Master's thesis

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

Henning Bang Halvorsen

Refining Commercial Open Source: Driving Adaption and Growing Ecosystems

A Case Study of MongoDB

Master's thesis in Informatics
Supervisor: Eric Monteiro
June 2020

NTNU
Norwegian University of Science and Technology

Henning Bang Halvorsen

Refining Commercial Open Source: Driving Adaption and Growing Ecosystems

A Case Study of MongoDB

Master's thesis in Informatics
Supervisor: Eric Monteiro
June 2020

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

NTNU
Norwegian University of
Science and Technology

Abstract

Open source is a widely acknowledged phenomenon in the software development industry and community because of its highly innovative success stories. However, people still think of open source as something free of charge, is of low quality and can never be as profitable as other proprietary business models. MongoDB is the world's leading NoSQL database company and has seen huge economic success being an open source project. In an attempt to uncover their secret to success, a case study was conducted and the data gathered was based on publicly available documents in email threads and forums. The thesis also includes an interview with the CTO of MongoDB.

The results indicate that MongoDB has utilised the marketing advantages of a platform ecosystem and has used the massive open source community as a fertiliser for growing such an ecosystem. Finally, their recent change of open source licence has ignited a debate around the definition of open source and its applicability on the modern tech scene. The thesis concludes that further research should be conducted on similar projects like MongoDB which in the long run can compile to a very successful business model for open source projects.

Keywords: Open Source, Platform Ecosystem, Open Source Licensing, Business Model

Acknowledgements

Before writing this thesis I had little to no knowledge of open source software. From my experience, it was something that I considered to be "free of charge" just like many others who do not know much about the topic. It has been really interesting to take a deep dive under the hood to learn more about this phenomenon.

This is my first time embarking on writing a thesis completely on my own. It has been hard, easy, fun and really, really frustrating. However, it has been very rewarding and I have learned a lot of valuable lessons along the way; both academically as well as personal.

This concludes my five years at NTNU in Trondheim, and what a ride it has been. I would like to thank all of the people I have met living here, my friends at Studentersamfundet as well as my friends at NTNU which I have had the pleasure to cooperate with.

I would like to thank my family for encouraging me along the way, and sometimes providing some money for a nice dinner on a Saturday.

I would like to thank my girlfriend for all the love and support she has shown me over these years, as well as helping me grow into a better version of my self.

Lastly, I would really like to thank my coordinator, Professor Eric Monteiro, who has guided me through the very overwhelming realm of scientific research. I would like to especially thank him for always staying positive and constructive even though I had very little to show for in my guidance hours.

Contents

1. Introduction **1**
- 1.1. Background . . . 1
- 1.2. Motivation . . . 3
- 1.3. Problem Description . . . 3
- 1.4. Research Questions . . . 4
- 1.5. Thesis Structure . . . 4

I. Literature Study **5**

2. The History of Open Source **6**
- 2.1. Richard Stallman and the Free Software Movement . . . 6
- 2.2. Free Software and Open Source . . . 8
- 2.3. Open Source 2.0 . . . 9
- 2.4. Commercialisation and Business Models . . . 11
- 2.5. Open Source Today . . . 12
- 2.6. Innovation and the Future . . . 13

3. Licences **16**
- 3.1. The Ten Rights of Open Source . . . 16
  - 3.1.1. Free Redistribution . . . 17
  - 3.1.2. Source Code . . . 17
  - 3.1.3. Derived Works . . . 17
  - 3.1.4. Integrity of The Author's Source Code . . . 17
  - 3.1.5. No Discrimination Against Persons or Groups . . . 17
  - 3.1.6. No Discrimination Against Fields of Endeavour . . . 17
  - 3.1.7. Distribution of License . . . 17
  - 3.1.8. License Must Not Be Specific to a Product . . . 18
  - 3.1.9. License Must Not Restrict Other Software . . . 18
  - 3.1.10. License Must Be Technology-Neutral . . . 18
- 3.2. Strong Copyleft . . . 18
- 3.3. Permissive Licences . . . 19
- 3.4. Weak Copyleft . . . 19
- 3.5. Custom Licences . . . 20

4. Software Development Process **21**
- 4.1. Traditional Development . . . 21
  - 4.1.1. Waterfall . . . 21
  - 4.1.2. Agile . . . 22

4.1.3. DevOps . . . 25
4.2. Open Source Development . . . 26
4.2.1. Community as a Team . . . 27
4.2.2. Decentralised . . . 27
4.2.3. Open Collaboration . . . 27
4.2.4. Knowledge Sharing . . . 28
4.3. Project Growth . . . 28
4.3.1. Forks . . . 28
4.3.2. Ecosystem . . . 29
4.4. Project Success . . . 30
4.4.1. Market and Technological Success . . . 30
4.4.2. Extrinsic Cues . . . 31
4.4.3. Intrinsic Cues . . . 31
4.4.4. Correlations between Cues and Success . . . 31

5. Developer Motivation **34**
5.1. Personal "Itch" . . . 34
5.2. Mobilising by Ambiguity . . . 35
5.3. Gifting Culture . . . 36
5.4. Need to Belong . . . 36
5.5. Renown . . . 37
5.6. Economic Incentives . . . 37
5.7. Choice of Licence . . . 38

II. Method **39**

6. Research Strategy and Method **40**
6.1. Method . . . 40
6.2. Literature Review . . . 40
6.3. Data Generation . . . 41
6.3.1. Case Study . . . 41
6.3.2. Documents . . . 42
6.3.3. Interview . . . 42
6.4. Interview Guide . . . 43
6.4.1. Interview Situation . . . 43
6.5. Analysis Method . . . 43
6.6. Quality . . . 44
6.6.1. Reliability . . . 44
6.6.2. Validity . . . 44
6.6.3. Generalisation . . . 44
6.7. Limitations . . . 45

7. The Choice of Case **46**
7.1. Practical Criteria . . . 46
7.1.1. Time alive . . . 46
7.1.2. Activity . . . 46
7.1.3. Success . . . 46
7.1.4. Growth . . . 47
7.1.5. Innovation . . . 47

7.1.6. Start-up . . . 47
7.2. Thematic Criteria . . . 47

III. Case Study: MongoDB 48

8. The History of MongoDB 49
8.1. Traditional Database Systems . . . 49
8.2. The Limits of RDBMS . . . 50
8.3. Why NoSQL? . . . 51
8.4. Original Business Plan . . . 52
8.5. Why Open Source? . . . 53
8.6. Early Stakeholders . . . 54
8.7. Expansion . . . 55
8.8. MongoDB Today . . . 56

9. Organization 59
9.1. The Company . . . 59
9.1.1. Development . . . 59
9.1.2. Products . . . 60
9.1.3. Open Source Licence . . . 60
9.2. The Community . . . 62
9.2.1. Code of Conduct . . . 62
9.2.2. Key Contributors . . . 62
9.2.3. Applications Built by the Community . . . 63
9.3. Forums . . . 63
9.3.1. User Forums . . . 63
9.3.2. Developer Forums . . . 64
9.3.3. MongoDB World . . . 65
9.4. Community Acknowledgements . . . 65
9.4.1. Community Badges . . . 65
9.4.2. Certifications . . . 66
9.4.3. MongoDB University . . . 66
9.4.4. Innovation Award . . . 66

IV. Analysis 67

10. Discussion 68
10.1. MongoDB's "Itch" . . . 69
10.2. Community Engagement . . . 69
10.2.1. Mobilising . . . 70
10.2.2. First Contact . . . 70
10.2.3. Gifting Culture . . . 70
10.2.4. Users over Contributors . . . 71
10.2.5. Virtual Incentives with Value . . . 71
10.3. The MongoDB Ecosystem . . . 71
10.3.1. Focus on Platform . . . 72
10.3.2. Rapid Development . . . 72
10.3.3. Acquiring Community Projects . . . 72

10.4. Business Model 73
10.4.1. Project Success 73
10.4.2. Multiple Value Propositions 73
10.5. Choice of Licence 74
10.5.1. Debate 74
10.6. Refining Commercial Open Source 74

11. Conclusion **76**
11.1. Limitations of the Study 76
11.2. Further Work 77

Appendices

A. Appendix **86**
A.1. Assignment Text 86
A.2. Interview Guide 87

List of Figures

2.1. Free Software Foundation Logo 8
2.2. The Browser Wars 10
2.3. The Open Source Initiative Logo 12

3.1. The Copyleft mark 18
3.2. Table of different open source licence types 20

4.1. The waterfall model as it was first presented by Royce 21
4.2. One sprint in Scrum 24
4.3. SEMAT Quick Reference Guide 25
4.4. The phases of the DevOps cycle 26
4.5. Elements of a platform ecosystem 30
4.6. Factors leading to IT Project Success 32
4.7. Original hypothesises model of cue impacts on success 33

8.1. MongoDB Inc. Logo 49
8.2. 10gen Logo 52
8.3. Milestones from the MongoDB website's about page 54
8.4. Milestones from the MongoDB website's about page 55
8.5. Milestones from the MongoDB website's about page 56
8.6. Milestones from the MongoDB website's about page 56
8.7. Numbers from the MongoDB website's about page 57

1 | Introduction

> *Today, 99% of all new software projects have open source dependencies.* Nat Friedman [1]

These are the words of GitHub CEO, Nat Friedman, at his talk entitled "The state of Open Source" at the GitHub Universe 2019 conference [1]. Over the last four decades, the open source phenomenon has evolved from being a way to protest proprietary software to becoming one of the most commonly used paradigms when developing software today [2] [3]. However, it is still commonly believed that open source development is only for the "genius hackers" sitting in the dark in a basement somewhere, coding twenty-four hours a day [4] [5]. This is a myth that strains back to the 1980s and the roots of open source. Yes, the original ideology was founded by so-called "hackers" but has evolved way beyond its cradle. The value of free and open source software has become more and more apparent both to individual developers as well as large enterprises. SourceForge, a web-based platform for distributing and finding open source business software, has over 35 million monthly users and well over 500,000 software projects. It is hard to believe that such a large phenomenon can fit in small, dark basements.

1.1. Background

As stated, even large enterprises are tilting towards using open source as their chosen way of working compared to traditional methodologies and closed models [6] [7]. They see the value in contributing to the worldwide knowledge base and to receive external input and expertise in return. Many think that open source software is equal to free software, where free in this case refers to "free of charge". Therefore, it is believed that utilising such a business model cannot be commercialised or bring any form of income for the company. This is a common misconception that originates from a concept fairly equal to open source which is in fact called free software, but it refers to "free as in freedom, not free beer". The free software movement was the first to rebel against propitiatory software, and by spreading their ideology, they successfully avoided the formation of monopolies in the software market. Some meant that the name free software was too ambiguous and too often mistaken for free of charge and that the free software movement was too focused on the social aspects behind "sharing". Thus, the open source foundation arose where, although open source implies the same values as free software, the focus shifted from a social aspect to a more practical aspect; open source development as a methodology. The founding father of the free software foundation, Richard Stallman, argues that free software and open source are the same categories of software but that the fundamental values behind the ideologies are different, namely being a social movement and a development methodology respectively [8].

What differentiates the open source way of working from traditional closed projects is

the formation of communities around each project as well as the large umbrella that is the open source community. A community that thrives on helping each other and being open to new ideas from contributors all around the world. This way of working has let projects grow at staggering speeds [9] and evolve from the simplest of ideas to some of the most commonly used software today. It also allows small companies and startups to compete with million dollar companies like Microsoft, Google and Apple by having external stakeholders in addition to the internal ones. Most software on the market today that are proprietary has got an open source alternative. These are often attractive because proprietary software can be very expensive, and open source software is usually free of charge. This is where another misconception arises. How can an open source product generate revenue if it is free of charge?

At its core, open source contribution is voluntary work. Most contributors are not getting paid directly. However, there are still many factors that motivate developers to do this work. Some developers and companies rely heavily on a certain piece of software for their business and want to ensure that the project is kept alive and is still evolving, preferably in the direction of their needs. Other developers might be specialists in a particular field, for example, software security, and use open source projects as targets of ethical hacking to practice their skills and then offer security fixes if they find any breaches. Finally, some companies even pay their developers to work on open source projects for a few hours a week or a month. These might be projects that the company relies on, or some other projects so that their developers gain experience from working on several different projects. So even though open source contributions is regarded as "free labour", there is always something to gain for the developers as well. Even though developers "give away" their contributions, there are some underlying expectations of getting something in return. It varies what this "something" is. Open source development has therefore been compared to gifting culture [10]. Gifting culture involves some kind of mutual understanding that a gift given has to be "returned" in one form or another. These returned gifts can take many forms in open source development, all the way from small merits to employment offers.

Open source has one key advantage that no company will ever have. It allows all the developers in the world to be become potential parts of the project, thereby enabling the quality of the project to be on another level than a proprietary competitor. This is derived from Linus' Law defined by Eric S. Raymond, one of the founders of the Open Source Initiative [11]. Linus' Law states that "given enough eyes, every bug is shallow" and is named after the developer of one of the most famous open source projects today; Linux. Because of open source development, there has been a paradigm shift in the world of technology which enables faster, safer and better development of new technology. Even though it was primarily related to software, the open source model has started to spread into other fields, applying the philosophies of sharing ideas openly. The battery patents of Tesla Inc. are open source which has enabled competitors to keep up in the electric car industry, Googles Android OS dominating in the mobile OS market with around 85% of all devices worldwide running it [12], and high-end universities like MIT have begun to share material and lectures openly to make higher education more accessible [13]. With the ongoing COVID-19 pandemic, tracking contamination has only been possible due to open data sets of peoples location and heath journals [14]. Even Microsoft, perhaps the biggest opposition to free- and open source software, have turned their head around and started to embrace open source software and develop many projects of their own.

> *The world runs on open source software.* - Devon Zuegel, [15]

Thus, developers and companies are starting to see the importance of open source through the innovative impact of larger open source projects. However, open source is still scary and misunderstood by non-developers in the commercial market. The applied business models today are simply not profitable enough for them to be willing to take risks. There are however open source projects who have achieved great commercial success. Some argue that the only reason open source is not the only methodology being used is that the ultimate business model has not yet been discovered. These success projects might hold the key to such a model.

1.2. Motivation

This thesis will try to guide future software projects who want to adopt an open source model and to fully reap its benefits by analysing an existing, long-lived, robust open source project which has frequent users or is a crucial part of several other projects. It will also, most importantly, argue that it can be a very lucrative business model for those with commercial motivations. There already exists a lot of papers and articles written on open source software, but many of them quantitatively approach the subject by only scratching the surface of the characteristics that make a robust open source project. Also, the field of IT as well as the open source community changes rapidly, and some times even drastically, so previously conducted research become outdated quite quickly. It would be interesting to see what previously conducted research is still applicable as well as to see if there are new patterns in developer incentives which deserve more attention. I will try to justify why open source is not just a popular thing, but why it is important as well. This will provide freshly conducted research to the knowledge base and may hopefully serve as a foundation for further research and provide developers and others in tech with insight in what makes make an open source project last as well as profitable.

