you solve both the biggest and the smallest of problems, it brings incredible satisfaction. Moreover, when the code does what it is supposed to do, depending on the way you write the code, the very way it looks can be a source of pride and satisfaction.

> "It's changed. I think when I first did it, it was the sense of achievement that I could do this. I could programme ... make, make it automated. And that was quite an amazing feat when I was younger, when I first grasp programming." (Case#9, 2020) talking about freedom (version of Question#6)

While in the beginning, the simple fact of being able to carry out a programming task brought with it a feeling of accomplishment, over time, only the final product brings the same level of gratification.

12.1.2. Solve problems

Solving problems is one of the reasons participants gave that motivated them to pursue a career in the information technology area, but also one of the most liberating and challenging in terms of coding. The most liberating thing is when you get stuck on an issue – I am not talking about something that blocks you for a few minutes, but for hours – and you try all kinds of things to attempt to solve it, and it is just not working, then you find the perfect solution. That is liberating. There is this feeling of elation at first and then satisfaction. The satisfaction is a hooker to get people into programming.

> "For me, the most liberating thing is when, you know, all your problems solving, everything on that code going through, every stack exchange, and, you know, even sometimes sneaking into other developers. I think it is good when you finally solve whatever caused that issue. I think that is the most liberating for me. Yeah." (Case#1, 2019) talking about freedom (version of Question#6)

If a person does not like solving problems, then one should not enter the programming profession. Finding the solution to a problem involves perseverance and the possibility of experimenting, trying different solutions, trying different codes, trying different tools, and having the opportunity to create something new, which is ultimately rewarding.

> “It's something that's constantly changing and, and it's nice at the end of the day, when you, even when you solve one small problem, you feel like you have, you have achieved something. So this is very rewarding. This is what I like about it.” (Case#16, 2020) talking about freedom (version of Question#6)

> “You know, I like solving problems, but it's not sexy. And if I could do something else I would, but I don't know anything else that it would challenge my mind and exercise it and the way the programming does.” (Case#9, 2020) talking about competition (version of Question#24)

The programmer has the ability to code to solve his own problems. If one encounters a challenge or sees a niche market for something one lacks, the programmer can always step up and create a solution for it. The solution is worthwhile if it is for the improvement of humanity, but if not, at least it works for the programmer. Over time there is a shift in the place where one wants to solve problems. Usually, the developer will move into places and projects where the programmer is asked to define what the problem one will solve is in reality. Moreover, that is great problem-solving.

There is a link between problem-solving and creativity. It takes creative thinking to find the solution to a software problem. Although at the time, finding a solution looked more like following a set of processes until a solution arose. Furthermore, the code looked like a series of problem-solving snippets resolved in sequence until the desired solution was reached.

> “I think it's, ... again, depending on the task that you're, you're working on. I think it can be very creative, because it's problem solving and problem solving is, I think, a creative process. ... Yeah, I think it's highly creative. ... Done. There you go. That is your answer.” (Case#12, 2020) talking about creativity and coding (version of Question#15)

And finally, solving problems under pressure adds a new level both in terms of focus and hard work as well as adrenaline in the solution. I get the impression that programmers get hooked on this exhilaration that happens when everything seems lost and creatively comes up with a solution to the problem.

> “... actually solving problems under pressure is a good fun thing to do, and I enjoy doing it. And I like, I like typing stuff in and seeing a solution come out as the right solution. And I enjoy it doing

> that. Hmm, I don't want to stop doing that." (Case#6, 2019) talking about achievements (version of Question#26)

12.1.3. Other reasons

Among other reasons for the participants to want to be programmers is a desire to "control the outcome", which is revealed in how a programmer can change code and obtain different results. There are those who like the process, but there are also those who like the details. Some like to create and own things alone, and some like to create together and share. There are people that only have ideas for software development, and there are the ones that have ideas and are able to develop them into software and apps. There are those who want to make an impact and those who just want to have fun.

Access to computers even before entering university helped some to make this choice for their professions. It went from code created at College to extensive programs run at night, including complex code related to spreadsheets and lots of data.

> " I built it an ASP. Classic. dotNet. VB, not dotNet. It was ASP classic. And then from there, I thought, yes, I like this. I spent a huge amount of time on it. What I used to do is I got hooked basically. And I used to go home after doing my lectures, and I would sit on the laptop ... so I do my other work and then I would sit on the laptop from that nine o'clock in the evening, eight o'clock in the evening. I wouldn't have any TV on or anything. I just sit there and I would vegetate in front of the computer learning how to do stuff for about six or eight hours." (Case#5, 2020) talking about the process of coding (version of Question#2)

And finally, there are those who think they were destined to be programmers from the beginning of their lives.

> "It was an area that I was quite keen to pursue, so, I guess, it's hard, it's hard to say when I started thinking like this but I guess I've always thought about programming. I always had programming in my life." (Case#6, 2019) talking about acquired skills (version of Question#3)

12.2. How they reached their level of expertise

Participants used various forms of learning to get to where they are today. My Learning path has been diverse, and I had to adapt, evolve and change to be on top of my career. Because of this constant changing in technology, environment, patterns, computer languages and functionalities, I never stop practising, studying, and developing new skills (Loback, 2018). I would like to analyse some of the learning paths used by the participants, starting with the most mentioned form throughout our conversations: self-learning.

12.2.1. Self-learning and learning by doing

The self-learning process begins with analysing what one needs to work in the chosen field. It is not about knowing everything but knowing what is relevant to one's work area. To be self-taught, one must also be observant and see what other people are doing and how they solve their problems, as one will have to do the same. Imitation is part of the learning process. For this, it is necessary to see the standard, the structure and the way in which the peers develop their codebase. Most importantly, with the advent of the internet, there is a ton of free, open-source software and code that allows one to be jealous of what others do without being awkward and secretly aspiring to be like them.

> "... because this is what I wanted to do, so, I thought, okay, if I could have a portfolio of these beautiful applications, put my source code out there, you know, I will start to approach being as good as this guy's." (Case#10, 2020) talking about skills acquisitions (version of Question#3)

In the section dedicated to "Lego Building" (section "7.4") I gave examples of several websites where developers can use free code. This is one of the best ways to see how other programmers have solved certain types of problems and use the same solution or an adaptation of it. It is also a great way to learn by doing.

> " I've just play on the on the computers and you type in programmes and learn your own." (Case#8, 2020) talking about skills acquisition (version of Question#3)

And, as a programmer, you have to keep learning, always. It is constant learning. Moreover, the question is not whether to continue learning but where to look for places to learn and what to learn next. One has the possibility to choose what one wants to learn next and prioritise personal perspectives and the individual development of one's skills.

Usually, what a programmer learns next relates to a need at work. For example, I started my learning process for a subset of a programming language that I recently started using in my work. Initially, it did not seem too complicated, but it became clear in the first few hours of use that a knowledge gap needed to be filled as quickly as possible. That meant putting in a few extra hours away from the workplace on tutorials, listening to podcasts, reading, analysing computer science papers, and reviewing other people's code to get a sense of the subset's logic and why it is a variation of the original language. In addition to doing my research online and watching tutorials, I also bought a physical book to have more organised access to the sub-set of the language. Reading books influences the way of thinking and finding efficient ways to achieve things. This opinion is also shared by the participants, who find books to be an escape from the time they spend in front of the computer.

> "A lot of it was self taught. So you pick up skills as you go throughout - throughout your journey, right?" (Case#5, 2020) talking about skills (version of Question#3)
> "So it was almost entirely self-taught, but with help from … someone … who is able to point me in the right direction of where to look." (Case#7, 2019) talking about skills acquisition (version of Question#3)

In another section (9.5.3 above, I talked about "Knowledge of Different Programming Languages" as a factor that generates expertise. It is interesting to note that exposure to different teams and projects also leads to exposure to different ways of organising and managing projects as well as different programming languages. Just choose your next role in an area you have not worked on before or have not had experience with yet, and that will be the best way to learn.

> "Well, mostly from my working experience. So it's by changing jobs, and being, you know, exposed to different teams and different projects. If I stayed at my very, very first job, I would probably, you know, go higher in the ranks, maybe become some kind of head of department and not do programming anymore. But that was kind of jumping from job to job and getting more exposed to new trends and technology. So it's mainly by, you know ... diversity of the projects that I was involved in … yeah, have hands-on experience and try things and learn by actually working and experimenting and building new things." (Case#4, 2019) talking about skills acquisition (version of Question#3)

The world of work is a catalyst for learning. It is a motivation, and it is also a necessity. When one is faced with a problem in a task, the best thing is to learn as much as possible about that particular area. Furthermore, experimentation tends to happen while a person is being paid to perform a programming task.

> "... and then I learnt it all on the job" (Case#2, 2019) talking about learning (version of Question#2)
> "I cannot read something and just understand it straight away. I have to do, so I will read something and if I haven't tried it, I won't be able to do it. I need to sit there and do the thing that I'm reading

> about. And then I will, I will fully understand it. I need to immerse myself in it before I, before it clicks, if that makes sense. Some will have an innate ability to, to understand something straight off the bat, but unfortunately, I do not. I have to put a little extra work in" (Case#5, 2020) talking about skills (version of Question#3)

Hands-on learning is a crucial aspect of comprehension, and some people need to fully engage with what they are learning in order to understand and learn the subject. Through reflection on his learning journey, my co-researcher understood what works for him and concluded, through self-awareness, of the importance of active learning and experimentation in his personal development.

In transitioning to the next topic, I would like to mention that there were also those who only used self-learning to enter a programming career over any other form of learning. As we are going to talk about experimentation below, which is intrinsically linked to self-learn, it is essential to mention this trajectory here:

> "Did six months or so of no, no work and not understanding the industry. And just studying constantly: like, NodeJS and JavaScript and business and just everything. That's what I was doing, like full time for six months or so. And I got got my first contract gig through an agency in, in [city name removed] … it was full time studying and learning and everything, Yeah." (Case#12, 2020) talking about choices (version of Question#1)

12.2.2. Side project for learning and experimentation

A few paragraphs above, I wrote about "Experimentation" (section 10.1) as one of the pillars of innovation in terms of code. Participants agree with the premise of learning by doing, but some struggle to find time to dedicate themselves to simply experimental projects, in the broadest sense of the term. Paid work takes up most of the time, and there are few opportunities to learn new things outside of working hours. For these, experimentation necessarily has to happen within the parameters of the working world. Experimentation in this setting is more limited than outside of work.

> "So for me, I don't have to spend more time away from, I mean, apart, not just this eight hours is enough to learn new technology, so because I don't have any other time apart from work to spend on ..." (Case#15, 2020) talking about creativity (version of Question#20)

> “ I always had something I was looking into for no reason, but I've decided not to do that anymore. I'm tired. I want to, I want my free time.” (Case#10, 2020) talking about experimentation (version of Question#17)

As programmers become more experienced, they begin to try to find a balance between personal growth and development and the demands of day-to-day work. At times, experimentation takes a backseat outside the workplace due to the desire to confine experimentation within the workplace to prioritise the most critical activities.