1.3. Problem Description

The open source model has greatly impacted the software and tech industries through innovative projects [16] [17]. Its effectiveness, quality of outcome and core values are highly acknowledges by developers [18] [19]. To drive the adaption of the model further, large enterprises and other commercially focused companies would need to embrace the model in favour of other proprietary models. However, many of these larger corporations struggle to understand how open source can be as lucrative and profitable of a business model. MongoDB has accomplished great economic success with $250 million revenue in 2019 and is an example of a very profitable open source company [20]. They seem to leverage open source to build a platform instead of a single product. Their ecosystem has grown into a large collection of products and services developed both by the company as well as their community. With this as a background, the problem definition of this thesis is formulated as such.

> *To what extent does the success of MongoDB rely on the application of the open source model, and how may this case contribute to the development of open source as a business model?*

1.4. Research Questions

Derived from my problem description, I have come up with the following research questions which, if answered, will serve most valuable in my opinion.

- RQ1: To what extent does MongoDB follow the definition of an open source project, and how can this have contributed to their growth?
- RQ2: How has the platform ecosystem model provided technical and economic advantages?
- RQ3: How does the community react to eventual deviations in the appliance of the open source model, and what are the potential consequences?

1.5. Thesis Structure

This thesis consists of three major parts. The first part is the literature review where I will present previous studies and background literature which is essential to understand open source and its history.

The first chapter will cover some of the history of open source, it's status today as well as some predictions for the future of open source. One of the main differences from traditional development methodologies is how open source projects are organised. The next chapter will present a comparison with traditional development methodologies and pinpoint the most important characteristics of open source development as a methodology. This chapter will also look at the different business models that are applicable as well as the main strategies for expanding and growing your project. The final chapter in this part will be solely dedicated to developer motivation and how to mobilise a community around an open source project.

The next part presents the research strategies and methods applicable to this thesis. There are several approaches to gathering data for a thesis like this, but I will argue why the methods I chose were feasible. I will also argue why the project *MongoDB* is a suitable candidate for this case study, both from a practical perspective and a thematic perspective.

The third part contains the actual case study with a presentation of both MongoDB Inc. as a business and organisation as well as the database product itself from a more technical perspective. This is where detailed findings that are presumably relevant will be presented as I discover them.

In the final part, I will analyse my findings and explain why they are relevant to the literature study, as well as attempting to answers my research questions. Finally, I will provide some final thoughts on this project, the limitations of my work and suggest further research on the topic based on my results.

Part I.

Literature Study

2 | The History of Open Source

To understand why the open source methodology is so widely used and acknowledged by developers today [18] [19] one need to understand why it arose in the first place, how it has grown into the phenomenon it is today and what the future holds for open source. The following sections will present the history of open source, its role in the tech industry today and its predicted future. This section is primarily historical and will introduce terminology and theory that are further explained in the next chapters of the literature study.

2.1. Richard Stallman and the Free Software Movement

In the late 1970s and the early 1980s, a man by the name of Richard M. Stallman was working in the MIT Artificial Intelligence lab. In those days most new hardware was shipped with a piece of software as well as the source code for that software because the primary business was selling hardware. There was no real market for software as a standalone product. This meant that, if needed or wanted, one could modify the software so that the hardware would suit the specific needs of the users. Stallman and the rest of the lab heavily utilised this opportunity to modify, or *hack*, most of the equipment in the lab, sometimes productively, sometimes just for fun. These groups of developers were named *hacker communities* [8].

As the years went by, more and more companies stopped shipping the source code along with the hardware and software. The software was pre-compiled as binary code that would run on the hardware but was unreadable by humans and thus un-modifiable. Soon, the whole computer farm in the AI Lab was completely replaced with proprietary systems and software. This damaged their hacker community and it was Stallman's first encounter with non-disclosure agreements and the restrictions that followed [21]. Stallman and his colleagues grew more and more hostile towards the whole idea of proprietary software and he eventually acted out his frustration by creating the idea of *Free Software* and started the Free Software Movement that later became The Free Software Foundation. His idea was that software should be completely transparent allowing the consumers to modify the code if they so needed, just like he had been able to just a few years prior. The free software movement did not just apply to software but carried a vision of a free society where sharing achievements and progress were at the centre of driving technological, as well as social, innovation forward [8].

Personal computers were becoming more and more accessible to the public. It was becoming common for developers to have computers at home instead of just at their university or workplace. At the heart of every computer, there is the operating system (OS) which connects software and hardware [22]. The UNIX operating system was the most common operating system at the time used at different universities and labs where developers

would most often access a computer. However, it was a proprietary software system and because of its licence and terms of use, it could not serve as the basis of an ecosystem of free software, because it could not be redistributed by third parties. As an operating system engineer Stallman discovered that he could combat the proprietary software market by creating a new operating system from scratch that he would encourage other people to use and modify, thus spreading his idea of free software. Stallman realised that the UNIX system architecture was composed of a set of many smaller components talked to each other. To create a system that could compete with UNIX, he just had to develop replacements for each of these components one by one. He made public announcements about his mission and encouraged other developers to join him in the development of these programs and by 1991 they had managed to replace almost every single component. Thus, the GNU project was born [23].

> *GNU is a recursive acronym. It stands for "GNU's Not Unix"* - Richard Stallman, [8]

One of the most crucial components of an operating system is the *kernel*. The kernel can be considered the core of an OS because of its responsibility to allocate resources to the other parts of the operating system. The kernel was one of the last components Stallman and his team were working on before they had made a complete, free rewrite of the UNIX OS. Enter Linus Torvalds, a college student from Finland who at the time was also writing his own OS for his home machine, only that he started on the other end with the kernel. He modelled the kernel after the machines he had been using in university so that he could use his home computer the same way. Torvalds was able to finish his kernel before the GNU team, so when the Linux kernel was released [24], people who knew about it started to look for other, free software components that could eventually make up a complete OS. Since the GNU team had been working from the other side of the bridge, these components were already available to them. The Linux kernel filled the gap in the GNU system and the first free, complete OS was created; The GNU/Linux Operating System [25].

An important thing to point out is that free software is not public domain. If it was, some developer could take that piece of software, make minimal modifications (or even no modifications) and then redistribute it as proprietary software denying those who download it to look at the source code, denying modification and further improvement, and put a price tag on it potentially making money on someone else's work. Stallman and the free software foundation, therefore, came up with the concept of *copyleft* [26].

> *It's the idea of copyright flipped over.*

When copyrighting your product, you take legal action to prevent someone else to make money based on your work. It prevents redistribution and sales under other names than yours. Copyright documents can be quite comprehensive and breaking copyrights can lead to large fines or other punishments [27]. Copyleft also protects your rights but those rights are the opposite of copyright. You are encouraged to distribute the product with modifications and improvements, to make money off it and brand it with your or your company's name. Both copyright and copyleft in themselves are just ideas which are manifested as legal documents and they can have many different degrees of restrictions. The most common copyleft based licences applied to free and open source software are more thoroughly presented in chapter 3.

2.2. Free Software and Open Source

One of the earliest contributors to the GNU project was a man named Eric S. Raymond [28]. In 1993, he received a copy of one of the first, commercial copies of the GNU/Linux system and he was completely blown away by the fact that this product in his hands had become reality. At that time he had many years of experience as a software developer and the ways which free software was being developed broke all the rules he knew about developing software. Things like control and complexity by keeping the development team small and tightly connected and having objectives that were tightly overseen were almost completely missing. A product like GNU/Linux should in his, and probably many other's, opinion has failed but it did not. It had become nothing less than phenomenon and Raymond was eager and determined to figure out how this could have happened [8].

A few years later, Raymond published a paper called *The Cathedral and The Bazaar* [5] which were his observations and analysis of how the open source world could work, even though the name "open source" had not emerged yet and it was still called free software. In his paper, he presents two models for developing software and compares the two against each other. One is what he called *The Cathedral Style* is what represent traditional, closed source project development environments. The cathedral style has characteristics like small developer groups arranged in a hierarchical and authoritarian structure, objectives that are very specific and detailed, and the time between version releases are often long. This style could also be applied to open source as the core developer team could release the source code but be very strict regarding accepting contributions from a "lower place" in the hierarchy, in other words from developers outside the core team. In this way, the cathedral style becomes very one-sided [5] [8].

The other style was named *The Bazaar Style* which strives to replicate how projects like Linux is being developed. Characteristics like short release intervals, and many contributors and peer reviews. Raymond argues that having this one characteristic of a very large group of peers getting involved with the project would, in the end, lead to better success than all the characteristics of the cathedral style. In this model, the "power" is equally distributed in the hands of every single contributor. All contributions, as well as discussions, are publicly available on the Internet [5] [8].

He presented his paper at a Linux conference in 1997. As his paper circulated in the community, it eventually reached the ears of the company Netscape. Netscape was in the internet browser market and are the company behind the Netscape Navigator which was the most popular browser before Microsoft decided to launch Internet Explorer. They became the first large company that decided to open source their products in an attempt to battle Microsoft in the *browser wars* [29] in which they desperately needed a secret weapon to overthrow Microsoft's browser monopoly. It became the first real test Raymond's observations on such a large scale. But Raymond saw a problem. What had until now been called free software, a term that was easily misinterpreted as "free of charge" or something that sounds cheap or bad, had a sort of unprofessional association. These

Figure 2.1.: Free Software Foundation Logo

misunderstandings had to be addressed and the message that needed to be conveyed was that

> *The software was open, and the source code was available* - Larry Augustin, then CEO of VA Linux (now Geeknet, inc.) [8]

hence the name *Open Source*. However, the term free software remained as Stallman and the rest of the free software movement were sceptical to the commercialisation of open software and they argue that

> *The freedom to cooperate with other people, freedom to have a community is important for our quality of life. It is important for having a good society that we can live in. That is in my view even more important than having powerful and reliable software* - Richard Stallman, [8]

To clearly define this new term open source, Raymond and a man named Bruce Perens [30] decided to write what would be called *The Open Source Definition* [31] which consists of a set of rights that a piece of software has to have to comply with the definition of open source. These rights are elaborated in on section 3.1. The Open Source Definition would serve as the core for what would later become *The Open Source Initiative* [32], an organisation with the main task to educate the world about open source by explaining and protecting the open source label [33]. They also developed several licences that would be considered *OSI-approved*, in other words, align with the open source philosophy [8].

One of the main reasons that the Linux kernel managed to grow so popular was that several companies decided to focus on developing complete OS distributions of their own built on the Linux kernel and providing support for their distributions. Red Hat, who was the first Linux based company to go public on the stock market, is one of these companies along with previously mentioned VA Linux. By no means was Linux the only software that occupied the open source scene. The following are some examples of open source software who also made huge names for themselves in the same era [8].

- Apache Server - A very flexible and powerful HTTP web server. [34]
- PHP - A popular scripting language suitable for web development. [35]
- GNOME - A Linux based desktop environment, part of the GNU project. [36]

2.3. Open Source 2.0

What began as a movement of hackers doing volunteer work finally got the attention of venture capitalists and larger corporations. Open source was becoming a serious competitor to the proprietary software giants, often represented by Microsoft. In his paper, *The Transformation of Open Source Software*, Brian Fitzgerald suggests that the open source phenomenon has "metamorphosed" from its hacker-inspired origins to something viable for commercialisation and is regarded as a legitimised way of developing software [4]. Fitzgerald argues that the conception that open source is a collective of extremely talented hackers that devote their skills to revolutionise the world is a myth. If this was the case, then open source would not be a global buzzword and would not have grown as much is it has. This stereotype can be criticised based on the amounts of research done on the inside of open source projects by looking through email lists, IRCs and asking developers directly about their incentives to contribute. Exactly what motivation and incentives drive

the developers of open source is still a puzzling topic for many researchers. In chapter 5 I will take a closer look at this topic in detail as it arguably serves as the basis for open source as a methodology [37] [38] [39].

Briefly mentioned earlier, the company Netscape Communications were the pioneers of opening up closed source projects and embracing the open source philosophy. At the time, the Internet was still something mainly used in academia and the military, but browsers like the Netscape Communicator enabled access to the Internet for the common man. There were many browsers like this emerging at the same time, but Netscape was ahead in the competition by "doing everything right" [40]. The Netscape IPO soared on their first day as public, and very quickly caught the attention of Microsoft who would then release the first version of the browser Internet Explorer that would bundle together with their Windows Operating System. At this time, the Netscape Navigator and Internet Explorer were serving the same purpose in a very similar way, so naturally, the consumers did not see why they would install another browser on their Windows machines when there was an identical one already at their disposal. Netscape soon had to face the hard reality of not being able to compete with Microsoft in the browser wars. As a sort of desperate measure, Netscape decided to release the source code for their Communicator suite [40]. The code was available for anyone to pick up and give new life to hopefully spark the next generation of browsers. This is how the Mozilla Project came to be, a project that would use the Netscape source code as a foundation for Netscape's newest browser by harnessing the power of an open source community. By creating this community around their project, the Mozilla project had grown larger than any company. The community were not just interested in creating the next, big browser but were also creating tools, like an email client, as part of a suite of applications that would provide the "best possible browsing experience to the widest set of people" [41]. In 2002, the first version of the Mozilla suite was released but went rather unnoticed as well over 90% of the users of the Internet were using Microsoft's browser. Mozilla kept on fighting and in 2003 they founded the Mozilla Foundation who would continue to manage the development of the Mozilla suite as well as making it their mission to fight for "openness, innovation and opportunity" on the Internet. In 2004, version 1.0 of the Firefox web browser was released and was finally able to compete with Internet Explorer as it was downloaded over 100 million times in the first year of release [41]. This meant one very important thing, it gave the user a choice. There was no longer one king and competition was once again brought back into the realm of internet browsers.

Figure 2.2.: The Browser Wars

What is important to note from Mozilla is their mission to satisfy as many people as possible, in other words, make their software *general purpose*. Since open source is all about distribution and modification, the product should be utilisable as a solution, or at least a foundation in a solution, to many different problems. To fully utilise a huge community of developers, the project has to suit the needs of as many as possible. Their work also expands beyond software like their browser and email clients to creating guidelines for the use of the web and push innovation and technical competence around the globe [42].

> *We're a non-profit organisation working to build a Web that is open, accessible, safe and - most of all - a force for good.* - Mozilla.org, [42]

2.4. Commercialisation and Business Models

Even though Mozilla is a non-profit working through donations and volunteer work, open source does not equal non-profit. In the era of open source 2.0, several business models were developed and others enhanced for a more modern market. With the software being open, anyone can do a deep dive into how it works under the hood. This means that anyone can become a potential expert on a particular software studying all the details of the software. This allows businesses to sell support services around certain software, which enables a competitive support market. With proprietary software, the company that develops the software also claims a monopoly on support services because they are the only ones who know and can know, every detail about the software. If the company does not provide quality support the customers will have bad experiences and there is nothing they can do about it [4].

Aggregating a set of open source tools into a suite and providing support for that particular set was the main business model, to begin with, utilised by companies such as Cygnus Solutions and Red Hat. Another one was to provide two versions of the product. One that was usually completely free of charge, and one that had more features which cost a fee to purchase. These models were called value-added service-enabling and loss-leader/market-creating respectively [4]. With open source 2.0, these models were refined. The value-added service-enabling model has allowed open source products to become infrastructures or ecosystems where smaller companies can become a part of this infrastructure by providing services like support and consultancy for the underlying product. For example, a small company could specialise in providing support for just one of Red Hat's products like their Cloud Suite, where 90% of the support the clients need can be handled by that company and the remaining 10% of more complex problems sent via them to Red Hat directly, thus Red Hat can relocate their resources from support to development [4].

Open source 2.0 has in some sense provided colour to the otherwise black and white stance between proprietary software and free software. It can be argued the GNU Public Licence (see chapter 3) to be in some sense "too free". This has led to the development of several licences that suit particular needs of a company, and as long as they are approved by the Open Source Initiative the project will qualify as open source. This has allowed large companies like Apple, who are very fond of their patents, to open source some part of their technologies to leverage the talents in the open source community to increase their product quality and productivity. As an icing on the cake for Apple, these contributions are mostly free of charge as well. As described by Fitzgerald [4], this becomes a "circular phenomenon" where Apple's reputation in the open source community increases for releasing

intellectual properties, their technology is evolving faster and becoming more attractive, which again will result in more contributions [4].

2.5. Open Source Today

As if it was not enough to revolutionise the world with his Linux kernel, Linus Torvalds did it again with his source code version control system called Git. He developed it as a way to handle all the thousands of incoming contributions to the Linux kernel in a manageable way for one person. Git is now an essential tool in software development in general and has spawned many source code management platforms like GitHub and Bitbucket. These platforms have allowed easy collaboration and distribution of projects in the open source community [43] [44].

> *Today, 99% of all new software projects have open source dependencies.* - Nat Friedman, Current CEO of GitHub [1]

In 2019 alone, over 10 million developers from all around the world contributed through GitHub. No company of any size will ever be able to contest the potential power of millions of developers worldwide. These platforms have become crucial for any project, proprietary as well as open source, and has allowed development to speed up quite drastically through tools for *continuous integration and deployment*, which are further presented in chapter 4. SourceForge which serves as one of the main platforms to distribute open source projects has as of 2020 more than 500,000 different projects, over 35 million monthly users connecting and 4 million downloads per day [45].

With the huge success of Linux that has been previously been described you might be thinking why not every computer is shipped with a Linux based operating system as the default operating system instead of Microsoft's Windows or Apple's macOS. One of the main reasons was that Linux was late to focus on the user experience for people without much technical skills and experience. Some Linux distributions today, like Ubuntu, have realised this and strive to provide a competitive user interface. However, even though Linux did not win the desktop wars, it won practically everywhere else. As of 2019, Linux runs on 96.3% of the world's servers, it runs on 90% on all cloud infrastructures, and it ran on whopping *all* of the world's 500 supercomputers in 2018 [12]. The Android Operation System for mobile [46] dominates the mobile OS market by over 70% worldwide [12]. Although argued that it is not a Linux distribution per definition, it is built on the Linux kernel that they could modify to suit their needs. The killer feature that has allowed the Linux kernel to be embedded almost everywhere is it's diversity or, in other words, it's

Figure 2.3.: The Open Source Initiative Logo

general-purpose [5].

The tendency today is that projects that radiate a certain value proposition and general-purpose will, once grown, become one of two phenomena. One path is the path of Linux where the core software is *forked* into a copy by someone else than the original developers, then modified and improved on until the product is somewhat distinguished from the original. This allows for several different companies and developers to create their version of a given software, in this example a Linux distribution, and distribute their product. The Linux project has many such distributions like Debian and Fedora, which again has sub distributions like the popular Ubuntu as a derivation of Debian, and Red Hat Enterprise Linux as a commercially supported derivation of Fedora supported by Red Hat. Forking is explained further in section 4.3.1. The other path is where the original product is so solid and well maintained by the original developers and that their terms of use are so free that there is no need for a fork. Instead, the community forms an ecosystem of associated apps and tools that makes integration of the core project into any other project much easier. These are tools like drivers for different programming languages, middle-ware interfaces for making the transition between another technology and this one seamless, and tools like graphical user interfaces to an otherwise console-based application. Platform ecosystems are more thoroughly explored in section 4.3.2.

From a business perspective, more and more applicable models for the generation of revenue has appeared in the last decade. Much like Red Hat's model you can sell support and certifications around a certain product. With Software as a Service (SaaS) models, you can take a piece of software that is open source, for example, the popular web platform Wordpress, and use the software and its ecosystem to tailor solutions for customers, thus distributing a product that is created by certain software. If you are developing the software yourself, revenue can come from sponsorships from larger consumers of your product. Larger companies often invest in sponsorships to make sure the project is kept alive and healthy. Exposure is also a consequence of having well-known brands back your project, which may attract more potential contributors [47]. If your product has not gotten the attention of larger potential sponsors yet, platforms like Patreon and Liberapay allows recurrent donations from individual people who back your project. Also on this stage is the newly launched feature GitHub Sponsors which tries to combat the eventual death of open source projects due to lack of funding. GitHub Sponsors offers a unique take where people can choose to fund individual developers directly, regardless of what project they are working on. It turns out that people will pay monthly fees towards projects they believe in and care about as well as developers they admire, and through these platforms, the developers can often provide some incentive to do so [48]. These incentives are more thoroughly explored in chapter 5.

2.6. Innovation and the Future

Today, open source lies at the heart of technological innovation [49] [17] [16]. Open source has evolved from only being applicable in the software world to become more aligned with what Richard Stallman originally envisioned. Some of the greatest achievements of the last decade are results open source and open collaboration. Tesla Motors Inc., one of the largest electric car manufactures in the world, pledged in 2014 to open source all of their patents as a means to advance the sustainable transport technologies of the world faster. At the rate, they were producing and the technology was advancing at the time, there was

no future in sight were electric-powered cars could ever compete with fossil fuel-powered cars. According to Tesla, less than 1% of the sales from major car manufactures were electric before their patent releases [50]. Today, almost every large car manufacturer has an electric vehicle available in the market [51]. Another great achievement is the recent first photo of a black hole. Scientists used the open source programming language Python along with many other open source libraries and plugins in the Python ecosystem to accomplish this great scientific discovery [52].

It has become common to refer to open source outside of the software world as the *open source model* [43] [37] [53], an abstraction of the open source values of open collaboration. The model has been applied to closely related fields like electronics with projects like Arduino, but also in totally different fields like food and beverages with projects like open source Colas and Free Beer who seek to prove the value of the open source model in fields outside technology. The term *open innovation* has also sprung out from the open source values and is promoting. Open innovating is encouraging companies and enterprises to move away from secrecy and the closed environments of traditional research facilities and labs. Even though this term has been researched since the 1960s, it has gotten new life and new meaning in the modern information age we live in today. The idea is that companies will no longer have competitive and innovative advantages by just keeping to themselves in a world where knowledge is so widely distributed. Open innovation encourages to synergise external sources of innovation like customer involvement, academic research and market competitors with the internal intellectual property of your company. This allows the reduction of costs in research and development quite drastically and allows the involvement of the customer early in a product development process, which has proven to be very effective [54] [49].

So where is open source headed, and if it is has so many great characteristics, why does has it not eliminated the proprietary marked? Even though open source is acknowledged by software developers, it is still misunderstood by those with less technical competence. It is still believed to have low security with bugs galore, as well as not as lucrative as a business model as proprietary software with high price tags [43] [39]. But it's ever-growing popularity and portfolio of impactful and innovative projects are sure to persuade even the most conservative companies to adopt it, where applicable, as their model. Open source has eliminated monopoly in the software market as almost every single proprietary product has an open source alternative. If it does not, it is sure to have it within a short amount of time. The main obstacle open source has yet to overcome is to clearly define a general-purpose business model that will work for a large number of projects. Larger projects like Linux does not have an issue with financial sustainability because of its size, renown and its wide applicability. Smaller projects, however, who are usually maintained by a single developer or a small group, struggle more with their funding. As an example, a small company might provide a great piece of software used by many larger corporations, for example in retail or marketing. The software may even be free of charge as with many open source products, but according to the licence used (in most cases), the companies using it are not obligated to "give back" to the project in form of funding. Who should be donating? Who should assure the survival of a crucial piece of software? These are the kind of questions which has remained unanswered for over two decades. As mentioned, more and more ways to fund open source projects are available but giving the customer the right incentives to become a supporter might be difficult. Once a general-purpose, sustainable business model for open source becomes commonly known, few reasons remain

to develop proprietary software. Even so, open source ethics and values will continue to guide society down the path of openness, just like Stallman, Torvalds and all the rest of the open source and free software communities have envisioned [8].

> *It's one, big act of subversive playful cleverness to change society for the better because I'm only interested in changing it for the better.* - Richard Stallman [8]

3 | Licences

Open source completely redefined the way software is being developed and has without a doubt been the catalyst for the for competition in the software market. The history can be quite overwhelming with terminology and different events that sometimes seem completely coincidental. In the following chapters, we will take a deeper dive into the building blocks of open source and hopefully uncover some of the secrets for why it has become the global phenomenon it is today. The very foundation lay in the documented versions of the ideology; the open source licences. It turns out that both developer and businesses carefully evaluate which project to work on or use based on the applied licence [55]. Therefore, it is necessary to take a closer look at the differences between them to understand how each licence can affect a project in the long run. Even though there are many to choose from, these licences can mainly be split into three different categories with some key differences which are presented below.

Normally for proprietary software or other non-software related products for that manner, you how to agree to some sort of licence of use [27]. These licences often present you with your right for refunding the product if damaged, in which situations or places that product can or should be used, and finally, and most importantly for this thesis, that you are not allowed to redistribute the product under another name or as an unlicensed third party. The open source paradigm is almost the complete opposite of this because it encourages redistribution and sharing of the software. The Open Source Initiative has issued several licences that can be applied to open source projects to protect their openly collaboratory nature. Open source companies have also developed their own licences which are run by the Open Source Initiative for approval. All of these licences emphasises the same key points [44] [43].

- The software can be freely used, both commercially and privately.
- The software can be freely modified to suit particular needs.
- The software can be freely distributed to allow others to use, modify and distribute a modification.

According to the Open Source Initiative, every project that declares itself as open source has to follow a set of terms or "rights" that covers how the project should be distributed to its consumers. These terms are called *The Ten Rights of Open Source* [31].

3.1. The Ten Rights of Open Source

The following subsections try to explain each right in a non-legal fashion for better understanding the intentions from a developer perspective. The term *product* will, in this case, refer to the software that has an open source licence. The term *developer* will refer to the

person or people who created the product. The term *consumer* will refer to the person or people who wish to acquire and utilise the product.

3.1.1. Free Redistribution

The right to free redistribution allows the consumer to use the product as part of a larger distribution which the consumer can charge for if they so wish but it is not required to provide the developer with a royalty or fee. In the example of Linux, consumers take the Linux kernel, aggregate a suite of software to go along with it and redistribute it as their product. In the early days of Red Hat, their business model was to sell Red Hat Enterprise Linux as a box set for a certain price [8].

3.1.2. Source Code

The developer has to provide the source code of the product so that the consumer can maintain the software and eventually modify it to suit their particular needs. For example, if the developer has a product that is written to run on Windows, open source code allows the consumer to rewrite the program to work on macOS.

3.1.3. Derived Works

If the product has some issues or features missing that the consumer decides to fix or implement, the consumer is free to redistribute their improved or expanded version of the product. Again, if a product is purely written to work on Windows, one such "derived work" can be a version of the product that works on any of the large operating systems.

3.1.4. Integrity of The Author's Source Code

The honour of the original developer is to be kept. This means that if a consumer redistributes a product that has been improved, it should clearly state what changes have been made. It can also be redistributed as a higher version number than the original product, or in some cases rename the product to completely separate the two versions. The essence is that av the developer created "Product name V1.0", a redistribution of that exact code should be credited to the author.

3.1.5. No Discrimination Against Persons or Groups

The consumer could be anyone of any background, political view, ethnicity, etc. Bruce Perens's example is that an abortion clinic and an anti-abortion activist is both allowed to be consumers.

3.1.6. No Discrimination Against Fields of Endeavour

This right builds upon the previous one stating that the product can be used in any institution or group. Again using Perens's example, the product can be used in business as well as a school.

3.1.7. Distribution of License

The licence which is applied to a product must be distributable in itself. It should be able to stand on its own so that a potential consumer does not need to apply for additional licences.

3.1.8. License Must Not Be Specific to a Product

This means that the licence cannot restrict the product to only being redistributed on a given OS or platform. For example, the product cannot be exclusive to Windows and restrict consumers to improve it to work on other platforms.

3.1.9. License Must Not Restrict Other Software

If the product is distributed on a CD or a website where other products exist that does not apply for the licence, the developers cannot insist that every other product on that shared medium has to be open source.

3.1.10. License Must Be Technology-Neutral

This licence aims to handle a very specific scenario. So-called "clickwraps" or dialogues of an agreement that require the consumer to actively agree (like actively clicking a button) should not be mandatory as this might hinder ways of redistribution via platforms like FTP and web mirroring. The product should also be usable on platforms that do not support dialogue windows that the consumer has actively engaged with.

3.2. Strong Copyleft

The GNU General Public Licence, or GPL for short, is one of the most commonly used licences in the open source world [56]. It is one of the earliest licences and was written by Richard Stallman himself back when they were distributing the GNU system. It was written as a general-purpose licence and consisted of what Stallman meant were the most important parts of all the previous licences he had written for each singular components of the GNU system like the GNU Emacs, the GNU C compiler and the GNU Debugger. It was written as a mirror image of all the characteristics of a copyright licence, hence the term *copyleft* [26]. As previously mentioned, copyleft take is the opposite of copyright where the consumers are encouraged to modify and redistribute software as we have already seen through the rights of open source. When the original version of the GPL was released in 1989 it set out to tackle two main problems which Stallman had with proprietary software.

The most obvious one was to make the source code "open", to provide human-readable code along with the software that was being distributed. Any redistribution, of either the whole software or parts of it, also had to provide all of the source code. Thus the licence assured that no one could take an open source product and make it proprietary. The other problem the GPL addressed was the aggregation of software into larger distributions. If one piece of software under an open source licence and one piece of software under a more restrictive licence where to be distributed together, one of the licences might restrict the

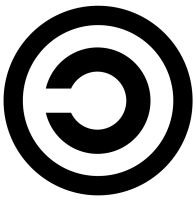

Figure 3.1.: The Copyleft mark

other hence reducing the "freedom" of the software. The GPL, therefore, states that every piece of software that has a component under the GPL must apply the GPL to the whole system. This assured that no more restrictive licence would compromise a software's freedom. This was the first version of the GPL.