Whether one has the time or not, I would like to stress the importance of having side projects alongside daily paid work to experiment, learn and improve processes. Some of these projects were done simply for fun, but some became the main job. The objective is to discover bleeding edge technology, see what companies are using and its evolution, to always be on top, and understand if it is worth embarking on the novelty or not.

> “I generally like to learn things by starting a new project, so the craziest example of that is I learnt Python and Jango because I thought, “this looks like it’s worth giving a go”, to create the first version of [company name removed], which still uses Python and Jango.” (Case#7, 2019) talking about skills acquisition (version of Question#3)

The fact that he had never used a particular technology in his day-to-day work was the motivation he needed to start a project. It paid it.

Experimentation outside the workplace allows one to explore limitations that one may encounter at work and explore some gaps in technical knowledge.

> “So I would just explore those things outside of work.” (Case#14, 2019) talking about challenging limitations (version of Question#14)

Having a side project also allows experimentation through the implementation of ideas and observation of results without the pressure that exists in the work environment, without the inherent risks.

> “Like, I just do it. I have some idea, go, implement, implement it, and then I see the results trying to see if it's good or bad.” (Case#13, 2019) talking about experimentation (version of Question#17)

> "Because if you don't have the greenfield project on a day to day basis, ... and it's all brownfield, then where else will you make those decisions where you're completely free and you're not thinking of all those restraints?" (Case#5, 2020) talking about constraints (version of Question#11)

And finally, the side project allows for personal development and the construction of new skills independently through self-direction. A side project allows you to experiment freely without the restrictions imposed on projects carried out in the business environment. As a consequence, the programmer can use all his creativity and imagination to solve complex problems, not being limited by any requirements. This experimentation also leads to the development of new skills.

> "... of my – I would probably say – may be 70% of my skills came from my own development, because you have free reign and you have to – you have to build it yourself, kind of thing." (Case#1, 2019) talking about skills (version of Question#3)

This discovery associated with experimentation shows the importance of exploring new things and acquiring new skills outside the everyday work environment. We find here an appreciation of experimentation that is the basis of side projects. In order to develop new skills and learn new technologies or skills, it is crucial to have side projects that allow the later application of this knowledge in day-to-day work.

In general terms, I would like to state that side projects are a way for programmers to explore their interests, experiment with new technologies, and acquire new skills that will benefit their personal and professional development.

12.3. Continuous Professional Development

In a section above, when I was writing about expertise, I mentioned the fact that there are many things I still had to learn ("Knowledge overflow" on section 9.2.2) and also some things I should unlearn in order for the knowledge to bring about a change and produce sparks of new things. A person can have much knowledge in an area, but you cannot know everything about everything in programming.

Developers in general, and contractors in particular, have the notion that they must constantly be on their toes in terms of knowledge to get the next gig. Unsurprisingly, you have to be used to continuous professional development. This in itself is a cause of anxiety and some frustration within the community.

> "I was always terrified that I would be going to an interview, into a meeting about a new contract and they mentioned a framework that I never heard of, and ask for my opinion about it, and I would be like, I have no way of knowing. I can't answer this question. Hmm, it makes you seem like that you are not an expert in the field that you were supposed to be an expert in, and that can be quite an exhausting experience." (Case#6, 2019) answering a question about coping with change (a variation of Question#23)

Here we have the idea of expertise associated with what a programmer knows as an accumulation of knowledge (Goldman, 2018). The nature of IT work leads to an overvaluation of explicit knowledge over tacit knowledge (Polanyi, 2009). This is reflected in the type of interviews given to candidates and the types of tests and examinations they undergo. Overvaluing explicit knowledge is a lazy way that recruiting departments use to get people for their projects, as it is much more difficult to quantify a programmer's tacit knowledge rather than all the experience accumulated over years of practice. When explicit knowledge is only evaluated through a test or exercise, recruiters are hiring not the best programmers but those programmers who knew how to answer a quiz, who took their time to research the type of questions from quizzes and focused their time trying to answer a certain number of questions that proved that their explicit knowledge was sufficient for the position in question. Explicit knowledge, by itself, cannot quantify the lived experiences, nor the insights that the programmer has about the most varied problems that are faced in real life. Nor can it prove the ability to solve new and different problems.

One justification for continuous professional development is the fear associated with the fact that all programmers are getting older, and sooner or later, there will be newer people in the workplace with entirely different ideas and who, as is to be expected, will challenge the status quo.

> "... a lot of the developers here are into their 40's and still doing the job that they love. That is great. Hmm, but I feel like that, that is an achievement, because sometimes you go into teams and everybody there is like twenty years younger than you, or fifteen years younger than you, and you, like, what am I doing, really? People look at you, like you, (mumbles) are their grandfather. Why are you here? Shouldn't you be …. (mumbles)" (Case#6, 2019) talking about achievements (a variation of Question#26)

A recent report by BCS – The Chartered Institute for IT in the United Kingdom showed that although the population over 50 years old accounts for 31% of the working-age population, they represent only 22% in IT, and in London, this percentage drops to 16%. If we look at web developers and programmers, they are just one in ten (BCS, 2021a). It is unsurprising that as the years go by, one starts to be self-conscious about this issue.

I have struggled with ageism in computer software for some time. For a programmer, it is when you reach the mid-forties and start to wonder if you can still learn new things and positively influence the projects you participate in (MacDonald, 2021). Ageism had some influence on the timing in which I decided to embark on this project, although I wanted to embark on a new challenge and open my horizons to new possibilities. All programmers want to stay relevant in their industry, the field of action and programming language, and the fact that there are newer programmers arriving in their teams helps bring innovation and presents an opportunity to develop new skills. Nevertheless, it causes some concern.

> "So I can do all those things at the same time, which is good stuff, because it develops a certain rhythm. And that being said, once again, there'll be someone out there better than me. Sure. Maybe younger. Maybe hungrier." (Case#9, 2020) talking about competition (version of Question#24)

The fear of not knowing a framework is associated with a gap in explicit knowledge which may appear to be a problem of lack of expertise in their field of activity or the very disarticulation of some tacit knowledge that is difficult to communicate in an interview environment. Competition in the workplace is inevitable, and with advancing age, ageism in the workplace can lead programmers to feel out of place and undervalued. It must not be forgotten that there will always be someone with a greater knowledge of the work to be done or a different perspective. This leads me to affirm and emphasise the importance of continuous professional development.

12.4. Summary

Following the reasoning of the previous chapter, in this section, I developed some concepts based on the fact that each participant developed a different path to becoming a programmer. Some intersections and patterns led me to develop a joint approach in terms of learning and development. Firstly, I looked at the reasons why the co-participants chose this profession. I looked at the desire to succeed, solve problems, and stay active in the community, among other reasons. I then analysed how they reached their level of expertise, more specifically through self-learning and learning by doing. The continuity of people in their programming profession depends on continuous professional development.

13. Summary

I analysed six themes that were identified based on the conversations with the participants that linked creativity and programming code: creativity, expertise, innovation, code and programming, the impact, and finally, learning and development.

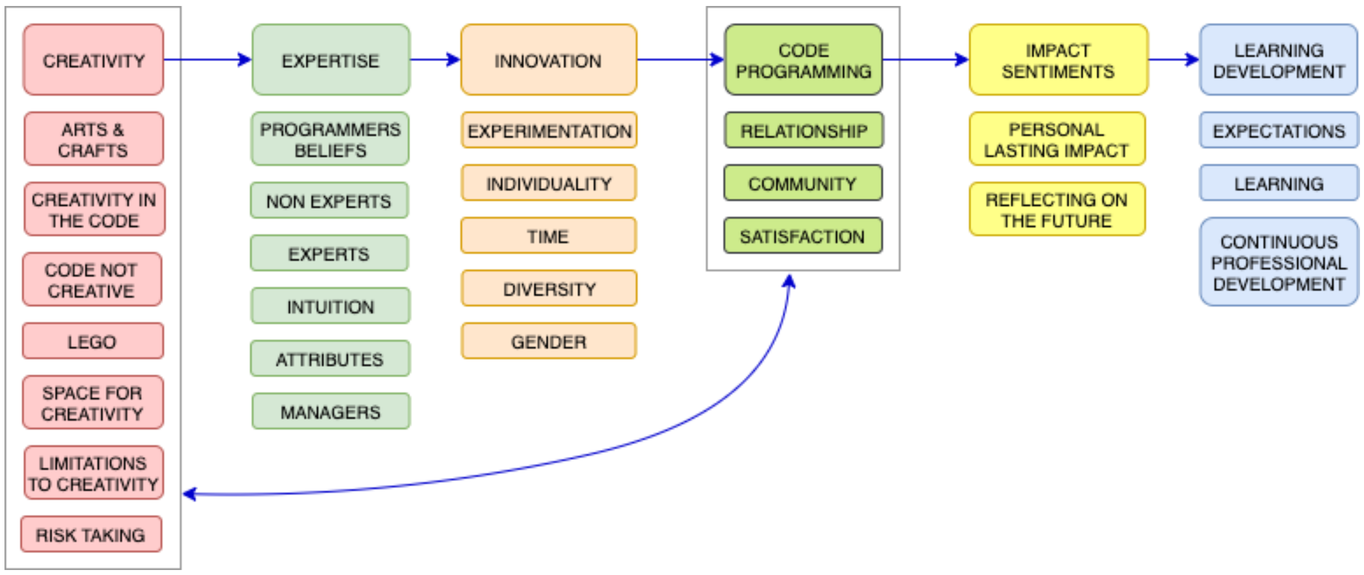


Figure 12: Alignment of the themes

When analysing CREATIVITY, I tried to answer whether computer code is creative. To answer this question, I entered into a discussion about the Arts & Crafts movement and through evidence, I positioned the creation of programming code within the art category. Through the way it is developed, I related it to craft since, after a long process of learning and reflection together with the skills acquired over the years, the code not only goes through a creative process, where the stimuli are organised but become a utility having a function once complete. Within the discussion on creativity, there was room to perceive that programming has no creativity simply because programmers do not have this perception when they think about lines of code. This is also partly due to the use of community code in projects nowadays, with the internet promoting this collaboration which exacerbates the previous feeling.

Nevertheless, beyond this fact, creativity also is needed to put together an application with the various pieces taken from the community. It was also agreed that it is necessary to create a space within organisations so that people can use their creativity. This was based on the Literature review analysis and the data analysis produced in conversations with the co-researchers. Related to the need to create a space for creativity is the relationship between creativity and risk-taking. We should create places where programmers can take risks without jeopardising other aspects of enterprise software. We need experimentation and free reign to innovate and take risks in these spaces. I looked at limitations to creativity, such as the rigid structures of organisations, lack of trust in experts and the limitations of programmers themselves, amongst other limitations.