The GPL is an example of what is known as *strong copyleft* because it demands its derived works to apply the same licence (or equivalent ones) thus forcing the spread of open source and free software. New software that is written is rarely written completely from scratch, which means developers that want to distribute their software under another licence which is less restrictive cannot do so. From a legal perspective, the GPL can become very complex [26].

3.3. Permissive Licences

On the other side of the spectrum from strong copyleft, we find what is called *permissive licences*. Products carrying this licence still provides all the open source rights, except when it comes to redistribution there is no demand that the modified version must carry the same licence. It can even be made proprietary thus not forcing software aggregations to be open source. Another common label for permissive licences are *non-copyleft* (tilting toward public domain and not towards copyright). The most commonly used permissive licence is the MIT licence. In short, it allows the consumer to do whatever they want with the product as long as the MIT licence is distributed along with derived and modified works. The licence seeks to handle any legal issue by simply stating that the product is provided "as is" meaning that the consumer has no rights of warranty or support for the product if something was not satisfactory to them. It also rids the developer of all ties to the product regarding legal issues. In no scenario will the developer be held accounted for the software's behaviour. The only thing the developer has to provide is their, or their company's, name and the year the product was released [57].

By being this permissive the MIT licence and other licences of similar calibre become compatible with other licences. This means that there is no longer a requirement to "force" the open source ideology on a modified or derived work of the product because it can be redistributed along with software that carries different licences. Although, if the product itself is never modified but simply used as is as a library or plugin in a larger system, that specific component will still carry the MIT licence. This essentially allows the spread of open source software to be halted, and in some cases also reversed, as derived works of a product carrying a permissive licence can be made proprietary [57].

3.4. Weak Copyleft

As a sort of middle ground between permissive licences and the strong copyleft licences, there is a domain called *weak copyleft*. The use case for weak copyleft was primarily for open source software to be distributable alongside proprietary software. Only works derived directly from a product with a weak copyleft licence has to remain copyleft. These licences are mainly used for libraries and plugins linked together with other software. If the library is modified, it is required to keep its licence, or licence equivalent, to remain open source. However, weak copyleft allows for proprietary software to be built. This can also halt the spread of the open source model but not to the same extent as permissive licences as the product that is copylefted cannot be made proprietary [57].

When the GNU team realised the need for some of its tools to be distributed along with side proprietary software, Stallman and the Free Software Foundation created the GNU Lesser General Public Licence. It was originally named Library General Public Licence because it was to be used with every library but Stallman later stated that it is a matter of strategy. If you can licence your library with the normal GPL, its in favour of the free software ideology because it prevents proprietary software from using your library. However, if there are already existing competing libraries that are licensed differently, the LGPL is recommended to appeal to both the open source market and the proprietary market [57].

3.5. Custom Licences

Choosing which licence is right a particular project can be a cumbersome task. If there is no specific license approved by the open source initiative that appeals to you or your company's vision, creating a custom licence might be the best option. This allows the handling of legal edge cases in a particular market or domain. These licences often take inspiration from one another and, most of the time, they can still be categorised as strong or weak copyleft, or permissive. As long as the custom licence gets approved by the Open Source Initiative, the community usually welcomes custom licences with open arms [57]. The following table categorises some of the most commonly used licences with some examples of projects which they are applied to.

Licence Type	**Example Licences**	**Example Projects**
Strong Copyleft	GPL, AGPL	MySQL, Wordpress, Git
Weak Copyleft	LGPL, MPL	Firefox, VLC Media Player, 7-Zip
Permissive	MIT, Apache	Kubernetes, Django, jQuery

Figure 3.2.: Table of different open source licence types

4 | Software Development Process

Through the history of software development, many have tried to devise the perfect development methodology. Which phases are important in a software development process, how can you guarantee that the customer receives what they are paying for, how can you involve the customer or product owner in the process to make sure there are minimal misunderstandings. These are some of the questions that when answered should point towards how software should be built [58]. Understanding which tools are at your disposal as project managers, as well as developers, is crucial to estimate project scopes, development time and potential costs. This chapter aims to highlight how open source projects are developed from a practical perspective to better understand which tools are present and which ones are lacking in the open source developer tool belt.

4.1. Traditional Development

To make sure the contrast between open source development and traditional development is thoroughly reflected, an introduction to traditional development methods is required. Even though some of them are rarely used and others have changed drastically over time, it is useful to look at which components in each methodology were the most restrictive or counterproductive.

4.1.1. Waterfall

One of the earliest ways of developing software is known is the *waterfall model*. It is a very rigid model that is split into several phases of development often displayed in a way that resembles a waterfall (see figure 4.1). It gets its name from the direction of the project phases. Like with a waterfall where the water only runs in one direction, when you go from one phase to the next in the waterfall model, there is no going back to the previous phase, in other words, no way back up the waterfall. It is a linear process where each new phase depends on deliverables from the previous phase [59].

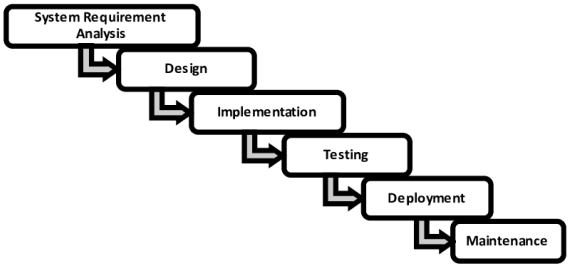

Figure 4.1.: The waterfall model as it was first presented by Royce

Because of its linear nature, the waterfall model is one of the easiest methods to understand both from a developer perspective and the customers perspective. However, it originates from the manufacturing and construction industries where going back and forth between several phases might be impossible. For example, you cannot go back and change the architecture of the foundation of a building of the foundation has already been laid. Applying the waterfall model to software development has been mostly critiqued through the years because going back and making incremental changes are much easier to execute with code than concrete. It was mostly used as a model in the early days of software development because there were no other general-purpose project models that were recognised for creative work like this. Although the waterfall model is mostly used as an example of early development models that are inferior to more modern methods, it is sometimes still used in military projects which require very strict planning. It is applicable where the project requirements very crystal clear, the technologies that are to be used are well known by the developers, and the scope is fixed and will not be changed [59].

Royce [60] presented this model in a paper in 1970 where it was not presented as a recommended model but rather an example of a model does not, in fact, work for software development. The initial model was split into six, sequential phases.

- System and software requirements: The requirements of the system are carefully planned by the project owners and the tech lead of the project.
- Analysis: Understanding the domain of project by creating models and business rules, in other words, the business logic.
- Design: The software architectural phase where the developer's plan which tools to use, which services to develop and how these will communicate, converting the business logic into software terms.
- Coding: Writing the software based on the architecture from the previous phase. Since everything is planned, the implementation should be rapid.
- Testing: Systematically searching through the system for bugs and mistakes. It is in this phase that the lack of work done in the planning phases starts to show. There might even be uncovered misunderstandings between the project owners and the developers.
- Operations: Getting the system up and running on the desired platforms as well as maintaining its uptime and providing support for the users.

Many have later refined his model to try and justify the use of the waterfall model, even Royce himself proposed variants that loosen up the strict sequential patterns. That water still runs the same direction, but it can take different paths down the river.

4.1.2. Agile

As we can see by the waterfall model, the later in the process errors and mistakes were to be uncovered, the more impact they would have and the more time and resources it would take to go back to fix the issue [61]. The cost of fixing errors like this grows exponentially with each phase it slips by. This was the main reason the *agile model* emerged. Developers criticised the restrictions and micro-management of methods like the waterfall model and in the 1990s smaller, more lightweight models began to appear. Seeing this rise of several

methodologies, a group of developers gathered in 2001 to create the *Manifesto for Agile Software Development*. A document that would present principles and values that agile software development should follow. The values are expressed as follows:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

> *While there is value in the items on the right, we value the items on the left more.* - The Agile Manifesto, [62]

In addition to these values, there are also twelve principles which go deeper into defining these values [63].

Agile methods also divide the project into phases, but instead of each phase having a dedicated task being handled, the phases in agile development are simply "timeslots" often referred to *sprints*. In each sprint, every task that was previously dedicated to a whole phase and regarded the whole system is executed, but each sprint focuses on smaller parts of the system. This makes it much easier to uncover obstacles that in models like waterfall would only be uncovered once the coding or testing phase came along. The developer becomes more adaptive to changes along the way which can greatly reduce costs. Agile software development often utilises techniques and tools for continuous integration and continuous unit testing of the already written code. This puts the focus on writing high-quality code that can be easily maintained and later built upon by using the same techniques [61].

Agile as methodology shifted the project focus away from the product to the customer. Using more time and energy understanding the actual customer needs has proved to yield much more satisfied customers [61]. The needs of the customer are rarely completely static as well, they change with time. After each sprint, the developers could verify with the customer that their requirements were being met. If not, they could reiterate and adjust the product more to the liking of the customer. By uncovering misunderstandings frequently during development, you avoid severe costs at the delivery phase due to some minor details interpreted wrong in the planning phase [61].

Scrum

Even though its first appearance was before the release of the agile manifesto, Scrum is still considered an agile methodology and is still one of the most well known and widely used software development methodologies. Agile methods aim to make the developers more independent while still keeping track of the team and projects process. In Scrum, the previously mentioned sprints often have a length of two weeks. Each day, the team performs a *daily scrum*, a daily meeting where they briefly present what they are currently working on and if they have obstacles they need to solve with the help of others. These meetings are a way to keep the project manager updated on what each team member is currently working on. If the teams grow very large, things like these meetings can become quite time-consuming. Scrum, therefore, recommends a team size of a maximum of ten people. At the end of each sprint, the team conducts a *retrospective meeting* where they

look back at what went well and what could be improved to the next sprint [61].

The way work is being distributed among the team members is by the use of what is called a *Kanban board*. This is, in its simplest form, a table consisting of the three columns "to do", "in progress" and "done", though exact wording may vary. In each of these columns are post-it notes containing what is known as "tasks". Tasks are small and very explicit so that ambiguity is at the minimum. Developers can pick a task, read it and (if written correctly) understand exactly what needs to be done. The tasks are derived from larger tasks called *user stories* or sometimes just *stories*. The user stories represent the requirements that have been worked out by the product owner and the *scrum master*, the person in charge of making sure the method is being used correctly [61].

There are a lot more details to the scrum framework that someone with the role as process manager is sure to know and teach the team over time as these rituals and the way of working becomes more of a habit. Due to many of these activities requiring the whole team to engage at the same time, teams that are geographically separated will have a harder time using this model. The whole point of agile development is to work together and communicate as a team as often as possible. Even though modern tools like video conference call systems and digital Kanban boards greatly reduces the geographical barriers, the agile manifesto strongly encourages face-to-face interaction [62]. Another limitation is team members with very specific and specialised skills. Ideally, the whole team should be able to work on every task to some degree. With teams that are limited to a maximum of ten people, it is important to make sure diversity is present. It would be bad if the pile of tasks regarding security were being dealt with very slowly because only one member of the team know how to solve them, while the rest of the team cannot do anything to help [61].

SEMAT

It is believed that the method used to develop a given project should be "tailored" to that specific project. This allows developers to pick and choose from a variety of activities and tools they see fit to utilise in a given project. The SEMAT Essence Kernel is a framework developed to help developers with this *method tailoring*. By visualising a path to the intended goal, developers can pick the components, called *artefacts*, that suit both their needs and their preferences for way of working. For example, your team might agree that a daily scrum meeting is totally redundant. You are then free to choose to leave that one out in favour of other ways to keep the team updated on each other. Maybe simply writing your name on the task you are doing on the Kanban board is enough. Many developers seek away from all the obligatory meetings and activities and want more time to write code [64].

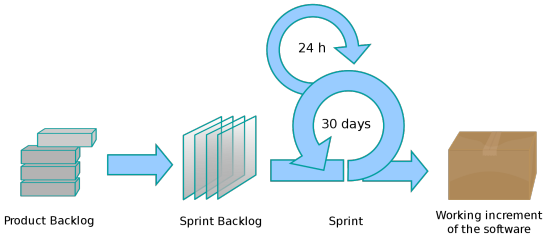

Figure 4.2.: One sprint in Scrum

However, a methodology like Essence Kernel can be viewed as a double-edged sword. The freedom to construct a whole new methodology requires discipline and knowledge of software development processes from the whole team. They have to know what works for them and what they want. Of course, this is not something that you just know, it has to be learned through experience. Besides, tailoring a method for each project means that no one knows how that particular method will work for that particular project. This requires the team to evaluate its constructed methodology along with the actual product along the project development time, which can be quite time-consuming [64].

4.1.3. DevOps

In later years, the term *DevOps* has become a buzzword on many developers lips. The name comes from the aggregation of development and operations. Like with agile development, tools for delivering high-quality code are being heavily utilised. Continuous integration, testing and deployment are at the heart of these processes. While agile development seeks to tie together the customers, or product owner, and the developers, DevOps strive to close the gap between the developers and the IT-departments, in other words, those who build the software and those who maintain it [65].

DevOps leverages all the stages of software deployment, many of which can be highly automated using different tools.

- Coding: Most developers use an integrated developing environment, or IDE, which provide auto-completed code snippets, commands to push and pull code to repositories, review code and merge code.
- Building: On platforms like GitHub, Gitlab and Bitbucket, automated processes can run to check that the code that is being pushed can successfully compile and be built without failure. This hinders code that fails to build to be merged into the repository.
- Testing: Similar to the automated building, automated testing can be done as well where suites of unit tests and integration tests are run each time a new commit is pushed. This makes sure no working code is broken when new code is being pushed.
- Packaging: Bundling large amounts of code dependencies into one package that is ready for the release can be quite a big and complex task. Automated scripts can run these processes with a simple command.

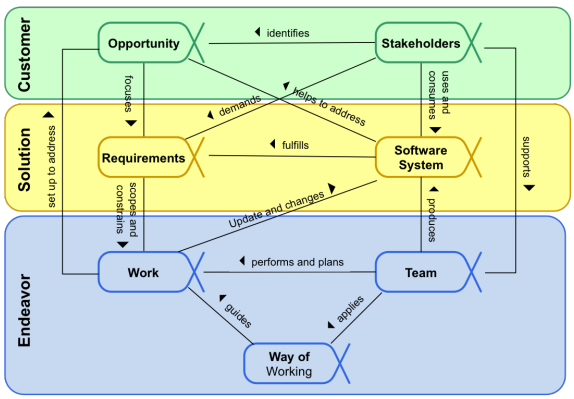

Figure 4.3.: SEMAT Quick Reference Guide

- Releasing: Taking down the newest release from production and releasing the new package. This also sometimes involves sending release packages off to approval in app stores or similar.
- Configuring: Many systems today are moved away from private company servers and into the cloud. Cloud vendors allow allocating the resources needed for the application to run without restrictions and auto-scale if needed.
- Monitoring: Monitoring tools allow administrators to be alerted if errors occur in production, or in the worst case the application stops working. There are also tools to track end-user behaviour for analysing usage patterns.

One way to measure performance in IT is by throughput and stability, in other words how often new releases happen and how error-free these releases are and how quickly eventual errors are fixed [65]. DevOps aims to better performance all across the deployment pipeline.

- Improving the deployment frequency.
- Shorten the time it takes for a product to get from development to the market.
- Lower failure rates when releasing new versions.
- Shorten the time it takes to fix errors that manage to get released, as well as the time it takes to reset the system should everything crash.

Automation has made the maintenance part of software projects much easier to handle for smaller teams, which is why continuous integration and testing are some of the most important tools for open source project maintainers.

There are many different books and papers written on software methodology that go much more in-depth than this section did. It is by no means a complete introduction to Scrum, SEMAT or DevOps but I found it necessary to present the essence of them to draw lines between these traditional methodologies and open source development, which I will do in the following section.

4.2. Open Source Development

Open source development carries some characteristics that make it hard to apply traditional methodologies like the waterfall model and sometimes even the agile methods like Scrum as well. In addition, developers tend to become lazy following certain methodologies when

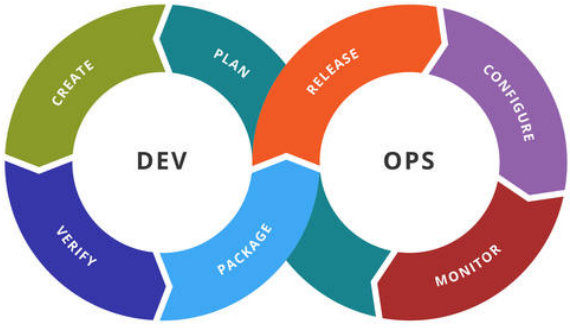

Figure 4.4.: The phases of the DevOps cycle

the framework becomes too restrictive or the exercises feel too mandatory instead of useful. How are open source projects being developed practically speaking? Are methodologies being used at all, or is chaos dominating? First, we need to take a closer look at the characteristics of open source projects from a practical perspective to see if methodologies are even applicable.

4.2.1. Community as a Team

Open source projects usually start with having a single developer or a small team of developers at the core. They might have been working on the project for some time before releasing to the public using a methodology like Scrum or another agile variant which have suited their need. However, one of the most unique characteristics of an open source project is that all the developers in the world could potentially become part of your team [66] [39]. How is this dealt with? The original developers might be restrictive towards who they allow becoming a part of the core team if they allow anyone at all. In this case, the core team can still be kept very small and to some extent continue with their chosen methodology. To utilise the power of a large community, they have to be able to contribute. Granting them access to a digital Kanban board of the project might be possible, where contributors can pick tasks to complete [67].

So far it seems like traditional methodologies might be applicable with some adjustments here and there. However, the biggest obstacle is simply the fact that open source contributions are volunteer work. That means it is not something you can assume is being worked on from nine to five, five days a week. Volunteer work is something you do in your spare time [44] [10]. Therefore, it is hard to do something like sprints because it is hard to estimate how many developers are available to help during each sprint.

4.2.2. Decentralised

As previously mentioned, according to the agile manifesto, the best way of communicating is face-to-face. This becomes practically impossible when contributors can be living anywhere on the globe. Of course, there are the video systems, but if potentially thousands of people are to participate in meetings over video there could be trouble and chaos. Nonetheless, with the help of email lists, forums and IRC chat rooms developers can communicate. The key is that contributors must be able to assign tasks to themselves which suits their skills and interests, and at a time that they can work [38].

4.2.3. Open Collaboration

Even though having a large community of people can be hard to manage, there are also some very important benefits. No matter the size of your company, you will never have as many employees as the possible millions of contributors that share a passion for open source contribution. Open source has done in the software industry what general science has done in other fields like mathematics, biology, physics and chemistry. It is an environment of peer review [10]. Software development can be very complex and sometimes very hard to debug when errors occur. In his paper [5], Eric Raymond formulated what he called Linus' Law (named after Linus Torvalds). In short, it states that "given enough eyeballs, every bug is shallow" and put in more formal terms:

> *Given a large enough beta-tester and co-developer base, almost every problem will be characterised quickly and the fix obvious to someone.* - Linus' Law, [5]

Due to its peer-review nature, open source projects have been proven to be very secure and close to bug-free which are two very important aspect of software products that potential customers care about. However, seems only to be believed by developers and not by those who lack computer science skills [10].

To accomplish true open collaboration, the project has to have high modularity. That is why traditional development methods like waterfall cannot be applied to open source software development. As Torvalds stated, you cannot have people working in parallel without modularity. This is crucial because of the different schedules of people working with open source. You have to allow flexibility regarding when to work. Modularity decreases the need to coordinate implementations between developers, which is highly needed as contributors do not necessarily know each other or see each other as colleagues [66] [68].

4.2.4. Knowledge Sharing

Another benefit of having a large community is the ability to seek out help from the community and not just via official support services. Developers often post questions on different forums like StackOverflow or designated forums for a given project, if they exist. Having the questions and potential answers to them available publicly means that other developers who might have the same or very similar problems can look up answers to their questions without having to ask them again and wait for replies. Asking questions on forums might not be a characteristic of open source per se but the knowledge sharing is enhanced due to the openness. Much finer details of the software can be shared due to visible source code as well, which allows very thorough debugging if needed [68].

4.3. Project Growth

Once an open source software has grown to a certain size and popularity, one of two things seems to happen. One alternative is that someone decides to create a *fork* of the project and start creating their own version of the product derived from the original. There can be many such forks. The other alternative is what I will in this thesis call an *ecosystem*. It is important to understand the difference between these two alternatives when considering the future of your project.

4.3.1. Forks

When developers want to contribute to a project on a platform like for instance GitHub, they cannot simply go to a repository, download the code, make some arbitrary changes and push the code back up. That could potentially ruin many, many hours of work if you were to accidentally (or purposely) delete files and folders. By default, you do not have access to make changes to an existing repository directly when it is owned by someone else. This is where the concept of forking comes in [69].

When you fork someone's project, a copy of that project at the time of forking is created and put onto your account on the source code platform at hand (for the sake of explaining, I will use GitHub as an example). The copy that now exists on your domain on GitHub is a copy that (given the right licence) you are free to download and modify any way you like. Let's say you have made some changes that improve some part of the software and you would like to contribute that change to the original project. You can do this by requesting the original owner to take your change and pull it into the original

project for others to see hand utilise. This is the main way contributions are being made these days [69].

But you have another option. Instead of proposing the original owner to pull in your changes, you keep your copy of the software and decide to build on it further. At this moment, your fork will (probably) strive further and further away from the original and become a derived work. A rationale for doing so might be that the original software announced that they were changing their licence and you wanted to keep a version of that product which kept the old licence. Another reason for forking like this is to tailor an otherwise more general-purpose piece of software to serve a more specific service, like the Nokia X Software Platform which is a fork of the Android Project. It was developed by Nokia to run specifically on their Nokia X family of devices [70].

4.3.2. Ecosystem

If several forks were to be taken out of your project, then essentially there will be several editions of your software out on the market that will eventually end up as competitors. This might be fine if the project you publish was never meant to be a business intentionally. It also serves as a way for your project to survive, if you were to abandon the original project, some fork out there might still be alive and carry on your vision [69].

If you intended to start a business, however, you would want to avoid too many forks of your project so that your edition of the project is the one that receives the most attention from the community of developers and users. You want to grow an ecosystem around your product to make it as accessible as possible to as many as possible. Even though the whole set of derived forks from a given project are sometimes being called ecosystems, I will for the sake of making a difference not refer to that as an ecosystem because forks try to separate themselves from the original, whilst my definition of an ecosystem is where the original project is still the active root [71].

In an ecosystem, the original product is still being worked on by a core team of developers, being the original creators or their successors. What the ecosystem consists of are additional applications and software surrounding the core products that make it more appealing to a wider group of potential users. For example, if you have a core banking system written in the C programming language, someone could write a *driver* or middle-ware that allows your banking system to be used by other applications written in Java, C++ or any other programming language that would suit particular needs. Another example can be providing a graphical user interface to your banking application, making it easier to do some operations that otherwise had to be done through terminal commands [71].

Tiwana [71] defines the set of elements that make up an ecosystem. The *platform* is the core software of the ecosystem which contains the main functionality and source code. It also contains *interfaces* which are the tools needed for the software to communicate with other software. These interfaces are the drivers. All the different software that utilises the platform are called *applications*. These are systems that have another main functionality than the platform but it uses the platform as part of its main functionality. The ecosystem is then defined as the collection of the platform and all its associated applications [71].

The whole ecosystem model can also be viewed from another angle, where you have written a small piece of software that excels at one specific thing that other, larger companies then

decides to use in their solution. Thus, your project becomes a part of their ecosystem and often ends in acquisitions by the larger company because they want to make sure that components of their ecosystem survive [71].

Software systems today are growing more and more complex and rely heavily on already written code through dependencies and plugins. Growing a platform ecosystem around your product is almost a necessity to be competitive in today's market. Ecosystems enable flexibility through multi-purpose. It is no longer a market where the competition is product versus product, but platform versus platform. Developers and other customers want one platform to solve as many of their problems as possible [71].

4.4. Project Success

What are the metrics that measure the success of an open source project, and what is even considered a successful project? One way to measure it is that the project is doable and that it produces some product that solves the problem it was developed to solve. There are huge IT projects in both the public and private sectors that fail to cost over $US150 billion in the United States and the European Union annually. To avoid such catastrophic numbers, Taherdoost [47] suggest several key components that lead to successful IT projects, see figure 4.6. It should be pointed out that their research was focused on closed source projects inside a company. However, some of the points are universal for development in general, and by being an open source project, some of these factors might not need to be present.

4.4.1. Market and Technological Success

Other research [68] propose that open source project success is defined as an aggregation of market penetration and technology advancements over a set period. In other terms, it is divided into market success and technological success. The model further proposes two sets of factors that affect these success types; extrinsic and intrinsic cues.

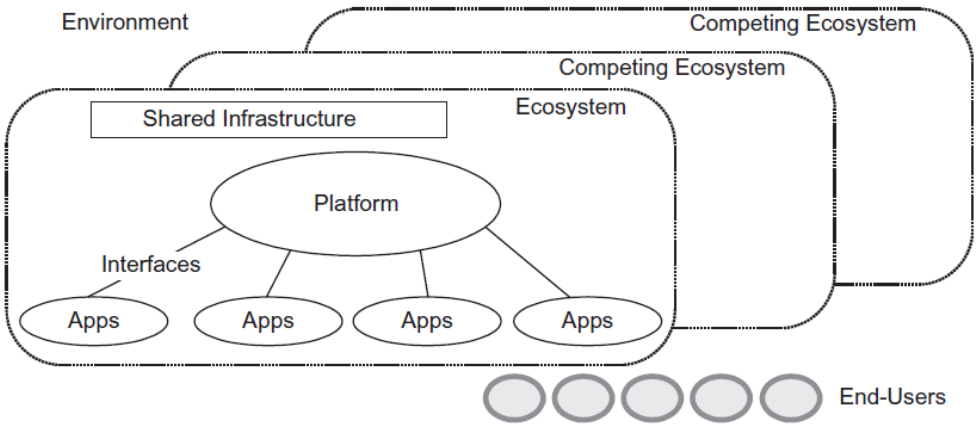

Figure 4.5.: Elements of a platform ecosystem

4.4.2. Extrinsic Cues

The extrinsic cues are the observable factors without investigating the functionality of the project through code. They are surface-level cues that can be understood by any user, developer or not. These cues are often visible on the websites of the project or other distribution platforms like SourceForge and GitHub. They are things like which licence the project uses, the size of the user base or the number of downloads, and the size of the developer community. These cues can also be things like language translations of the project [68].

4.4.3. Intrinsic Cues

The intrinsic cues are the ones that require technical expertise to observe. They are defined as project complexity and modularity. Complexity refers to how readable and understandable to code is. High complexity leads to developers having to either have a very high skill level to read or that the flow of the program is not easily understood and require a lot of time to fully understand. Modularity refers to the ability to work on several different parts of the project in parallel. Breaking the software up into smaller components which could, in theory, serve as individual projects. Linus Torvalds has stated that modularity is alpha-omega for operating system development [68].

4.4.4. Correlations between Cues and Success

The research proposes a connection model as shown in figure 4.7 where hypothesises of positive impacts are indicated by arrows. In their results, most of these hypothesises were true, except there was no significant evidence that technical success implies market success. However, some other research suggests that there might be implications the other way around. That a popular project, in other words, market success, attracts developers who want contributions to famous projects in their portfolio [68].