The link between creativity and code goes through the other two themes, EXPERTISE and INNOVATION.

EXPERTISE occupied a prominent place in this project because it was through the experts that I wanted to understand creativity in programming. In this analysis, I looked at the co-researchers perception of their own expertise and the various aspects and characteristics of expertise associated with creativity and programming. It was evident that some aspects undermined the perception of expertise, namely the constant changes in programming paradigms and the amount of knowledge necessary for co-researchers to feel like experts. I realised that, in terms of programming, you cannot be an expert in all areas but that there is a long way to go in terms of experience and hours of practice to reach a high level of knowledge. Intuition plays a vital role because it is one of the characteristics of expertise and consequently connects both to aspects of creativity, such as risk-taking and creative freedom, as well as producing the necessary innovation to be used in the creation of lines of code. It was challenging to understand the extent to which the co-researchers were engaged in the use of intuition because it is difficult to articulate a tacit concept. In understanding the experts in this project, practical discussions about expertise emerged, such as the desire to have their voices heard and the importance of experts becoming managers and taking their leadership to the highest ranks. As managers, they can manage programmers' risks better and be their best advocates.

INNOVATION is a bridge between creativity and the lines of code produced, in the sense that it makes a connection between existing things, the result of creativity, to obtain something completely new. By placing innovation between creativity and code, I am demonstrating what is needed for a common denominator between creativity and the production of lines of code. As a result, by analysing the data from my study, I found five differentiating factors that will be catalysts for creative production in programming: experimentation, individuality, time, diversity and gender. Experimentation is a fundamental element in reducing the gap between theoretical knowledge and practice, having an intrinsic connection with the risk factor and creating environments conducive to innovation. Along with experimentation, we have reflection, transforming tacit knowledge into explicit knowledge. Individuality refers to the importance of having an expert carry out a project instead of using AI or pre-designed templates. It takes us to the human side of programming. The third concept is that of "time" and its nuances, both an element that allows the production of a quality code, as well as being a constraint on the exercise of expertise, in the sense that at certain times there is no availability to accumulate more knowledge in a timely manner. Diversity refers to different academic backgrounds concerning the profession and how this knowledge was acquired. Finally, we have gender imbalance, knowing that having diverse perspectives and experiences helps foster creativity and innovation.

Within CODE, we have at least three places where creativity is revealed: in the code processes and the way it is structured; in the way different people write the same tasks differently; and thirdly, in the programming language, where there are many places to innovate.

When looking at aspects related to the lines of code produced for computer software, I analysed aspects related to the type of relationship that programmers have with the code, the importance of community cooperation and the satisfaction that professionals take from their chosen profession. The way programming professionals see their code affects how they face the present and the future, as their commitment to what they are doing depends on it and how they look at their learning and development. This was evident in the satisfaction one obtained in one's profession and the connection to one own community in terms of sharing and belonging. Nevertheless, above all, in connection with creativity, I can say that the programming code is creative. Within what are the rules of the various programming languages and the concepts inherent to each one, there is ample space for the exercise of creativity.

The creativity associated with the code produces an IMPACT and legacy of the programmers associated with the accomplishments of the co-participants. As a result, we have a personal impact reflecting individual goals, values and desires and, on the other hand, leaving a legacy for the future. This impact also involves reflection on the future; in that sense, it was a pleasant surprise. The use of reflection is rooted in programmers working within the "agile" method of software development.

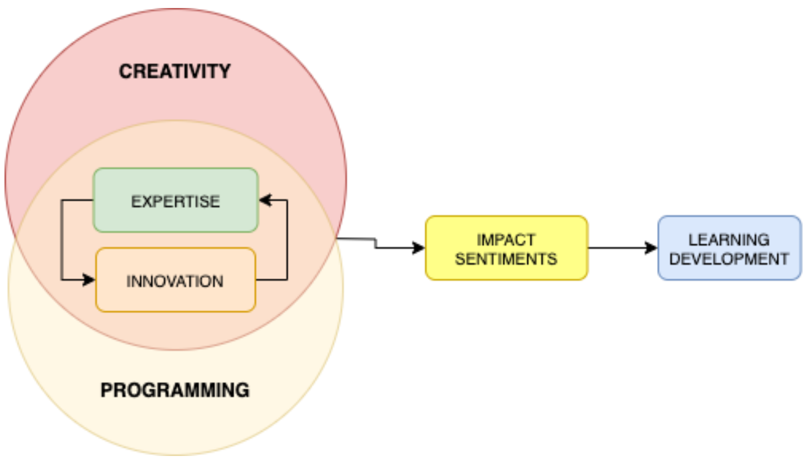


Figure 13: Summary of the themes interconnections

Furthermore, finally, we have the part dedicated to LEARNING & DEVELOPMENT as one looks to the future after scrutinising creativity in programming. Firstly I looked back and saw co-researchers development to become programmers and experts; secondly, I looked at continuous professional development as a way to stay on top. As one of the most interesting parts that came out of the data that was analysed, we have the importance of side projects as a tool for experimentation and self-development. This tool is interconnected in the various themes and sub-themes that were developed and appeals to discovery associated with experimentation and the importance of exploring new things and acquiring new skills outside the workplace.

Part 4: Remarks

14. Findings and Recommendations

In this chapter, I will present my conclusions regarding the data analysis and its implication for practitioners and the community in general, as well as the implications for my practice. This chapter, together with Chapter 5 (Project Activity), constitutes a reflection on this project.

The possibility of doing a Professional Doctorate came at a stage in my career when I was undecided about the steps to take in the future. On the one hand, I already had many years of experience in the area of programming and had been helpful to many teams by acting as a senior consultant. On the other hand, I would have to decide if, instead of continuing to work in the area in a hands-on programming role, I would be willing to manage larger teams and put aside this active programming role, which gives me immense pleasure. Initially, I thought that I would have something to add to the academic community since, although I have already carried out some academic projects, I would still have more to offer to academia. I was already working as a professional at a doctoral level, but I had not yet had the opportunity to demonstrate this knowledge academically. This was the attraction of the professional doctorate. I wanted to go further in my professional career, which is guided by an agile environment where changes are constant and significant. I wanted to open my horizons to new possibilities.

> "Practices are always reflective – people involved in practices 'observe themselves' in the conduct of their practice and can modify their performance as they do so, or on future occasions." (Kemmis, 2010)

14.1. Reflection

The constant observation of my practice led me to transform my way of acting as a practitioner and evolve within my profession. For example, I had several moments when I decided that I had to specialise in a specific area or increase my knowledge in other areas. This comes with deep reflection and close observation not only of what I am doing or producing but also an observation of my community of practice and what other professionals are doing at certain times. Reflection changes who I am in relation to my practice.

Sometimes I felt confused about which way to go. This happens because we cannot see much further ahead when we are embedded in our practice. Schön (1987) describes this as a flat place where we cannot see very far when describing the professional practice.

I already had the notion that whatever I learned as part of this professional project, I would have the opportunity to apply it in my practice, but the opposite also happened. In my work, I constantly investigated complex data and had to present results based on analysis, and I realized that over the time of this doctoral project, this skill of mine was perfected. More and more, I tried to base my professional choices on evidence from the correct and professional analysis of the available information using reflection as a super tool to help with critical thinking and problem-solving. Problem-solving is a fundamental function of a programmer, and all that resilience and ability to see what is hidden and reach plausible conclusions was something that I also brought from my day-to-day work to my academic work. In my programming world, I often encounter setbacks and obstacles that require adaptability and resilience to overcome challenges. My biggest example concerns the "bug fixing" task, in which finding a reproducible error is more complicated than fixing it.

14.2. Professional Impact

Below I present the main findings of this research.

- Programming code is creative. Programming is a craft due to the amount of knowledge and years of experience needed to master it, and it is inherently creative;
- Code is the language used to give instructions to computers, but it needs to be written in such a way that other programmers can understand it;
- Having a side project allows experimentation through the implementation of ideas and observation of results without the pressure that exists in the work environment, and the inherent risks;

The answer to the question of whether there is creativity in the production of lines of code came in the explanation of the art & craft movement, namely (1) the creative process of the work, (2) maintained through the arrangement of the work's stimuli, and which becomes craft when it begins to (3) have a utility, which serves a function when complete. Programming code is creative: 81% of expert co-researchers believe that there is creativity in programming. Looking at other aspects of creativity and code, sometimes there is a perception that there is no creativity in code simply because practitioners are comparing the production of code with other media where creativity is more evident. There is creativity in the process of choosing, experimenting and fitting together the different pieces of code as if they were building blocks. When we look at the production of lines of code by programmers, the code can be presented in different patterns, classes, separation of concerns, and a range of different concepts that reveal the programmer's creativity. There is creativity in the different variety of uses and expressions of programming languages because there is a possibility of choice. A lot of thought and dedication is put into the process of creating code so that we have new libraries and new programming paradigms, and also, there is much room for creativity in the architecture process, including how the code is structured.

The findings suggest that programming code is a creative process and a craft that requires a significant amount of tacit and explicit knowledge to master, as well as some hands-on experience. This has some implications for other practitioners who do not recognise creativity within programming. This means that, whether they recognize it or not, during their careers, they will have to deal with situations where they will have to use their creativity. As a result, we need to bring an appreciation of the art & craft of the code and the effort that goes into producing quality code. This will be achieved through innovation and the five differentiating factors that are the catalysts for the creative production of code in programming: experimentation, individuality, time, diversity and gender, according to the emerging themes of the data. They have been explained in detail above, and the first three aspects relate to programmers in particular in the sense that they explore what programmers can do in order to be more creative in their work. Experimentation is a fundamental element in reducing the gap between theoretical knowledge and practice, having an intrinsic connection with the risk factor and creating environments conducive to innovation. It is essential to create these individual and collective spaces where there can be free reign and freedom to create and innovate with regard to computer code. Along with experimentation, we have reflection, transforming tacit knowledge into explicit knowledge. Individuality refers to the importance of having an expert carry out a project instead of using AI or pre-designed templates. It takes us to the human side of programming. The third concept is that of "time" and its nuances, both an element that allows the production of a quality code, as well as being a constraint on the exercise of expertise, in the sense that at certain times there is no availability to increase the knowledge capacity in a timely manner. Diversity refers to different academic backgrounds concerning the profession and how this knowledge was acquired. Finally, we have gender imbalance, knowing that having diverse perspectives and experiences helps foster creativity and innovation.

For the community at large, the findings suggest that programming should also be recognised as a creative area and that creativity should be encouraged in software development. This implies that creative thinking and other areas related to creativity should also be taught alongside computer science. These new perspectives on the software development process include the importance of experimentation, exploration, collaboration and innovation.