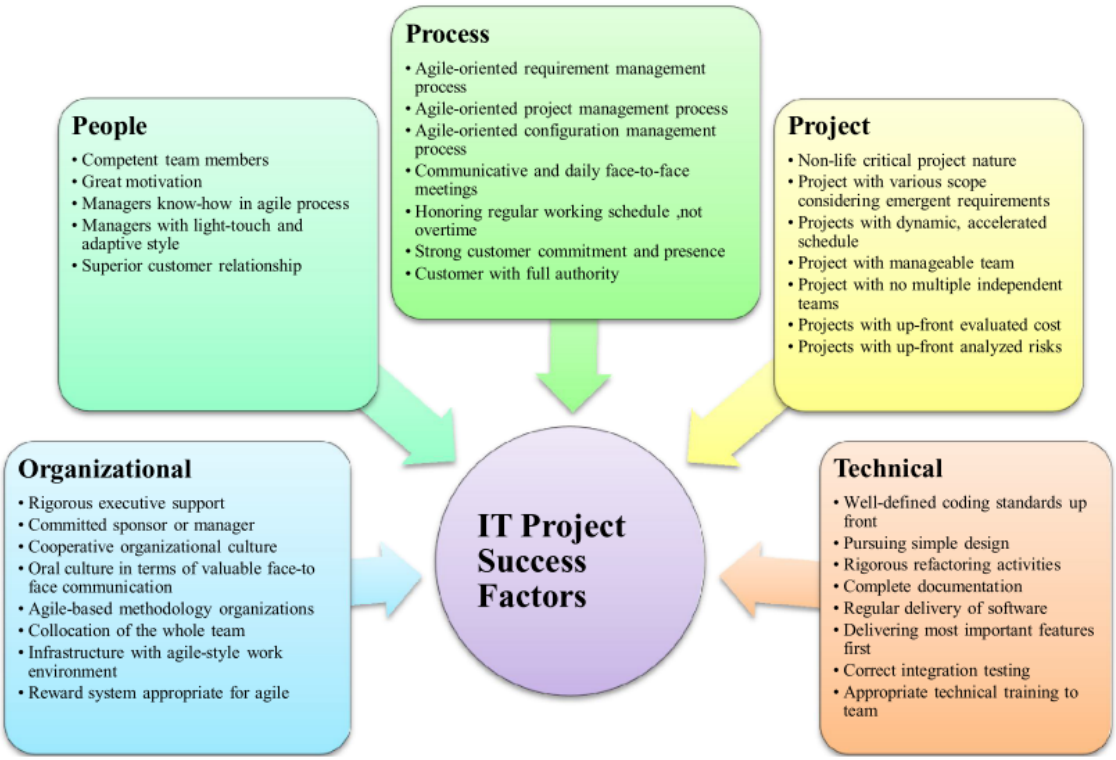

Figure 4.6.: Factors leading to IT Project Success

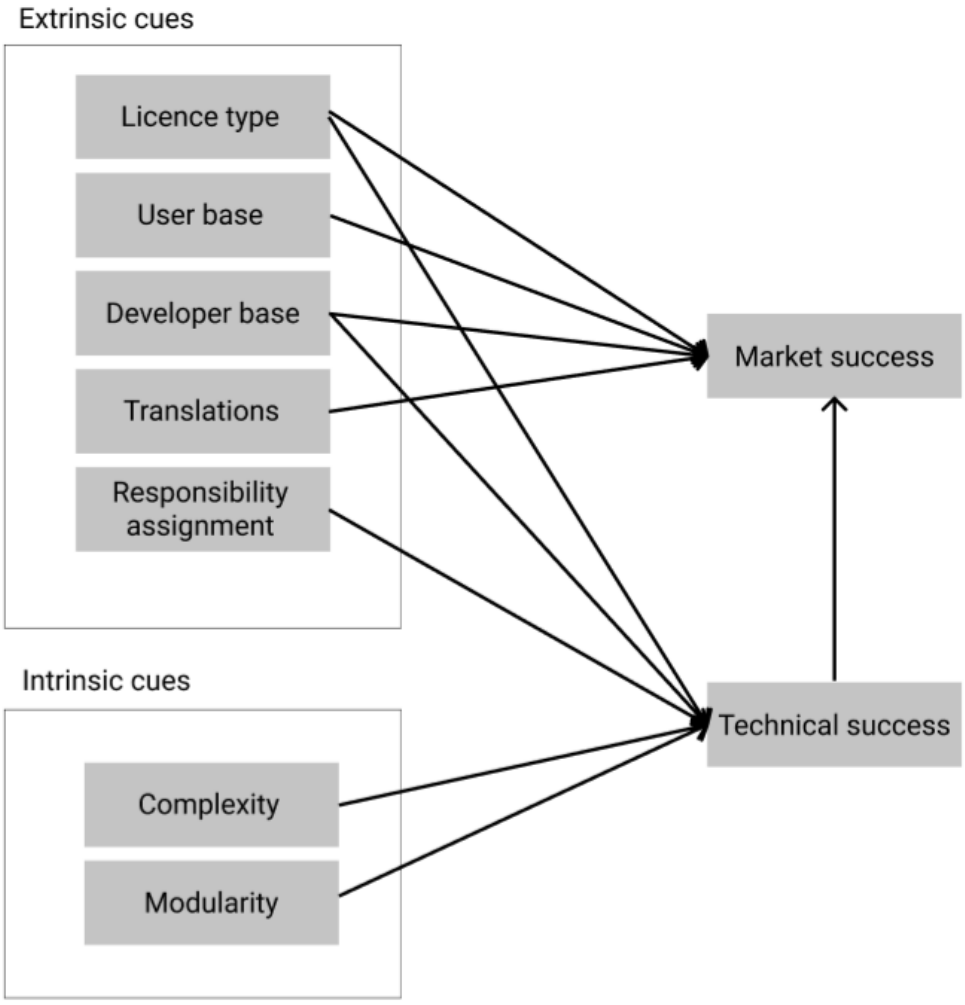

Figure 4.7.: Original hypothesises model of cue impacts on success

5 | Developer Motivation

There have been conducted many studies on the motivational factors for contributing to an open source project but many scholars are still puzzled by their findings and often opt for further research on the topic. Many argue that discovering the incentives behind these contributions can help businesses grow their communities around their products, which is arguably one of the most crucial characteristics of successful open source projects [5] [72] [73] [39]. In this chapter, we will take a closer look at previously researched factors of motivation for developer contribution to projects, both from the contributor's perspective and the project owner's perspective.

5.1. Personal "Itch"

For some developers, writing code is not simply their job but also what they like to do in their spare time. Coming up with pet project ideas is not always easy. Still wanting to do development outside of their work, developers can exercise their passion for already existing open source projects. They use open source projects as practice tools to improve their coding skills. They also get the chance to work with technologies that are not part of their work-related technology stack to discover new interests in other branches of IT which they normally do not tap into.

A big part of software development is automating an otherwise time-consuming, perhaps even boring, task. Having to do cumbersome tasks that take up a lot of time is referred to as having a *personal itch*. "Scratching" this itch is when a developer decides to solve this particular problem and make it easier on themselves. Much like the focus in agile development is directed towards the customer needs, so is it in open source but with the developer also being the customer. The developer faces a problem and decides to solve it. Then he or she thinks that they are probably not the only ones having this problem. If they are at the current time, surely someone else will come along and stumble upon the same problem. In good faith, the "right thing to do" is then to publish their solution so that others may find it. Others can then find the software and adapt it to their own problem, often having to rewrite it slightly and might as well end up improving it thus helping the original developer in return as well. In his paper [5], Raymond presents a set of lessons from his observations of the open source phenomenon.

> *Every great piece of software starts by scratching a developer's personal itch* - Raymond [5]

Another one of his lessons states that the value of taking some already existing and available piece of software and modifying it to specific needs. With the growing codebase that is available through open source, existing solutions for your problem is almost guaranteed

to exist. It might not fit your needs perfectly but it is better to take those solutions and tailor them to solve your problem rather than reinventing the wheel.

> *Good programmers know what to write. Great ones know what to rewrite (and reuse).* - Raymond [5]

A personal itch can often become more than simply a problem that you want to solve, it can become something you are passionate about. Passion is often the main factor that drives volunteer work in the first place. Finding a job that perfectly aligns with your passion is not an easy task, so to follow that passion free time consumed thus resulting in volunteer work, being it for yourself or others. People with a passion for something also tend to have some experience, bad and good. In other words, things they have tried that have been successful as well as unsuccessful. Having an open and transparent project allows outside contributors to locate the parts of the project that needs improvement. After all, things cannot be fixed if people do not know that they are broken [5].

5.2. Mobilising by Ambiguity

Sometimes the personal itch can be something you cannot reach or something you simply do not know how to scratch. You recognise where it itches, in other words, what kind of problem you need to solve, maybe you even have some vague ideas of how to solve it. By looking online for solutions, people tend to be met with a wall of noise [74]. Today, there are so many options when choosing software to solve particular problems and developers would like someone to stand out in the crowd that appeals to their needs. Someone who can scratch their back.

Throwing lots of very specific details in developers face can scare them off, as most people do not know, or need to know, every single detail about the tools they are using. They just need to know that the tools can solve their problem. Thus, using more vague descriptions that sound appealing at first glance will allow many developers to believe the solution works for them. Open source gives the developers the freedom to verify this for themselves because of the ease of acquiring the software in question. Ambiguity and vague descriptions, in this case, helps spread adaption in the community. For example, let us say we have a product X which many developers know and have used for a long time. X is becoming obsolete when tackling a new, rising problem. For the sake of the example, X cannot handle over 1000 simultaneous users in your application. You ask your friends or colleagues "I need to find something that will handle over 1000 simultaneous users", and they then reply "Oh, there is a tool called Y. It is just like X, but it can handle lots of simultaneous users". That immediately sounds like an attractive alternative to most developers. A tool that is like something they already know, but also might solve their problem. I say *might* here because of the ambiguity in the reply given by the friend or colleague. It was never said the exact number of users it could handle. Maybe you need to handle one million users, and maybe Y turns out to only handle 5000 users. This is example is obviously very simplified. The point is that these vague statements serve as leverage for projects to stand out in the noise of other alternatives, so the developers likely to attempt to scratch their back with it. This is a very effective way of mobilising a community of both users and potential contributors to gain interest in a particular project [74].

5.3. Gifting Culture

At first glance, the idea of contributing to open source projects free of charge and with passion as the sole motivation seems quite puzzling to many. Even though, in most cases, there are no direct exchanges of services and payments open source contributing has been compared to the idea of gifting culture [10].

Imagine you are at a bar with a couple of your friends. One of your friends orders a round of beer for everyone at the table. You happily accept your drink and enjoy it, after all, you did not pay for it! After a while, another one of your friends get up and buys another round for the whole table. You enjoy another "free" beer. For is it free? For each beer you drink, expectations build up for you to at some point get up and return the favour and buy a round of beer yourself.

Virtual gifting does not necessarily work the same way as physical gift-giving. Usually, a physical gift has an intended receiver and is not meant to be giving to someone else. That is not the case with publicly available software. Who are the intended receivers of an open source software? Yes, it is meant for those who may have to solve the same problem as the project creator but with no guarantee that these people exist. In other words, it does not become a gift before anyone claims it [10].

Depending on the gift that is given, different levels of moral obligations arise in the receiver. Some believe that acknowledgement and praise are enough because it points a spotlight at the gifter and if these acknowledgements become many, it can raise the gifter's social status and reputation in the community. Others believe that more than words is in order. That gifts in some form or another have to be returned to the project. Open source gift culture differs from classical gift culture in that gifts are very rarely, if ever, rejected. Have many times haven't someone received the most horrible Christmas sweater from a family relative but still accepted it and even put it on right then and there. Contributing code or information to an open source project, on the other hand, might lead to rejection if the contribution is of too low quality or not aligned with the vision of the project owner. This differs from project to project in terms of how close the project relates to the cathedral model or the bazaar model [10].

There is a phenomenon which is called the "field of dreams" which refers to a belief that "if I build it, people will come". This mentality might work in traditional gift-giving where the exchange of the gift is happening at a certain place and time and where both parties, as well as others, observe the event. However, in the virtual gifting environment, this place and time are not known to both parties. Developers can break this mentality by being as present as possible when a gift is received [75].

5.4. Need to Belong

There are many developers today who work as freelancers who work alone from home or other places where there are no colleagues to interact with. These developers tend to want a place to belong to. They seek to be part of a group or a team of other developers with the same interests in technology as them. Open source communities are one of the ways these freelancers can satisfy this need [76] [74].

Flaming has been brought up as a real problem in most forums and mailing lists. Potential contributors hold back their work in fear of being flamed by the community. Is that still the case today? If a person is afraid of flaming, it is more "safe" to start a project of your own than to deliver code to an existing project. But if no one is contributing to each other's projects, then the whole idea of open collaboration falls apart. Developers tend also to hate being criticised for their code, which is why they spend a lot of time "wrapping their gifts" in humble words. Even though famous people in the community tend to flame for no apparent reason, contributors need to be able to differentiate between pure flaming and good criticism [10]. Putting yourself out in the public eye also puts you in the line of fire of negative comments, flaming and Internet trolls. You are more likely to remember the one bad comment over a thousand positive. It can, therefore, be as including and humble as possible when publishing your project. Approach it like you try to encourage the shy kid in class to speak up as well [77].

> *Never write something and don't submit it, it goes against the whole philosophy of free software* [10]

5.5. Renown

An open source community can in many ways be compared to society with its social hierarchy. Many strive to become famous for their talents, knowledge or actions. Contributors who put in a lot of effort and time to further improve an existing project or write a new tool in the ecosystem tend to become "celebrities" in the community. Their voice then carries further and they are more respected when making a presence in the forums. Some developers seek this status to add to their portfolio which they can use when applying for a job that might be a tough competition to get [73].

This status can be acquired simply by word of mouth where people recognise you as a contributor of importance but some companies, often those who have their own forums, have some kind of merit system implemented where contributors can earn visible awards for others in the forum to see. Having these merit systems can be enough to convince a developer to contribute to your project in favour of another [73].

Celebrity status in a community serves another purpose from the company or project owner's perspective. The contribution of these acknowledged developers allows the companies to locate skilled programmers that stand out, which are potential employee candidates. Programmers that are above average in skill are hard to uncover based on other metrics like the number of commits or lines of code. Does a skilled programmer write 1000% more code than the average programmer? What about those who write shorter, more efficient code? Through peer reviews in an open source repository, the developer's skills can be more easily verified [73].

5.6. Economic Incentives

Some companies have built their products on open source software. Sometimes these companies are so reliant on a particular open source component that they want to assure the progress and innovation of that piece of software as well as making sure that its community is kept alive. One way for them to achieve this is by directly funding this project. This is more common when the company is not considered an IT company but might be using

a complete open source solution to handle some, or all, of their tasks [43]. These kinds of services are usually what is known as software as a service, or SaaS. In this case, they are most likely using an enterprise edition of the software, paying for support services or a subscription of some sort. If the company only uses a free version of the software, they can donate funds to the developers of the software directly. However, if the company is, in fact, a software company that uses the open source product as part of their software stack, they might make their own developers contribute to that particular software as part of their workday. In this way, the developers are receiving direct payment from their company, and the company as a whole becomes the "gifter". In most cases what economic incentives comes down to is if the benefits of the contribution outweigh the investment or cost. In the case of companies, the cost might be hours they need to pay developers to execute the work. Even more generalised, the cost is simply referred to as time spent on a particular open source project that could have been spent on other areas [78].

5.7. Choice of Licence

The choice of licence has also proven to be a crucial part of the rationale behind choosing to contribute to a specific project or not. Those believing in the free software philosophy will sometimes refuse to contribute to projects that are not strong copyleft. Some are more liberal but will only contribute to those projects with licences approved by the Open Source Initiative [67] [44].

Most developers in the open source community are familiar with the most common licences and exactly which rights they grant those who use them. Open source is built on trust. The community believes the licences that are approved by the Open Source Initiative keeps this trust between projects and the developer community. Some believe that those who wish to do commerce with open source are testing the limits of this trust with custom licensing. Companies with commercial interests are often motivated by extrinsic factors that indirectly satisfy their needs. This is often a monetary need which is satisfied by doing some work. However, this clashes with the values of the open source community which we have seen are driven by intrinsic motivation, factors that directly satisfy their needs, like with personal itches when developers create software that directly scratches their back [79].

The open source licences clearly emphasise that nobody is to be excluded from development or use of open source projects, and commercial actors that define their own licences need to tread lightly when defining custom licences so that trust and honesty like this are not defiled. A classic study by Mary Douglas [80] looks at the definitions of what is *pure* in this world. It can be compared to the definition of open source where the rights of open source and the Open Source Initiative approved licences are viewed as pure in the open source community, while commercial companies are sometimes viewed as defiling the pure open source. It sparks the question of who has the right to define what is pure. Does the open source definition need a redefinition to enable full utilisation of open source as a commercial business model?

Part II.