When I think about areas that could be improved in the school curriculum regarding the teaching of computer science and programming, some ideas emerge based on my experience and the participants' experience. At the top of my list to improve the curriculum of programming students so that they become more creative than the average comes "Art & Design", with its principles of aesthetics and user experience that could foster creativity and increase the visual appeal of programming. Next, we have "Creative Writing and Storytelling", which is so important when I stated that the code must not be written only to be comprehended by the computer but also to be understood by other programmers. An ability to write creative stories can help a programmer write and improve narratives and consequently write and justify code that a more significant number of people can understand and engage with. Among other topics that could improve the creativity of programmers, I would like to mention "Game Design", which combines art, storytelling, sound design and programming. This study area is essential for improving creativity because being a multidisciplinary field requires creativity at all levels, from conceptualization to the final product. By studying principles associated with game design, we would inspire innovative thinking for a series of problems that programmers face daily.

Other findings of this research:

- The facilitation of an environment focused on creativity allows experts to optimise their impact within organisations;
- Some limitations are not harmful to creativity;
- Some limitations imposed on programmers undermine their creative capacity, as is the case of rigid structures, code limitations and self-imposed limitations;
- Computer software engineers are averse to taking risks, and this acts as a limitation to creative potential;
- The perception of the relationship with the created code affects how programmers use their creativity to develop their code and, ultimately, their entire career;
- Half of the co-researchers think of themselves as experts. Half, do not;
- Acquiring the skill is not enough to be considered an expert. Deliberate exposure to various forms of acquiring knowledge is required resulting in different patterns gained through experience that can be used in similar situations in the future;

- Not listening to what programmers have to say (their voice) is a risk one should consider in business;
- Taking ownership is a characteristic of experts that involves taking responsibility for your choices but also looking for solutions amongst others;
- Knowing different programming languages could open doors;
- The best ideas come from collaborating with others;
- The biggest hurdles when programmers have a leadership position are leading people and being promoted against their will;
- Programmers become better developers after starting to manage other people;
- Programmers-managers should manage the willingness to control others and impose one's will;
- Each person thinks differently, and it is necessary to accept that each one will come up with a solution to problems differently;
- Programmers with different backgrounds approach problems in a different way than those who have had formal education or training in programming.
- Programmer-managers are those who take expertise to another level: they are catalysts for change for the better, or they become an embarrassment to other programmers;
- Experimentation is one process that makes it possible to bridge the gap between theory and practice.
- Experimentation as one of the pillars of innovation in terms of code.
- Experimentation outside the workplace allows for the exploration of limitations that one may encounter at work and explore some gaps in technical knowledge;
- Experimentation leads to the development of new skills;
- Side projects are a way for programmers to explore their interests, experiment with new technologies, and acquire new skills that will benefit their personal and professional development;
- Programmers need time to produce quality code. Programmers do not have enough time to properly develop their ideas on the code;
- Having people with different backgrounds in academic terms and origin increases the creative capacity of the projects;

- The accomplishment was what the participants in this study were looking for when they decided to embark on a career in IT;
- There is a link between problem-solving and creativity. It takes creative thinking to find the solution to a software problem;

14.3. Personal impact

Regarding my practice, the findings suggest an approach to code as a creative process focused on exploration, experimentation and innovation. This would also imply starting side projects, looking for new programming paradigms, and developing new libraries to pursue personal development and push my knowledge to the limit. In additional terms, I have to emphasise the craft of the coding by writing code that is functional yet structured and easy to understand.

We are all aware of the difficulty and time it takes to complete an academic work of this size. There were many tasks, some carried out in parallel and others in cascade (professional jargon), and I could not consistently reconcile the time necessary to carry out all of them. If doing a doctorate studying full-time is tricky, I add a few degrees of uncertainty and complexity when it is done part-time. Countless times "paid" work took precedence over academic work, sometimes for whole seasons. Here I can also see an intersection between my daily work and academic research, especially concerning self-discipline, time management and long-term project management.

The doctorate also put extra pressure on my body and mind among the many challenges I faced. It is a work of persistence, dedication and much reflection, and sometimes the body feels this pressure and signals that something is incorrect. It happened to me in the middle of the project when I was diagnosed with skin cancer, and it was not pleasant. I had surgery, spent some time in recovery, and returned to my usual self. These things are part of our humanity, and it was necessary to look with clarity and determination not to abandon the unfinished project. Furthermore, it was good that I did not.

Carrying out this project led me to think more deeply about language and the mother tongue and its influence on how we act and think. I always thought knowing another language was essential to communicating with people from other cultures. This led me to study English and improve my Spanish, adding other cultures to my original Portuguese language culture. During this project, I discovered the writings of Father António Vieira, a Portuguese Jesuit Priest who had the task of evangelizing the natives of Brazil in the seventeenth century. In one of his many sermons, he referred to the communication difficulties encountered as "we are all mute, and all of them are deaf" (Rocha, 2019). He made an effort to learn a common language so they could understand him. Moreover, there are stories that only make sense in the original language. All this made me feel more comfortable with my different personalities in the different languages I speak, and, in a way, it contributed to a linguistic improvement in general, both in terms of writing and speaking.

As I am about to finish this project, I have the notion that I can think and write differently. My thoughts, although sometimes less organized, manage to nurture my creativity in a way that I produce innovative thinking and new solutions for my workplace. For example, I have a "Treaty for Pull Requests" project that will revolutionize the way feedback is given to programmers concerning their produced lines of Code, among other ideas. I recently implemented a self-development project at work on Friday afternoons that included the development of hackathons[12] in some weeks and individual projects in others. The main objective was that together we could solve some programming issues that we were facing in the company in a practical, collaborative and quick way. By working together within a defined timeframe, we wanted to develop creative thinking and innovation so that new ideas could emerge in this environment, which is very different from the more structured environment of our daily work. New tools and technologies emerged from this project, providing a hands-on learning experience crucial to staying on top of the technological changes we face in our careers. The other aspect was encouraging the development of side projects within working hours to allow the exploration of areas of interest and deepen knowledge and concepts that met their own self-development needs. The idea was to develop projects that address community problems that facilitate the development of programming skills for problem-solving in creative and innovative ways.

14.4. Follow-up

This is the beginning of my journey, and I have several ideas that could be developed in the follow-up of this research.

Quantitative Study: My research involved collecting qualitative data through interviews. Conducting a quantitative study to complement the qualitative results would be interesting. Through surveys or experiments, I could try to measure and quantify the impact of creativity on the production of lines of Code, on the degree of satisfaction as a programmer and even on team performance.

[12] "A hackathon is an intensely focused working event, typically a technology company, health care institution or university hosts for programmers and other stakeholders to address a particular challenge or project. The participants work together to meet project parameters quickly and efficiently." Rand, B. (2022). *What Is a Hackathon? Definition, Benefits and Tips.* [online] Indeed Career Development. Available at: https://www.indeed.com/career-advice/career-development/what-is-hackathon [Accessed 31 Mar. 2024].

Longitudinal Studies: My study was cross-sectional and gave an idea of programmers' experiences at a specific point in time and in their careers. It would be interesting to longitudinally study programmers over a more extended period to examine changes in programming practices and behaviour concerning creativity over time. As a result, we could see the evolution of creativity over time and its effect on performance.

Diversification of samples: My focus was on expert programmers, but the study could be extended to programmers at different levels of experience, from novices to expert professionals, giving a broader sense of creativity in the various stages of their careers.

Comparative studies: One could explore comparative studies between creativity within different programming languages in a more specific way to discover language-specific or platform-specific insights.

Cultural and Contextual Factors: I could investigate cultural and organizational factors about creativity in programming, which could reveal unique approaches to fostering creativity, and understanding these influences could lead to a greater appreciation of creativity in diverse settings.

Creativity Training: I could design and implement interventions or training programs to increase creativity among programmers to evaluate the effectiveness of these programs in increasing the programming outcomes of innovative practices.

15. References

Collins English Dictionary. (2011) 11th edn. Glasgow: HarperCollins.

Adamson, G. (2007) *Thinking through craft.* London: Bloomsbury Visual Arts.

Agência Lusa (2019) *Nações Unidas estimam em mais de 2,6 milhões os portugueses emigrados.* Available at: https://observador.pt/2019/11/05/nacoes-unidas-estimam-em-mais-de-26-milhoes-os-portugueses-emigrados/ (Accessed: Mar 19, 2022).

Allmark, P., Boote, J., Chambers, E., Clarke, A., McDonnell, A., Thompson, A. and Tod, A.M. (2009) 'Ethical Issues in the Use of In-Depth Interviews: Literature Review and Discussion', *Research Ethics,* 5(2), pp. 48-54 Available at: 10.1177/174701610900500203.

Augsburg, T. (2014) 'Becoming Transdisciplinary: The Emergence of the Transdisciplinary Individual', *World futures,* 70(3-4), pp. 233-247 Available at: 10.1080/02604027.2014.934639.

Autry, J. (2004) *The Servant Leader: How to Build a Creative Team, Develop Great Morale, and Improve Bottom-Line Performance.* New York: Three River Press.

Axbey, H., Beckmann, N., Fletcher-Watson, S., Tullo, A. and Crompton, C.J. (2023) 'Innovation through neurodiversity: Diversity is beneficial', *Autism: the international journal of research and practice,* Available at: 10.1177/13623613231158685.

Bazeley, P. (2013) *Qualitative data analysis.* Los Angeles et al.: Sage.

BCS - The Chartered Institute for IT (2021a) *BCS diversity report 2021: Age.* Available at: https://www.bcs.org/policy-and-influence/diversity-and-inclusion/bcs-diversity-report-2021-age/ (Accessed: Aug 30, 2022).

BCS - The Chartered Institute for IT (2021b) *BCS diversity report 2021: Women in IT.* Available at: https://www.bcs.org/policy-and-influence/diversity-and-inclusion/bcs-diversity-report-2021-women-in-it/ (Accessed: May 4, 2022).

Beaird, J. and George, J. (2014) *The Principles of Beautiful Web Design.* 3rd edn. Collingwood: SitePoint.

Beck, K., Beedle, M., Bennekum, A.v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. *Principles behind the Agile Manifesto.* Available at: https://agilemanifesto.org/principles.html (Accessed: Jun 27, 2022).

Berners-Lee, T. and Fischetti, M. (2000) *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web.* New York: HarperCollins.

Bhattacharya, A. (2016) 'The many ways of knowing: Embracing multiplicity in narrative research', *Qualitative social work : QSW : research and practice,* 15(5-6), pp. 705-714 Available at: 10.1177/1473325016652683.

Bolton, G. and Delderfield, R. (2018) *Reflective Practice: Writing and Professional Development.* London: SAGE Publications.

Braun, V. and Clarke, V. (2022) *Thematic analysis: a practical guide.* London: Sage Publications.

Braun, V. and Clarke, V. (2013) *Successful qualitative research: a practical guide for beginners.* Los Angeles: SAGE.

Brine, R. (2019) 'The Basics of Evidence-Based Practice', *People & Strategy,* 42(1), pp. 1-7.

Briner, R. (2019) *The Basics of Evidence-Based Practice.* Available at: https://www.shrm.org/executive/resources/people-strategy-journal/winter2019/pages/ebp-briner.aspx (Accessed: June 9, 2023).

Cain, S. (2012) 'When Collaboration Kills Creativity '*Quiet - The Power of Introverts in a World That Can't Stop Talking* London: Penguin, pp. 71-94.

Campbell, D. (2019) *The many human errors that brought down the Boeing 737 MAX.* Available at: https://www.theverge.com/2019/5/2/18518176/boeing-737-max-crash-problems-human-error-mcas-faa (Accessed: April 30, 2023).

Case#01 (2019), *Interview Transcript* (unpublished).

Case#02 (2019), *Interview Transcript* (unpublished).

Case#03 (2019), *Interview Transcript* (unpublished).

Case#04 (2019), *Interview Transcript* (unpublished).

Case#05 (2020), *Interview Transcript* (unpublished).

Case#06 (2019), *Interview Transcript* (unpublished).

Case#07 (2019), *Interview transcript* (unpublished).

Case#08 (2020), *Interview Transcript* (unpublished).

Case#09 (2020), *Interview Transcript* (unpublished).

Case#10 (2020), *Interview Transcript* (unpublished).

Case#11 (2020), *Interview Transcript* (unpublished).

Case#12 (2020), *Interview Transcript* (unpublished).

Case#13 (2019), *Interview Transcript* (unpublished).

Case#14 (2019), *Interview Transcript* (unpublished).

Case#15 (2020), *Interview Transcript* (unpublished).

Case#16 (2020), *Interview Transcript* (unpublished).

Catmull, E. (2014) *Creativity, Inc.: Overcoming the Unseen Forces that Stand in the Way of True Inspiration.* London: Transworld Digital.

Chapman, N. and Chapman, J. (2006) *Web Design - A Complete Introduction.* Rep. 2011 edn. Chichester: Wiley.

Christensen, C.M. (2016) *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Boston, MA: Harvard Business Review Press.

Cloudfare (no date) *Why minify JavaScript code?* Available at: https://www.cloudflare.com/en-gb/learning/performance/why-minify-javascript-code/ (Accessed: Dec 9, 2022).

Code Project (no date) *Code Project - for those who code* . Available at: https://www.codeproject.com/ (Accessed: Oct 19, 2022).

Cohn, M. (2010) *Succeeding with Agile: Software Development using scrum.* Boston, MA: Pearson Education.

Collins, H. (2010) *Tacit and Explicit Knowledge.* Chicago and London: The University of Chicago Press.

Corbin, J. and Strauss, A. (1998) *Basics of Qualitative Research.* 2nd edn. Thousand Oaks, California: SAGE Publications, Inc.

Costley, C., Elliott, G.C. and Gibbs, P. (2010) *Doing Work Based Research: Approaches to Enquiry for Insider-Researchers.* 1st edn. London: SAGE Publications.

Crawford, A. (2002) 'W. A. S. Benson, Machinery, and the Arts and Crafts Movement in Britain', *The Journal of decorative and propaganda arts,* 24, pp. 95-117.

Cropley, D. (2015) *Creativity in Engineering.* Burlington: Elsevier.

Csikszentmihalyi, M. (1996) *Creativity: The Psychology of Discovery and Invention.* New York et al.: HarperCollins.

Dall'Alba, G. and Sandberg, J. (2006) 'Unveiling Professional Development: A Critical Review of Stage Models', *Review of Educational Research,* 76(3), pp. 383-412 Available at: 10.3102/00346543076003383.

Dane, E. (2010) 'Reconsidering the Trade-off Between Expertise and Flexibility: a Cognitive Entrenchment Perspective', *The Academy of Management Review,* 35(4), pp. 579-603 Available at: 10.5465/amr.35.4.zok579.

De Bono, E. (2000) *Six thinking hats.* Rev. and updated edn. London: Penguin Books.

Desouza, K.C. and Awazu, Y. (2005) 'Managing radical software engineers', *ACM SIGSOFT Software Engineering Notes,* 30(4), pp. 1 Available at: 10.1145/1082983.1083110.

Dingsøyr, T., Nerur, S., Balijepally, V. and Moe, N.B. (2012) 'A decade of agile methodologies: Towards explaining agile software development', *The Journal of systems and software,* 85(6), pp. 1213-1221 Available at: 10.1016/j.jss.2012.02.033.

Doyle, N. (2020) 'Neurodiversity at work: a biopsychosocial model and the impact on working adults', *British Medical Bulletin,* 135(1), pp. 108-125.

Dreyfus, H.L. and Dreyfus, S.E. (2005) 'Peripheral Vision Expertise in Real World Contexts', *Organization Studies,* 26(5), pp. 779-792 Available at: 10.1177/0170840605053102.

Edmondson, A.C. and Nembhard, I.M. (2009) 'Product Development and Learning in Project Teams: The Challenges Are the Benefits', *Product Innovation Management,* 26(2), pp. 123-138.

Epstein, D. (2019) *Range: How Generalists Triumph in a Specialized World.* London: Macmillan.

Ericsson, K.A., Krampe, R.T. and Tesch-Römer, C. (1993) 'The Role of Deliberate Practice in the Acquisition of Expert Performance', *Psychological review,* 100(3), pp. 363-406 Available at: 10.1037/0033-295X.100.3.363.

Ernst, C. and Chrobot-Mason, D. (2010) *Boundary Spanning Leadership: Six Practices for Solving Problems, Driving Innovation, and Transforming Organizations.* New York et al.: McGraw Hill.

Euchner, J. (2017) 'Creating a Culture of Innovation', *Research Technology Management,* 60(6), pp. 10-11 Available at: 10.1080/08956308.2017.1373043.

Euchner, J. (2016) 'Building a Culture of Innovation', *Research Technology Management,* 59(6), pp. 10-11 Available at: 10.1080/08956308.2016.1232131.

Gibbs, P. and Beavis, A. (2020) *Contemporary Thinking on Transdisciplinary Knowledge: What Those Who Know, Know.* Switzerland: Springer Nature Switzerland AG.

Gigerenzer, G. (2007) *Gut feelings: The Intelligence of the Unconscious.* London et al.: Penguin Books.

GitHub (no date) *GitHub: Let's build from here* . Available at: https://github.com/ (Accessed: Oct 19, 2022).

Gladwell, M. (2008) *Outliers: The Story of Success.* 1st ed. edn. London: Penguin.

Glass, R.L. (2006) *Software creativity 2.0.* Revised and expanded definitive edition edn. Atlanta, Georgia: Developer. Books.

Goldman, A.I. (2018) 'Expertise', *Topoi,* 37(1), pp. 3-10 Available at: 10.1007/s11245-016-9410-3.

Google (no date a) *Google.* Available at: https://www.google.com/ (Accessed: Oct 19, 2022).

Google (no date b) *Google Docs: Online Document Editor — Google Workspace.* Available at: https://www.google.co.uk/docs/about/ (Accessed: Dec 30, 2021).

Govindarajan, V. (2010) *Innovation is Not Creativity.* Available at: https://hbr.org/2010/08/innovation-is-not-creativity.html (Accessed: Dec 9, 2020).

Govindarajan, V. and Trimble, C. (2010) 'Innovation Initiative - The Idea is Just the Beginning', *Leadership Excellence,* 27(5), pp. 15-15.

Gratton, L. (2007) *Hot spots : why some teams, workplaces, and organizations buzz with energy-- and others don't.* 1st ed. edn. San Francisco, CA: Berrett-Koehler Publishers.

Greenleaf, R.K. (2002) *Servant Leadership: A Journey into the Nature of Legitimate Power and Greatness.* New York: Paulist Press.

Guest, G., MacQueen, K.M. and Namey, E.E. (2012) *Applied thematic analysis.* Los Angeles: Sage.

Hamel, G. and Breen, B. (2007) *Theœ future of management.* 1. print. edn. Boston, MA: Harvard Business School Press.

Haverbeke, M. (2022) Email to Lisias Loback, 27 November.

Haverbeke, M. (2014) *Eloquent JavaScript : A Modern Introduction to Programming.* 2nd edn. San Francisco, CA: No Starch Press.

Hurst, M. (2023) *ChatGPT's dangers are starting to show.* Available at: https://creativegood.com/blog/23/chatgpt-dangers.html (Accessed: March 17, 2023).

IBM (no date) *What is Cloud Storage?* Available at: https://www.ibm.com/topics/cloud-storage (Accessed: Dec 30, 2021).

Jarrett, C. (2015) *Great myths of the brain.* Chichester: Wiley.

Jarrett, C. (2012) *Why the Left-Brain Right-Brain Myth Will Probably Never Die.* Available at: https://www.psychologytoday.com/blog/brain-myths/201206/why-the-left-brain-right-brain-myth-will-probably-never-die (Accessed: Jun 12, 2021).

Johnson, S.C. (1978) *Lint, a C Program Checker.* Murray Hill, New Jersey: Bell Laboratories. Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.1841&rep=rep1&type=pdf (Accessed: Oct 5, 2022).

Jolley, J. (2020) *Introducing Research and Evidence-Based Practice for Nursing and Healthcare Professionals.* Milton: Taylor and Francis.

Jolly, J. (2020) *Boeing 'gambled with public safety' in run-up to two deadly crashes.* Available at: (Accessed: Apr 21, 2023).

Jscrambler (2022) *JavaScript Obfuscation: The Definitive Guide.* Available at: https://blog.jscrambler.com/javascript-obfuscation-the-definitive-guide (Accessed: Dec 9, 2022).

Kahneman, D. (2011) *Thinking, fast and slow.* London: Penguin.

Kelley, T. (2016) *The Art Of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm.* London: Profile Books.

Kelley, T. and Kelley, D. (2015) *Creative confidence.* Paperback edn. London: William Collins.

Kemmis, S. (2010) 'What Is Professional Practice? Recognising and Respecting Diversity in Understandings of Practice', in Kanes, C. (ed.) *Elaborating Professionalism: Studies in Practice and Theory* The Netherlands: Springer Netherlands, pp. 139-165.

King, N. and Horrocks, C. (2010) *Interviews in Qualitative Research.* London: Sage.

Kitroeff, N., Gelles, D. and Nicas, J. (2019) *Boeing 737 MAX safety system was vetoed, Engineer Says.* Available at: (Accessed: Apr 20, 2023).