Method

6 | Research Strategy and Method

The main purpose of this thesis is to add to the body of knowledge as well as shed light on a problem of importance. Although it would be of great value to solve the problem, it is not within the time and resource restrictions of thesis to accomplish that goal. It is also a case of re-interpretation of the existing theory that might have changed through the course of the years [81].

To fulfil the purpose of the thesis, a scientific research method was applied. Research is mainly split into two categories; quantitative and qualitative research. Quantitative research uses data gathering methods to acquire lots of surface-level data like people demographics, short opinions on different topics or how many times a certain artefact of interest is observed in an environment. These kinds of data are *quantifiable*, meaning that they can be translated into numbers which can be used for calculations and predictions. This category of research results in statistical findings which, if significant, uncover patterns in society [81].

Qualitative research, on the other hand, seeks to dive deeply into single artefact such as a single company or a small group of people, projects, tools, etc. Qualitative data is represented by everything that does *not* translate to numbers. Examples are things like words, images and sounds [81].

In this chapter, I will justify my choice of research strategy, data gathering and analysis methods, and in the next chapter, I will present the rationale behind choosing the case that I did for this study.

6.1. Method

There are lots of quantitative research conducted on open source which usually looks at things like single incentives for developer motivation to justify their validity. To solve my problem definition, a qualitative approach was more suitable. It allowed me to aggregate several previous studies on many different factors to then validate their presence in the chosen case. In this way, anomalies could also be discovered if previous research was, in fact, no present in the case.

6.2. Literature Review

A huge part of this thesis consists of a literature review of previously conducted research and other relevant information which combined with my findings lay the foundation for discussion. A literature review is usually conducted in two steps, or as two different reviews depending on who you ask. The first has the purpose of finding a topic of interest in the

thesis as a whole. Because I was given an assignment text (see appendix A.1, I knew that the thesis was going to be about open source. This is way too broad a topic, so a literature review was conducted to find more specific topics to research. The review resulted in the problem definition described in the introduction.

My approach was *inductive* meaning I started by learning about the case before finding relevant theory. This approach allowed me to avoid bias from previously conducted research on the topic. Therefore, the second part of the literature was conducted after I was done investigating the case, and it focused on finding a theory that could explain or justify the findings [82].

The material for both of the reviews mostly consists of journals, reports and books, but also some video material as well. See the complete bibliography. I made sure to always criticise what I was reading and compare it to other similar research.

6.3. Data Generation

Many data generation methods are applicable for gathering qualitative data. As the overall strategy for this thesis, a case study was used and as ways of data gathering, I conducted an interview and the rest of the data is collected through observations [81].

6.3.1. Case Study

The main purpose of a case study strategy is to focus on one artefact or a "thing" that is to be researched. It can be on many different levels from large organisations all the way down to a single decision. This artefact or thing is called the case. Usually, several different data generation methods are applied such as interviews and documents which are used in this thesis [81].

A common characteristic of case studies is the conservation of a natural setting. This means that the environment that the case operates in is not affected by the arrival of the researcher. By being like a fly on the wall, the researcher's presence can be unnoticed. There are several types of case studies that can be conducted based on what kind of outcome that is wanted. These are *exploratory*, *descriptive* and *explanatory* studies. An exploratory study is conducted when there are little to no knowledge on a topic and the research goal is to understand a problem more thoroughly. Descriptive studies usually result in a detailed analysis of the case in a certain context. Explanatory studies take the descriptions even further by tying them together with literature and explaining why certain events occur [81].

Based on the existing knowledge base on open source software development, I intended to conduct a descriptive study and analyse the case in great detail. However, since the case I chose was a more extreme instance rather than a typical instance, I adapted my approach and found that an exploratory study suited the thesis outcome better as the results from the analysis sparked new questions.

Case studies as a strategy have been criticised for being too biased towards the case under investigation. This means that the results are not necessarily generalisable enough to be applied elsewhere. However, case studies provide theories and implications for further studies [81].

6.3.2. Documents

The main source of data in this thesis comes from *documents*, and only from the type of *found documents*. A document is simply all kinds of written material produced by an organisation, formal as well as informal. Found documents are data that is already generated once the researcher enters the scene. Formal documents contain data like sales figures, personnel records and official announcements. Examples of informal documents are notes, emails, chat messages, forum posts and so on. Documents are not just textual content but expand to visual content as well like images, graphs and videos [81].

Gaining access to documents is relatively easy compared to other data gathering methods like interviews or questionnaires which require a lot more time invested in planning, issuing and conducting. Most forums and email lists are made public, especially in open source projects because most of the interaction is in written format. Document-based research is ideally conducted on the Internet due to the vast amounts of data that exists [81].

6.3.3. Interview

There are three types of interviews; structured, semi-structured and unstructured. Structured interviews are more common when you have several people you want to interview and you wish to ask the same questions to every single interviewee. Semi-structured are when you still have a set of questions you want to ask, but you allow the interview to diverge from the original order of questions based on how the interviewee answers. The unstructured interview is much like a conversation about a topic when both the interviewer and interviewee talk freely and exchange ideas [81].

In addition to documents, one interview was conducted for this thesis to gain more detailed insight into the history of the case that was not well documented publicly. Interviews are useful when there is a need for additional details, and when you have more complex questions that are not easily answered without asking someone directly. They are usually open-ended, meaning that the answers might vary from each time the question is asked because the interviewee might sidetrack differently. This way, topics can be viewed from different angles as well as the chance of unexpected findings. Therefore, I decided to prepare and conduct a semi-structured interview where I encouraged the interviewee to answer my questions in as much detail as possible, sometimes moving away from the questions as well, or answering several later questions in an earlier question [81].

Recruiting

When recruiting interview candidates, they have to fulfil certain criteria. Those include being reflected on the topic of interested and that they are in a position to comment on the topic, by for example being a trustworthy and knowledgeable source. This is a strategic recruitment approach [82].

For my interview, it was important that the interviewee could elaborate on their personal experiences early in the lifetime of the case as well as in the current state of the case. A suitable interview candidate that fulfils these criteria is the CTO of MongoDB who was also a founder. I decided to reach out to him by his email address which was publicly available on MongoDB's website. In the email, I presented the purpose of my thesis and why I was contacting him. He was quick to respond and allowed me to pick a time slot of

30 minutes in his calendar wherever it fit my schedule. I estimated that one respondent would suffice, thus not pursue more than one interview.

6.4. Interview Guide

According to Tjora [82], a well-structured interview is split into three main parts. A warm-up phase, a reflection phase and a closure phase. In the warm-up phase, very simple questions and light questions are asked for the interviewer and interviewee to become comfortable with each other and the interview situation. This can also be as simple as small talk, which is what I did. The reflection phase is where the more complex questions are asked, and this is the core of the interview where the relevant data is being gathered. In my case, I had prepared an interview guide consisting of open-ended questions. Finally, the closure phase normalises the situation and it is common to inform how the research will proceed from here [82].

6.4.1. Interview Situation

Many factors can affect the interview situation. It is important to make the interviewee feel comfortable which enables them to speak more freely in the reflection phase. Another factor is for you as an interviewer to dress and appear in a way that suits the situation. Differences in social status and skill level, as well as age and the spoken language of the interview, are all factors that weigh in [81].

Research shows that there is a considerable difference in the medium used when conducting interviews. Face-to-face interactions are more valuable due to interpretations of body language between the interviewer and the interviewee, in contrast to conducting a purely spoken interview over a phone call. Video calls are somewhere in between the two but when either party becomes aware of the screen between them, it can affect the interview [81].

In my case, the interview was conducted over a video call which I recorded. To record the interview is common because you as an interviewer can focus on the interviewee and the flow of the interview itself rather than taking notes. I decided to dress business casual which resonated with how the interviewee dressed. Breaking the ice was not as easy as a face-to-face situation because of the lack of natural eye contact as well as an introductory handshake. The interview was conducted in English, which is not my mother tongue. The difference in social status, as well as age and skill level, were high, with me being a student and him being the CTO and founder of a multi-million dollar company.

6.5. Analysis Method

The interview was mainly meant to be an additional data source to the much larger document-based data already gathered, as well as the literature studied. I transcribed the recording of the interview to be able to search through and analyse it more effectively. When transcribing, the key is adding details like pauses and sounds because they might have a significant meaning in the analysis. Even though it might prove to be irrelevant, it is better safe than sorry to include it [81].

My analysis mostly consisted of documents being treated as vessels. This means that

they are containers of data. I applied themes analysis where I was not looking for specific keywords or terms but rather a broader topic over several chunks of data. What I did was comparing observed topics in documents along with the interview answers with characteristics and observations found in the literature. By doing this I wanted to find anomalies and exceptions that lay foundations for further research on more specific topics. Documents studies, as well as all other data generations methods, can never paint an objective picture of reality. However, be researching a large enough quantity of documents you move closer to objectivity.

6.6. Quality

All qualitative research is influenced by the researcher's interpretations. To assure the quality of the research, we use three main indicators of quality. They are *reliability*, *validity* and *generalisation*. They are each defined in the following subsections as well as the measures I have taken to improve the quality of my research in these three areas [82].

6.6.1. Reliability

Reliability defines the reliability of the research. In an ideal world, another researcher should be able to conduct the same research and arrive at the same conclusion. Regarding my interview, none of my questions involved sensitive data where the interviewee could hold back information in other interview situations. Our prior relationship was non-existent and would not influence the answers. On the other hand, in the document data gathering, some data was gathered from articles which may be biased. This might reduce reliability. To counteract this, I made sure to read articles and documents from different points of view to see all nuances. My personal view on the topic of this thesis might lead to bias affecting the research. As my position as a novice researcher, I lack experience with conducting large scale research like this thesis. It may have affected my approach and how I analysed my findings [82].

6.6.2. Validity

Validity defines if our findings are in fact answering our research questions. The validity can be strengthened by making sure your research has roots in other relevant research. My discussion chapter aligns my findings with previously conducted research on many of the same topics, where many of the sources are highly acknowledged in the community. Originally, my research was going to purely document-based but I decided to conduct an interview as well to get a better understanding of the topic and details around it, which also improves validity. Talking with one of the founders of the case can both increase and decrease validity. It can be argued that it can decrease if the interviewee wants to provide answers which put their product in a positive light. However, it can also increase validity as I can verify already defined questions and get answers straight from the main source [82].

6.6.3. Generalisation

Generalisation is how the findings of the research can be transferred or applied elsewhere. Case studies can be argued to have low generalisation because the knowledge only relates to the case. Broader knowledge, however, can be of value to construct new concepts, theories and implications which lay foundations for further research. My chosen case is already

an edge case, so it can be argued that generalisation might be low. However, if this their method is simply untested, it can still be generalisable to a degree [82].

6.7. Limitations

The topic of the research has been altered several times, sometimes drastically. The method had to be adapted to these changes which have taken up a lot of time and resources. Had the last of these changes not occurred as late as they did, I might have conducted more interviews and even applied more ways of data generation like questionnaires. The literature study was also impacted by these changes and ended up more limited than I had expected.

7 | The Choice of Case

For this thesis, MongoDB, the most popular open source, NoSQL database today [83], was chosen as a suitable candidate of a sustainable open source project. It was chosen through a selection process which involves a set of criteria that had to be fulfilled. It was also chosen due to its innovative impact on the software industry which might be an interesting factor to discuss later on. In this chapter, I will present the different practical criteria behind the choice of case.

7.1. Practical Criteria

A lot of characteristics of a successful and sustainable open source projects were presented in the prestudy chapters, some of them were backed by several studies and some were just observations that might not be as relevant to this research as first assumed. These criteria are mainly thematic, which means that they affected the development process of the project. However, the case must also fulfil a set of practical criteria which indicates if it can be analysed in enough depth for this research. The practical criteria are defined as follows.

7.1.1. Time alive

The chosen project should have existed for some time so that we do not have to predict the future of the software but can analyse its past. However, the tech industry changes rapidly. Therefore, the project should not be too old either. For the sake of this research, I define the time alive span to be three to fifteen years. At the time of writing, the software needs to have been released between 2005 and 2017 and still be *active* today.

7.1.2. Activity

It is important that even though the project was released some years ago that it still is active. There are several ways a project can be characterised as active. One metric can be the number of pull requests in the project repository over a set period. These pull requests can be from both a core development team or project owners, or by external contributors. These two sources might also be important to differentiate between. Having an inactive core team that never accepts external pull requests, or never answers questions on the forum or comments might impact the survival of the project.

7.1.3. Success

The project chosen should also be fairly *successful* which in this case means a high number of downloads and also preferably a high number of active users (active in this case means reported use over some time). A highly successful project can also be an indicator of the

impact on the industry in terms of innovation. If the project is highly innovative, it might be an important part of becoming sustainable.

7.1.4. Growth

In terms of growth, the steeper the curve the better. Having a fast-growing project comes with its own challenges and obstacles and how the team overcomes these might be crucial for a sustainable project. In addition, how open the team is to input from external contributors is crucial for the project to be popular and or its survival.

7.1.5. Innovation

It is not mandatory, but I think it would be more interesting to analyse a project that has had a revolutionary or innovative impact on the tech industry. It would be interesting to see how being an open source project has affected this aspect.

7.1.6. Start-up

Ideally, the project should not be an internal project in some large million dollar company. This is because having a start-up that has to find their own investors and does their own venturing will potentially take more thought-through decisions, and there are few projects out there with large enterprises having their back if the project should fail.

7.2. Thematic Criteria

Choosing MongoDB as the case for this thesis was mainly based on the fulfilment of the practical criteria and that, on a surface-level, it was labelled open source. However, after studying it more closely, MongoDB turned out to have a different approach to the whole idea of open source than most other projects and the definitions of open source projects in general. From my findings, MongoDB turned out to be an edge case in the open source world but a part of it nonetheless. The original intention of this thesis was to analyse a more generic example of an open source project to uncover its success factors, fragility and eventual other secrets to sustainability. However, instead of abandoning MongoDB as a case in favour of another project, MongoDB's success story piqued my interest and I wanted to see if their utilisation of open source and the platform ecosystem was the key to their huge commercial success.

Part III.

Case Study: MongoDB

8 | The History of MongoDB

Today, MongoDB stands as the number one choice of *NoSQL* database system in the world [83]. The term NoSQL was first defined back in 1998 by Carlo Strozzi who used the term to state that his "Strozzi NoSQL Relational Database" did not support the traditional Structured Query Language (SQL) interface [84]. However, the term used by Strozzi does not have the same meaning is NoSQL in the 21st century. MongoDB has grown from a small team of developer into a large company with over 1900 employees and $250 million in annual revenue and has built a great ecosystem based on open source [20]. This chapter will take a closer look at the history of MongoDB and their rise in the database market.

8.1. Traditional Database Systems

To better understand why a NoSQL database was innovative and much needed, we first need to look at traditional database management systems or DBMS. Before computers became commercialised, companies used to store all of their data on customers, employees, economy etc. on paper and in folders. As their data collection grew it became tedious to organise and maintain due to a large amount of physical material. It was taking up lots of storage space and designated people had to be hired to keep track of every single piece of data. When digital databases came about, this task became much easier by using a DBMS which is defined as "a set of software and an operating system that is used to maintain a database". The database systems allow the user to perform so-called CRUD-operations which stands for create, read, update and delete. This way data could be easily accessed, modified and maintained by companies and also allowed them to gather more data as they no longer required physical storage space for folders and papers [85].