Klein, G. (2015) 'A naturalistic decision making perspective on studying intuitive decision making', *Journal of applied research in memory and cognition,* 4(3), pp. 164-168 Available at: 10.1016/j.jarmac.2015.07.001.

Klein, G. (2013) *Seeing what others don't: the remarkable ways we gain insights.* 1st edn. New York: PublicAffairs.

Loback, A. (2021), *Five Keys for Being Better Creatives* (unpublished).

Loback, L. (2022) Email to Marijn Haverbeke, 15 September.

Loback, L. (2019), *Field note after an interview* (unpublished).

Loback, L. (2018), *DPS 4561 Planning a Practitioner Research Programme* (unpublished).

Loback, L. (2017), *Excerpt of a handwritten note about Scalography* (unpublished).

Love, P.G. and Guthrie, V.L. (1999) 'Kegan's Orders of Consciousness' *Understanding and Applying Cognitive Development Theory* San Francisco: Jossey-Bass.

Lynch, J. (2017) *The Worst Computer Bugs in History: The Ariane 5 Disaster.* Available at: https://www.bugsnag.com/blog/bug-day-ariane-5-disaster (Accessed: 21 December 2022).

MacDonald, M. (2021) *Ageism in Tech: Real Barrier or Overhyped Myth?* Available at: https://medium.com/young-coder/ageism-in-tech-fatal-barrier-or-outdated-myth-fdf872130f1c (Accessed: May 9, 2021).

Macnamara, B.N., Hambrick, D.Z. and Oswald, F.L. (2014) 'Deliberate Practice and Performance in Music, Games, Sports, Education, and Professions: A Meta-Analysis', *Psychological science,* 25(8), pp. 1608-1618 Available at: 10.1177/0956797614535810.

Mahler, C., Gutmann, T., Karstens, S. and Joos, S. (2014) 'Terminology for interprofessional collaboration: definition and current practice', *GMS Zeitschrift für Medizinische Ausbildung,* 31(4), pp. Doc40 Available at: 10.3205/zma000932.

Marcotte, E. (ed.) (2011) *Responsive Web Design.* New York: A Book Apart; Series number, 4.

Meyerson, B. (2016) 'Embedding Innovation in Corporate DNA', *Research Technology Management,* 59(6), pp. 30-35 Available at: 10.1080/08956308.2016.1241657.

Microsoft (no date) *Microsoft OneNote Digital Note Taking App — Microsoft 365.* Available at: https://www.microsoft.com/en-gb/microsoft-365/onenote/digital-note-taking-app (Accessed: Dec 30, 2021).

Miles, M.B., Huberman, A.M. and Saldaña, J. (2014) *Qualitative data analysis - a methods sourcebook.* 3rd edn. Los Angeles et al.: Sage.

Miller, P. and Wedell-Wedellsborg, T. (2013) *Innovation as Usual: How To Help Your People Bring Great Ideas To Life.* Boston, MA: Harvard Business School Publishing.

Montuori, A. (2012) *Transdisciplinary Reflections.* Available at: http://integralleadershipreview.com/7072-transdisciplinary-reflections/ (Accessed: May 13, 2023).

Montuori, A. (2008) 'The Joy of Inquiry', *Journal of transformative education,* 6(1), pp. 8-26 Available at: 10.1177/1541344608317041.

Montuori, A. and Purser, R.E. (1995) 'Deconstructing the Lone Genius Myth: Toward a Contextual View of Creativity', *The Journal of humanistic psychology,* 35(3), pp. 69-112 Available at: 10.1177/00221678950353005.

NCH Software. (no date) *Express Scribe Transcription Software for Mac* [0]. Available at: https://www.nch.com.au/scribe/index.html (Downloaded: Jul 20, 2019).

Nicolini, D. (2011) 'Practice as the Site of Knowing: Insights from the Field of Telemedicine', *Organization Science,* 22(3), pp. 602-620 Available at: 10.1287/orsc.1100.0556.

Nielsen, J. (2000) *Designing Web Usability: The Practice of Simplicity.* Indianapolis, Ind: New Riders.

Nielsen, J. and Loranger, H. (2006) *Prioritizing Web usability.* Berkeley: New Riders.

Nishimuta, M. and Layng, T.V.J. (2021) 'On the Distinction between the Abstract Tacts Art and Craft: A Concept Analysis', *The Psychological record,* 71(4), pp. 585-594 Available at: 10.1007/s40732-021-00491-w.

Nowotny, H., Scott, P. and Gibbons, M. (2003) 'INTRODUCTION: 'Mode 2' Revisited: The New Production of Knowledge', *Minerva (London),* 41(3), pp. 179-194 Available at: 10.1023/A:1025505528250.

Nussbaum, B. (2013) *Creative Intelligence: Harnessing the Power to Create, Connect, and Inspire.* New York, NY: HarperCollins.

O'Hara, S. (2018) 'Autoethnography: The Science of Writing Your Lived Experience', *HERD,* 11(4), pp. 14-17 Available at: 10.1177/1937586718801425.

O'Grady, P. (2012) 'Achievement vs Accomplishment: The accomplished student is a successful student', *Psychology Today,* Nov 11,. Available at: https://www.psychologytoday.com/us/blog/positive-psychology-in-the-classroom/201211/achievement-vs-accomplishment (Accessed: Jan 28, 2023).

O'Leary, T. (2015) *Helios project - Renault-Nissan Alliance.* Available at: https://www.behance.net/gallery/24801489/Helios (Accessed: Sept 11, 2021).

Opdal, P.M. (2001) 'Curiosity, Wonder and Education seen as Perspective Development', *Studies in philosophy and education,* 20(4), pp. 331-344 Available at: 10.1023/A:1011851211125.

Otter.ai (no date) *Voice Meeting Notes & Real-time Transcription* . Available at: https://otter.ai/ (Accessed: Nov 26, 2019).

Parker, I. (2005) *Qualitative psychology: Introducing Radical Research.* Maidenhead: Open University Press.

Pikkarainen, M., Codenie, W., Boucart, N. and Heredia Alvaro, J.A. (2011) *The Art of Software Innovation : Eight Practice Areas to Inspire your Business.* Berlin, Heidelberg: Springer.

Pikul, D. (2023) *Neurodiversity and creativity - the (almost) magic link.* Available at: https://www.linkedin.com/pulse/neurodiversity-creativity-almost-magic-link-dominika-pikul/ (Accessed: May 1, 2023).

Polanyi, M. (2009) *The tacit dimension.* Chicago and London: The University of Chicago Press.

Prensky, M. (2015) 'Achievement vs. Accomplishment: An important distinction in education', *Educational Technology,* . Available at: http://marcprensky.com/wp-content/uploads/2013/04/Prensky-Achievement-vs-Accomplishment-FINAL.pdf (Accessed: Jan 28, 2023).

QRS International. (no date) *Nvivo* [0]. Available at: https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/about/nvivo (Downloaded: Jan 31, 2019).

Raelin, J.A. (2008) *Work-based learning : bridging knowledge and action in the workplace.* 2nd edn. ProQuest Ebook Central: John Wiley & Sons, Incorporated.

Ridley, D. (2012) *The Literature Review.* 2nd edn. London: SAGE Publications.

Rocha, T.C. (2019) 'A persuasão pelo caráter do orador em Antônio Vieira', *Letras,* , pp. 153-168 Available at: 10.5902/2176148537956.

Salmons, J. (2014) *Qualitative online interviews : strategies, design, and skills.* Los Angeles: SAGE.

Sanderson, K. (2023) *GPT-4 is here: what scientists think.* Available at: https://www.nature.com/articles/d41586-023-00816-5 (Accessed: March 17, 2023).

Saunders, M., Lewis, P. and Thornhill, A. (2009) *Research methods for business students.* 5th edn. Harlow: Financial Times Prentice Hall.

Savignano, L.K. (1996) *Taskmaker Color.* Available at: https://web.archive.org/web/20141117082823/http://www.allgame.com/game.php?id=11915&tab=review (Accessed: Apr 20, 2022).

Savignano, L.K. (1989) *TaskMaker.* Available at: https://web.archive.org/web/20141115114208/http://www.allgame.com/game.php?id=664 (Accessed: Apr 20, 2022).

Sawyer, R.K. (2013) *Zig Zag: The Surprising Path to Greater Creativity.* First edn. San Francisco, CA: Josses-Bass / Wiley.

Sawyer, R.K. (2012) *Explaining Creativity: The Science of Human Innovation.* Second edn. Oxford: Oxford University Press.

Schön, D.A. (1994) *The Reflective Practitioner: How Professionals Think in Action.* Abingdon, Oxon: Taylor & Francis Group.

Schön, D.A. (1987) *Educating the Reflective Practitioner. Toward a New Design for Teaching and Learning in the Professions.* San Francisco: Jossey-Bass Publishers.

Seidman, I. (2013) *Interviewing as qualitative research: a guide for researchers in education and the social sciences.* 4th edn. New York: Teachers College Press.

Selinger, C. (2004) 'The creative engineer: what can you do to spark new ideas?', (8, 41), 47-49. 10.1109/MSPEC.2004.1318183.

Smith, J.A. and Osborn, M. (2008) 'Interpretative Phenomenological Analysis ', in Smith, J.A. (ed.) *Qualitative Research - Practical Guide to Research Methods*. Second edn. SAGE Publications, pp. 53-80.

Smith, S. and Paquette, S. (2010) 'Creativity, chaos and knowledge management', *Business Information Review,* 27(2), pp. 118-123 Available at: 10.1177/0266382110366956.

Snyder, H. (2019) 'Literature review as a research methodology: An overview and guidelines', *Journal of Business Research,* 104, pp. 333-339 Available at: 10.1016/j.jbusres.2019.07.039.

Stack Overflow (no date) *Stack Overflow - Where Developers Learn, Share, & Build Careers* . Available at: https://stackoverflow.com/ (Accessed: Oct 19, 2022).

Sternberg, R.J. (1999) *Handbook of Creativity.* Cambridge: Cambridge Univ. Press.

Stokel-Walker, C. (2023) *ChatGPT can find and fix bugs in computer code.* Available at: https://www.newscientist.com/article/2356482-chatgpt-can-find-and-fix-bugs-in-computer-code/ (Accessed: March 17, 2023).

Sullivan, G. (2009) 'Making Space - the Purpose and Place of Practice-led Research', in Smith, H. and Dean T., R. (eds.) *Practice-led Research - Research-led Practice in the Creative Arts* Edinburgh: Edinburgh University Press, pp. 41-65.

The Cynefin Co (2023) *The Cynefin Framework.* Available at: https://thecynefin.co/about-us/about-cynefin-framework/ (Accessed: 17 March 2023).

Thompson, N. (2017) *Theorizing practice: a guide for the people professions.* 2nd edn. London: Palgrave.

Travis, G. (2019) *Ship the airplane: The cultural, organizational and technical reasons why Boeing cannot recover.* Available at: https://spectrum.ieee.org/how-the-boeing-737-max-disaster-looks-to-a-software-developer (Accessed: 21 December 2022).

Van Essen, A. (2014) 'Circling Stories: Cree Discourse and Narrative Ways of Knowing', *Writing on the edge,* 25(1), pp. 44-55.

W3Schools (no date) *W3Schools Free Online Web Tutorials* . Available at: https://www.w3schools.com/ (Accessed: Oct 19, 2022).

Wallace, D.B. and Gruber, H.E. (1989) *Creative people at work: twelve cognitive case studies.* Oxford and New York: Oxford University Press.

Walliman, N. (2004) *Your Research Project*. reprint. edn. London: Sage.

Willig, C. (2008) *Introducing Qualitative Research in Psychology.* 2nd edn. Maidenhead: Open University Press.

Wisker, G. (2007) *The postgraduate research handbook: Succeed with your MA, MPhil, EdD and PhD.* 2nd edn. Basingstoke: Palgrave Macmillan.

16. Appendices

16.1. Questionnaire

Questioning and guiding

1/ What made you choose to be a developer?
2/ Tell me about the process of you becoming a professional developer?
3/ How did you acquire the skills you have?

4/ Do you believe you are an expert?
5/ How do you make decisions about what you do?

6/ What is most liberating to you in terms of coding?
7/ What is more frustrating?
8/ Define your relationship with the code?

9/ How do you solve complex programming problems?
10/ How do you feel about taking risks on your code?

11/ How important is to have rules and constraints in code?
12/ What are the rules that you can break?

13/ Can you tell me about limitations imposed to programmers?
14/ How do you challenge the limitations?

15/ How creative is the coding process?
16/ Do you feel supported in your decisions in terms of code?
17/ How do you experiment with new ideas in terms of code?

18/ How does your perspective of code changed when you started managing other people?

19/ Does anything gets in the way of you exercising your expertise?

20/ Out of last jobs where did you feel more creative / liberated / inspired? Why?
21/ Where did you feel you couldn't contribute?

22/ How do you deal with complexity on the code? Libraries?
23/ How do you cope with the constant changes in programming paradigms / libraries?

24/ What is the competitive advantage that you bring to the profession?
25/ If you leave tomorrow, in 1 or 2 years what will people remember you for?
26/ What are your big achievements as a developer?
27/ If I ask your team what they find more inspiring about you, what the answer will be?

28/ what other things that we didn't cover and you want to tell me?
29/ Is anything else that you think is important and I have not covered?
30/ Female interviews

16.2. Invitation Email

From: ll646@live.mdx.ac.uk
To: participant@email.com
Title: Doctoral Research Invitation for participants from Lisias Loback

Hi Mxxxx,

It's been a while since I've spoken to you, but I wanted to keep you informed of what I've been doing lately.

I embarked on a Doctorate project at Middlesex University - London, where I am studying the link between creativity and programming code within a context of code and system complexity, and limitation on the creative ability of programmers. This will involve me interviewing a range of professionals that has demonstrable capabilities on this field, like you, in order to get a deep insight into the subject.

I want to invite you to participate in this project because you are an expert programmer with several years of accumulated experience, including as a manager, whose knowledge, history and opinion can add value to this research.

To this, I add the fact of knowing you personally.

It would only take about one hour to one hour and a half of your time to be interviewed anonymously, between March and July 2019, at a location, day and time of your best convenience.

I would love to have you on board.

Attached to this email, I am sending a "Participant Information Sheet" where you have more information about the project objectives, procedures, confidentiality and consent.

Please, reply with your response.
Is this your phone number? - 07XXX XXXXXX

Yours sincerely,

Lisias Loback MBCS MSc
Professional Doctoral Candidate
Middlesex University London

Tel: +44 (0) 203 411 8641
Mob: +44 (0) 792 226 7590

Email: lisias@loback.com
Web: www.lisiasloback.com

16.3. Participant Information Sheet

Middlesex University London

Participant Information Sheet

Doctoral Research for Lisias Loback

Thank you for agreeing to take part in this research project for my Doctorate at Middlesex University. You have been selected because you are an expert programmer with several years of accumulated experience, whose knowledge, history and opinion can add value to this research project. You will have to give written consent to participate, and you will be able to withdraw at any time, without the need to give any justification whatsoever. If you wish to withdraw before or during your participation, the information already collected will not be used in this study. Thank you for taking part and contributing to the advancement of scientific knowledge.

Research Objectives

This project aims to investigate the relationship between creativity and the production of innovative computer code. It intends to look at the limitations placed on the experts and to perceive this dynamic of negotiation and connectivity between the various agents in the organisation. By analysing these relationships, new knowledge will be created. This knowledge will be shared with the various agents of the class community and outsiders. It will produce new ways of working, new ways of learning in the workplace, and a path to self-learning and development. It will change the way organisations see programmers, creating new opportunities for experts and programmers with the characteristics listed.

Procedures

Your participation will be through an interview conducted by the researcher, which can last between one hour and one hour and a half. There is an intention to record the interview and your consent is required for the recording and subsequent processing of the information. Be as sincere as possible. If you do not want to answer a question, do not feel pressured to do so.

Information Processing

The treatment of your data will be anonymous. Some questions are about your professional data such as years of experience, academic background and preferences regarding work, programming languages and details about your coding process. All references to companies, people, co-workers, managers, will be anonymised.

Confidentiality

Participation is confidential, but because the project states that the participants are people I worked with, it may be the case that co-workers, who witnessed the interaction referenced in the final paper, can identify the participant.

Consent form

I am willing to participate in this research study. I understand that I can withdraw at any time, before or during my participation. I give consent for the recording of data and later processing. I may be identified by co-workers in the final project.

Name: ______________________________ Sign: _________________________ Date: __ / __ / ____

16.4. Privacy Notice for Participants

Middlesex University Privacy Notice for Research Participants

The General Data Protection Regulation (GDPR) protects the rights of individuals by setting out certain rules as to what organisation can and cannot do with information about people. A key element to this is the principle to process individuals' data lawfully and fairly. This means we need to provide information on how we process personal data.

The University takes its obligation under the GDPR very seriously and will always ensure personal data is collected, handled, stored and shared in a secure manner. The University's Data Protection Policy can be accessed here: https://www.mdx.ac.uk/__data/assets/pdf_file/0023/471326/Data-Protection-Policy-GPS4-v2.4.pdf.

The following statements will outline what personal data we collect, how we use it and who we share it with. It will also provide guidance on your individual rights and how to make a complaint to the Information Commissioner's Officer (ICO), the regulator for data protection in the UK.

Why are we collecting your personal data?
As a university we undertake research as part of our function and in our capacity as a teaching and research institution to advance education and learning. The specific purpose for data collection on this occasion is to investigate the relationship between creativity and the production of innovative computer code, making room for "radical engineers" with different ways of thinking and challenging the status quo. This project aims to explore the gap in the research in computer programming area linking creativity and innovation to the lines of code produced, with the aim of publishing literature in this area during or after the execution of this project.

The legal basis for processing your personal data under GDPR on this occasion is Article 6(1a) consent of the data subject.

Transferring data outside Europe

In the majority of instances your data will be processed by Middlesex University researchers only or in collaboration with researchers at other UK or European institutions so will stay inside the EU and be protected by the requirements of the GDPR.

In any instances in which your data might be used as part of a collaboration with researchers based outside the EU all the necessary safeguards that are required under the GDPR for transferring data outside of the EU will be put in place. You will be informed if this is relevant for the specific study you are a participant of.

Your rights under data protection

Under the GDPR and the DPA you have the following rights:

- to obtain access to, and copies of, the personal data that we hold about you;
- to require that we cease processing your personal data if the processing is causing you damage or distress;
- to require us to correct the personal data we hold about you if it is incorrect;
- to require us to erase your personal data;
- to require us to restrict our data processing activities;
- to receive from us the personal data we hold about you which you have provided to us, in a reasonable format specified by you, including for the purpose of you transmitting that personal data to another data controller;
- to object, on grounds relating to your particular situation, to any of our particular processing activities where you feel this has a disproportionate impact on your rights.

Where Personal Information is processed as part of a research project, the extent to which these rights apply varies under the GDPR and the DPA. In particular, your rights to access, change, or move your information may be limited, as we need to manage your information in specific ways in order for the research to be reliable and accurate. If you withdraw from the study, we may not be able to remove the information that we have already obtained. To safeguard your rights, we will use the minimum personally-identifiable information possible. The Participant Information Sheet will detail up to what point in the study data can be withdrawn.

If you submit a data protection rights request to the University, you will be informed of the decision within one month. If it is considered necessary to refuse to comply with any of your data protection rights, you also have the right to complain about our decision to the UK supervisory authority for data protection, the Information Commissioner's Office.

None of the above precludes your right to withdraw consent from participating in the research study at any time.

Collecting and using personal data

The treatment of your data will be anonymous. Some questions are about your professional data such as years of experience, academic background and preferences regarding work, programming languages and details about your coding process. All references to companies, people, co-workers, managers, will be anonymised.

Data sharing

Your information will usually be shared within the research team conducting the project you are participating in, mainly so that they can identify you as a participant and contact you about the research project.

Responsible members of the University may also be given access to personal data used in a research project for monitoring purposes and/or to carry out an audit of the study to ensure that the research is complying with applicable regulations. Individuals from regulatory authorities (people who check that we are carrying out the study correctly) may require access to your records. All of these people have a duty to keep your information, as a research participant, strictly confidential.

If we are working with other organisations and information is shared about you, we will inform you in the Participant Information Sheet. Information shared will be on a 'need to know' basis relative to achieving the research project's objectives, and with all appropriate safeguards in place to ensure the security of your information.

Storage and security

The University takes a robust approach to protecting the information it holds with dedicated storage areas for research data with controlled access.

Alongside these technical measures there are comprehensive and effective policies and processes in place to ensure that users and administrators of University information are aware of their obligations and responsibilities for the data they have access to. By default, people are only granted access to the information they require to perform their duties. Training is provided to new staff joining the University and existing staff have training and expert advice available if needed.

Retention

Under the GDPR and DPA personal data collected for research purposes can be kept indefinitely, providing there is no impact to you outside the parameters of the study you have consented to take part in.

Having stated the above, the length of time for which we keep your data will depend on a number of factors including the importance of the data, the funding requirements, the nature of the study, and the requirements of the publisher. Details will be given in the information sheet for each project.

Contact us

The Principal Investigator leading this research is Lisias Loback

Tel: +44 (0) 203 411 8641
Mob: +44 (0) 792 226 7590

Email: lisias@loback.com
Web: www.lisiasloback.com

The University's official contact details are:
Data Protection Officer
Middlesex University
The Burroughs
London
NW4 4BT
Tel: +44 (0)20 8411 5555
Email: dpaofficer@mdx.ac.uk

16.5. Codebook

mynvivo.nvpx Codebook

Codes\\Data Analysis

Name	Description	Files	References
= Code	Top aggregator for the theme Code	17	658
code	sentiments towards coding and programming	15	181
attached to the code	This is a positive answer to the question "What is your relationship with the code?"	6	12
code as a tool	a means to an end	5	11
code community - cooperation	details about the importance of the community and the cooperation, specially at the beginning of the career	12	33
code for fun	enjoyment in coding; type of projects; longevity	12	24
code is not creative	answer to the question about creativity and code	5	10
code perception	The perception that people have about the code and the programmers	7	14
confused	love / hate relationship	3	4
detached from the code	Relationship with the code - detached	10	16
lego	view of code development as a bunch of reusable pieces that the developer fits together	8	15
reusability	the importance of resisability when developing over other ideas	4	9
structured	OCD; order and structure as the main driver	10	18
supported	feel supported in terms of code decisions; the importance of feeling supported	9	14
complexity	complexity theory; the amount of complexity in the code;	14	55
amount of knowledge	amount of knowledge that the developers must know	4	8
refactoring	reusability and refactoring as a way to deal with complexity	8	16
speed of change	This could be a good or a bad thing; reference to a fast pace environment	5	6
decisions	Answer to the question "how do you make decisions in terms of coding?"	17	146
ask other people - counselling	ask other fellow developers	8	14
break down to smaller parts	this is a well known principle in terms of programming	14	28
business value	business acumen; put the business first in terms of coding decisions	9	26
deep thought	deep analytical thought; reflecting;	12	17
gather information	the user of research methods	11	17
planning	spend extra time on planning to assess the possibilities	5	5

Name	Description	Files	References
try different approaches	experimentation over other methods	7	12
understand the code	a mixture between research and planning	9	13
Visualization - Imagination	graphical visual representation of the problems with scribbles and doodling	8	12
fascination about the first lines of code	the way coders got attracted to the profession; self-reallization;	7	9
frameworks and libraries	aggregate information regarding the frameworks and libraries	17	149
add complexity	libraries add complexity to an already complex codebase	3	3
change and evolve	constant change and different evolutions	14	22
fit for purpose	there is always a reason for a library and you should only use it in the way intended	9	15
limitation	libraries are a limitation to code	13	32
saves time	all in for libraries; main reason being saving time	10	16
structure	libraries are a good way to structure code;	7	13
time consuming to learn and keep up	lack of time to learn; frustration with libraries; major issue in relation to libraries;	15	37
Risks	Taking risks; aggregator for the theme / sub-theme	16	97
caution	caution approach to taking risks in programming;	10	18
risk taker	all-in in terms of risk taking;	7	12
risks = bad	situations where taking risk is a bad thing	1	1
risks = good	situations where taking risk is a good thing	4	7
risks and creativity	situations where there is a link between risks and creativity	8	14
risks at the research phase	risk framing only in the research phase of development; not on the actual code delivered;	5	9
time constraints	excuse for taking risks	11	20
way to learn	risks as a way to learn	3	4
Rules	aggregator of ideas about rules in programming and risk taking	12	21
agreement	rules must be agreed rather imposed;	7	11
best practices and creativity	relation between the “best practices” and creativity;	4	5
consistency in a team	the use of rules for consistency on the teams;	13	29
constraints for newcomers	the use of rules as a constraint for new joiners; follow the rules;	6	10
lack of experimentation	result of adding too many rules: lack of experimentation;	4	6
rules not to break	what rules can an expert developer CANNOT break? What is the rationale?	5	7
rules to break	What are the rules that an expert developer can	11	22

Name	Description	Files	References
	break? What is the rationale for that?		
trade-off balance	the rationale for breaking / not breaking the rules; find the balance;	10	20
= Creativity	Top aggregator for the Theme Creativity	17	335
= Innovation	Aggregator for the theme Innovation	15	45
= Diversity	Diversity in relation to innovation and creativity	11	31
female developers	the role of female developers in the context of diversity	5	5
low self-steem	self-esteem as mentioned in the programming context	4	9
Transdisciplinary	contribution from transdisciplinary to the diversity in the innovation and creative process	5	6
women	aggregator for mentioning female developers	6	11
advantage	advantage being a women in the context of diversity	2	2
defiance	opportunity to prove themselves	1	1
fair treatment	women need a fair treatment in relation to their counterparts	4	4
fit in	accommodate and fit women in	2	2
no difference	aim for a no difference in treatment	5	6
Innovation	main code for innovation; no aggregation from child nodes;	8	14
Innovation as a choice	developed this idea early on; capture the opinion of the interviewees;	3	3
arts & crafts	computer programming as a craft;	6	8
Creativity	This is a general placeholder for creativity;	17	187
code is creative	affirmative answer about the concept of coding being creative	13	29
code is not creative	negative answer about the concept of coding being creative;	3	12
Create space for creativity	conversations about the need to create a space for creative / innovation to flourish;	9	19
creative side	information about people recognising that they have a creative side in relation to programming or else;	13	33
experimentation	the usage of experimentation as a creative way of coding.	10	22
lack of creativity	lack of creativity in coding	5	10
Limitations to Creativity	explains the limitations to creativity in coding;	10	21
need time to be creative	reference of lack of time as the main constraint to the creative process	3	5
limitations - constraints	List the limitations and constraints to the creative process in programming;	16	95

Name	Description	Files	References
challenge limitations	situations where the limitations were challenged;	7	12
hardware	hardware as the main constraint for creativity	4	5
limitations foster creativity	limitations as a way to further foster the creativity	8	13
management	management as a constraint to creativity;	10	16
own limitations	self-limitations	7	9
working with others	working with others as a constrainst	8	15
= Expertise	Top aggregator for the Theme Expertise	17	268
Expertise	Questions about expertise	17	178
= Developer Beliefs	Opinions in relation to their own epxertise	16	34
don't believe they are an expert	Do you believe you are an expert?	10	19
pointing towards being an expert	Do you believe you are an expert?	8	15
experts take the leadership		7	12
exposure to different programming languages	exposure to different programming languages helps in the pursuit of expertise	11	28
exposure to other people	Collaboration as as a way to innovate	11	17
Intuition	The use of intuition in the context of expertise	13	28
lack of time	to be experts you need time to work out and practice things;	6	13
perfectionism	experts as masters of their craft	6	9
wear different hats	different things done at the same time	6	7
Expertise - Drawbacks	things that hinder the use of expert knowledge;	9	21
limitations to expertise	limitations to expertise in the context of programming;	4	5
not being heard	not being heard as a major factor of not exercising the expertise;	8	15
manager	natural evolution from being an expert to be a manager; aggregate;	14	69
become a better developer	become a better developer as a result of being a manager;	4	5
change in terms of code	changes after being appointed a manager;	4	5
dictator - big ego	negative trait when become a manager;	6	8
interested in people	people factor that lead to management;	8	16
mentoring	help other people	5	8
no change in terms of code	reference to no change in the way they code after becoming a manager;	2	2
organizational skills	management	5	5
troubles transitioning to manager	experience when transitioning to manager;	8	14
= Impact - Sentiments	Top aggregator for the Theme Impact. The impact	16	217

Name	Description	Files	References
	they have on the profession. Abstracts sentiments towards the profession		
achievements	perception of their career as programmers;	14	28
code impact	impact in terms of code	6	7
just did my job	minimal impact	3	4
national impact	nationwide impact	1	2
no impact	no impact as developer whatsoever; no change;	1	1
personal lasting impact	the impact was just personal; personal improvement;	7	8
teaching	help others	1	1
worldwide impact	huge impact; worldwide;	2	2
balance	Challenged but not overwhelmed; balanced impact;	6	8
frustrations	aggregator for the list of frustrations faced by programmers;	16	128
bad code or language	bad code or language	10	20
bad system	bad system	10	15
blockages	main frustration: blockages;	6	6
copy paste code	the way that the development / programming is going forward;	2	3
Dogmatic	Dogmatic;	1	5
management	management as a source of frustration;	10	18
monotonous - boring	coding is boring;	9	15
new things and no knowledge	new things coming out all the time, and the lack of knowledge about them;	4	6
no coding	Move on to other responsibilities and are coding less	6	10
people	frustration with people in general;	2	2
stuck on permanent job	no career progression prospects and stuck on a permanent job;	5	5
time	no time; lack of time;	7	13
wasted time	wast time on useless projects; have projects scrapped half-way through;	4	4
future	How to see your future and the new generation	9	17
inspiring	inspiring impact	3	4
positivity	keep positive and move forward	2	2
most liberating	list of the things that the programmers love to do more of it;	5	6
create something for myself	own projects	11	23
create something new	novelty	9	19
freedom	ability to decide the path to follow	16	60

Name	Description	Files	References
good communication	no communication issues	2	4
lab rat	Peace and quiet	8	13
love the code	love /hate relationship with the code;	6	13
mix of personal interest and work	balanced approach to work; personal and work based;	2	3
recognition	narcissistic way of working;	7	13
variety of tasks	to prevent boredom	1	1
purpose and meaning	something bigger than themselves;	2	6
think ahead of the time	Forward thinking in terns of career; visionaries;	11	20
= Learning & Development	Cluster for the way developers use to self-learn and develop their skills	17	298
Continuous Professional Development	Continuous Professional Development	11	22
formal education	importance of formal education	10	21
grab opportunities	When the opportunity to be a developer arrived, they were in; importance of grabbing the opportunities;	5	6
learn by doing	process of learning; practice;	14	27
Learning	learning as a stimulus	4	6
other related skills	skills other than computer skills, like communication	5	7
practical skills from university	liked practical skills from university	5	6
Reason for becoming a developer	Broader answer about the question “What made you choose to be a developer” or “How did you become a developer?”; list of reasons;	16	115
acomplishment	fulfilled	15	32
Become a developer by chance	Become a developer by chance	5	7
coded before university	Got contact with code before going to university, as a kid or teenager	4	5
Control the Outcome	control freak; control de outcome;	6	8
design background	developers with a design background that helped shape the future career;	3	6
finantial reward	went after the money; reason to become a developer;	4	7
influence	looking for influence; power;	4	4
interest in computers	already had prior interest in computers hence becoming programmers;	7	8
interest in design	prior interest in design, hence becoming a programmer;	6	9
solve problems	problem-solvers; get energised by solving problems;	11	27

Name	Description	Files	References
videogames	videogames was the reason to get into programming;	2	2
self-learning	process of learning: self-taught;	16	50
side project for learning	importance of having a side project for learning and development;	11	36
work for free to get experience	highlight the beginning of the career;	2	2