A DBMS uses a function called *querying* which allows the users to "ask" the database to find certain information for them. A query can also be used to generate new information like "how many of our employees are male" or "how many sales were under $100,000 last year". There are many different types of database management systems but the most common, still to this day, is the relational database management system, or RDBMS. The data in RDBMS are organised in tables, probably to feel similar to ways of organising data before the computer. The tables are constructed of columns and rows, where a column

Figure 8.1.: MongoDB Inc. Logo

in an RDBMS is called a *field* and a row is called a *record*. An example of a field can be the name of a product, the price of that product, the weight of the product and so on. A record contains a specific entry where each of the fields is assigned a value (even though fields are allowed to be empty some times). An example is a product named "apple", the price of "$1" and the weight of "100g" [85].

One of the most important features of an RDBMS is that the database can be spread across several tables and connect them together through what is called *relations*. This is in contrast to flat-file databases where all of the data is stored in a single table. These relations are what allows the users to perform advanced and complex querying to locate exactly the data they need. This is possible by using what is known as SQL, or Structured Query Language. It consists of several commands to perform the previously mentioned CRUD-operations, namely Select, Insert, Drop and Delete. SQL allows aggregations of several tables to be made temporarily so that the user can see connections between data. For example, one table might contain a client name and address, another table might contain an address and the price of the house at that address. By only knowing the name of the client, the user can construct a query to yield the price of the house in which the client lives [85].

The main advantage with these systems over physical folders and papers is that the database becomes accessible by more than one person or a small designated group. Queries can even be performed close to simultaneously by different users on the same database. Another advantage is that the data retrieved is presented in a well-formatted manner and is concise each time it is retrieved. This is due to the structured table format implemented by the RDBMS. However, an RDBMS is not the solution to every database requirement. There are some disadvantages to these systems as well. Structuring the data this way can easily become very complex and require personnel which specialise in this sort of table management. Even for skilled people, setting up a robust and consistent RDBMS can challenge even the most experienced of developers due to complex data relations. It might sometimes also require a more powerful machine to run the system on once the company's data collection grows. There are many implementations of RDBMS out there including but not limited to MySQL, Microsoft SQL Server and Oracle. All of these use SQL to perform the operations on the database [85].

8.2. The Limits of RDBMS

With more and more data being generated on the web from social media, advertisements and new web applications, developers started to realise that traditional SQL databases did not scale very well with this rapid expansion of the web. RDBMS relies on structured data which can be organised into tables. Sorting the data in different tables like this requires the use of primary keys and foreign keys which allows the data tables to be indexed similarly to tree structures. Assigning these keys becomes hard with large amounts of free text and other unstructured data generated by users online. It also struggles with data objects that are taking up a lot of storage space, such as high-resolution video. Web applications were becoming more and more complex and demand several new features from developers. Because of this, a trend in the market was that developers wanted to use less time maintaining database relations and more time developing their applications [85].

Enter Eliot Horowitz and Dwight Merriman. In the early 2000s, they were both working

at a company called DoubleClick which handled internet ad services. They experienced the explosion of data on the web first hand and came up with the idea for a document-based database after realising that they had built over twelve custom workarounds for their traditional databases to solve their problems. They never found existing solutions efficient, flexible and scalable enough to solve their particular tasks. Even though relational databases were, and still are, very powerful, in terms of big data they lacked a simpler way to store data as objects instead of relations, and a way to scale horizontally. In 2007, they decided to experiment with ways to improve these features. They drew their inspiration from cloud computing where software and services were spread across multiple machines to balance workload. In addition, they observed that many applications were becoming so complex that they could have over 70 relations in their database just to store a user profile. Developers wanted more rapid development and releases, and they wanted a database that "just works" taking the load of database administrators. They realised that a lot of time could be cut from relation management, thus proposing a document model instead [86].

8.3. Why NoSQL?

As mentioned, an RDBMS does (when designed and implemented properly) provide a robust and consistent database model. However, the data structures used makes this kind of system slow when performing large tasks and queries. Also, they scale poorly horizontally due to all the relations between tables. If horizontal scaling, or *sharding*, cannot be executed properly, then the time it takes to retrieve data from a single database source scales with the volume of data that is stored. It also means that all your data has to be stored in a centralised location. This means that users in Australia might have to access a server in New York which can lead to annoying delays. You can perform vertical scaling, or *normalisation*, by upgrading the server machine with better CPUs and other components, but this will peak when there is no more room in the physical machine or when there are simply no better components out there. An RDBMS strives to follow the CAP-theorem [85].

In short, if an RDBMS follows the CAP-theorem it has:

- Consistency: Every time a user requests to read something from the database, the most recent entry is returned or the user is informed that an error has occurred.
- Availability: The user can always request data from the database and will always be given a response that is not an error. This means that the most recent entry might not be returned.
- Partition tolerance: The Internet is by no means foolproof in regards to message delivery. Partition tolerance means that the system keeps operating even though some messages are lost along in the network.

To make sure the RDBMS follows this theorem requires a lot of work, so huge amounts of both time and resources are invested in database management. As mentioned, when the amount of data becomes humongous, performing queries on the database takes more and more time and becomes very hard to maintain. This time consumption is the main reason the NoSQL-databases were developed. They sacrificed some of the consistency provided by relations in an RDBMS for faster performance as the databases grow. This was a much more crucial metric in, for example, social media and other platforms with huge amounts of data. The name MongoDB is a play on the word *humongous* and refers to its ability to

handle huge amounts of data [87].

Implementing a NoSQL database requires much less time and effort invested in designing detailed and complex relations. MongoDB and other NoSQL solutions operate using document-oriented storage which means that the data is stored together in the same document instead of spreading out across tables in need of aggregation of advanced queries to acquire. However, it is common to think that NoSQL stands for "No SQL", but it does, in fact, stand for "Not Only SQL" because SQL operations are still supported in some NoSQL implementations. Even though a NoSQL database like MongoDB requires less work to set up and requires no predefined schema, one should not blindly begin storing data without some idea of how the data is going to look. It is advisable to create a data model, but in contrast to traditional RDBMS where this is done on the database level, in NoSQL this is usually done on the application level. This means that even though you have absolutely no idea of how your data is going to look, or if it is completely dynamic changing its structure all the time, a document-oriented database can handle it [87] [86].

NoSQL databases are (in general) cheaper to maintain because you do not require as many system admins to manage the database, both maintaining and expanding if required. Expansion is the main way to cut cost due to no schema requirement. Cost cuts can also be made due to the horizontal scaling ability that NoSQL possesses. This means that if you need to expand your storage units or processing capacity, you add an additional node in your cluster. You do not need to buy the largest, most powerful machines and components, but you can combine a set of several cheaper machines into a cluster [85].

It is important to state that even though it sounds like MongoDB and other NoSQL databases has many advantages they are by no means going to replace RDBMS entirely. Some systems might have requirements that can only be solved by an RDBMS, for example, some kind of system used in the medical field that has to be available all the time. RDBMS is often the better choice where data has a tight relation, such as a patient journal [85]. RDBMS are also applicable to normalisation which in the world of databases is to reduce, or often remove, redundancy of data. This allows the data to take up as little storage space as possible, which also results in better performance. But in regards to MongoDB, their value proposition was to tackle the coming era of big data [86].

8.4. Original Business Plan

When Horowitz and Merriman sat down to the develop what would solve the ever-approaching big data problems they and others had, their original plan was to develop an app engine, much like the Google App Engine, and a database system that together would form a *platform as a service* (PaaS). A PaaS product grants customers a set of resources and tools to solve their problems often without the need for support. It means that instead of inte-

Figure 8.2.: 10gen Logo

grating a product, for example, a database system, into their own ecosystem, they simply outsource the handling av the database to the service. All they have to do is provide the service provider with the data they want to store, the service provider handles the lest. They originally looked to simply combine several already existing, open source solutions to form their platform but the lack of a suiting candidate for the database system made them create their own. As previously mentioned, Horowitz and Merriman had already built several niche database solutions for other companies and projects. They decided to build a solution that would not only satisfy a niche but that would serve as a generalised solution for frustrated developers that wanted less to do with databases. They decided they would build their PaaS from scratch, both the app engine and the database system which at the time where called *ed* (for Elliot and Dwight) and *p* (for platform) respectively, later renamed MongoDB and Babble [87]. A database system has previously been defined but it is important to define an app engine as well because its very nature has had an impact on a very important decision in the MongoDB history.

Today, Google's app engine has become so popular that the definition of what an app engine is refers to Google's implementation. What it really consists of is an infrastructure that allows developers to build and host applications on someone else's servers and data centres, for examples those owned by Google. The 10gen app engine would be written in server-side JavaScript, being the so-called "language of the web" and a very popular programming language at the time, and that would also affect how the syntax for their database system would look. From a developer perspective, this meant less time learning new technology as there were many similarities in the way they usually programmed and how the database interface work feel. This kind of solutions, called cloud platforms or similar, allows the customers to rent computing power through the Internet without the need for powerful machines at their physical disposal [88].

However, most of their potential customers and beta testers flinched at the thought of having to use an app engine. This technology was considered restricting and even unnecessary at the time. In 2008 when Google's App Engine launched, all applications had to be written in Python and were limited to 500MB of storage among several other limitations. So when 10gen announced that their PaaS would include such an app engine, it was not well received. Their database project, however, was of interest to many who were also tasked with handling the global expansion of data. Their plan to launch the platform as a whole package was not successful as stated by Kristina Chodorow, one of the four early developers on the project, 10Gen had "practically no users". They knew that a bold decision had to be made [87].

8.5. Why Open Source?

Later that year, as Chodorow so nicely puts it, they "ripped the database out of the app engine and open sourced them" as two separate projects [87]. However, according to Horowitz, the plan was to have the project open sourced all along, even the full app engine. The reason was that developers often are scared of proprietary software solutions, especially when it comes to security and handling personal data. By open sourcing the project, every developer could inspect and verify the code and eventually figure out exactly how it worked. In addition to this, the goal for 10Gen was to create general-purpose software which Horowitz believes can never become true general purpose and widely adopted unless it is open source, again because of the fear of closed source.

When asked for his reflections on how MongoDB would look if they had not open-sourced the project, Horowitz stated that

> *No one would have used it.*

which sets things into perspective because of how huge MongoDB is today. When your project is open source, a whole new community of people and companies becomes interested in it, quite many simply because they are very interested in open source and want to push that paradigm forward.

> *You open up a whole new avenue for yourself as a way of getting people interested and exited about your product.*

This helps to spread the news of your product by word of mouth in the ever-growing, global open source community.

It seems being open source helps with getting the word out about your product, which is what you would want as a new startup. But what about trade-offs? Is there anything that you miss out on when choosing open source over closed source? As previously mentioned, open source projects are usually free of charge and the licensing is commonly understood, so anyone can download it and start working with it right away. From a money-making perspective, having the product free of charge means losing revenue from individual sales. You also end up losing the users that do not use it correctly and then do not ask for help. These users might end up having bad experiences with the product and you cannot do anything to help them. Horowitz realised this and went to great lengths to be present when he showcased MongoDB to his friends and others who were existed about the project. He even stated that he used to reply to every single post about MongoDB on forums online to make sure people knew that his assistance was just a comment away [89].

8.6. Early Stakeholders

At the beginning of the project, Horowitz and Merriman ventured to gather funding and their first potential customers were developers who facing challenges with the large amounts of data at hand. With their impressive portfolio of previous, very successful startups, Horowitz and Merriman gathered quite many investors and raised as much as $81 million through venture capital funding. Big names at that time, and also today, like Intel Capital and Red Hat, were some of the most eager. Funny enough, SourceForge was also involved very early on. As for stakeholders, it was mainly the team themselves as the investors simply put their faith in the 10Gen to come up with a feasible solution. After all, they were experimenting and they were never sure if the product they were developing was even

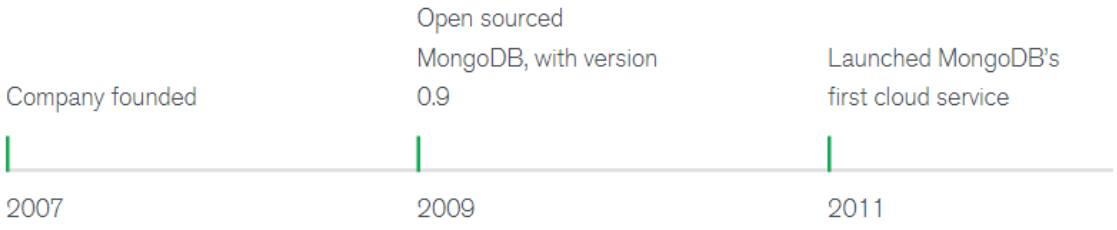

Figure 8.3.: Milestones from the MongoDB website's about page

something that would satisfy their own needs, much less the needs of others [89]. As with many software projects, there is always a risk that the project would not be as feasible as first promised but with a large amount of capital raised 10Gen could take their time to develop the platform.

As mentioned, 10Gen was not the first startup that Merriman and Horowitz had been a part of. They both worked on DoubleClick, an Internet ad serving service which Merriman founded and Horowitz worked on as a software engineer. In the early 2000s, Horowitz and Merriman left DoubleClick to found ShopWiki, a search engine for e-commerce. DoubleClick ended up being acquired by Google in 2007 for $3.1 billion, and ShopWiki which was acquired by Oversee.net which at the time operated several large comparison sites for travel and finances. The CEO and President of Oversee.net at the time, Jeff Kupietzky, stated:

> *ShopWiki is the most comprehensive shopping search engine in the market.* - Jeff Kupietzky [90]

Only two years after ShopWiki was founded, Merriman and Horowitz left ShopWiki to found 10Gen. There is no doubt that these two had an impressive portfolio of successful startups to show to potential investors when venturing for 10Gen. Horowitz was even named one of the top 25 entrepreneurs under the age of 25 by BusinessWeek in 2006 [91].

How do previous accomplishments by developers affect projects? It was clear that both Merriman and Horowitz had made names for themselves as entrepreneurs, quite successful ones at that too. How would the lack of this reputation have affected the product?

> *It's not about the developers, it's about the organisation, and that the organisation is someone that you can rely on that is not going to destroy the product by accident.* - Eliot Horowitz A.2

By being open source, it allows surveillance of the code by the eyes of the community to avoid these "destructions" that Horowitz speaks of. For them, the more valuable asset from successful startups was experience running companies, not necessarily the amount of funding from investors. They realised that there were resources to be leveraged by being open source A.2.

8.7. Expansion

MongoDB was growing. As the years went by, more and more companies were interested in a way to tackle the data volume problem and sought out MongoDB as a solution. They

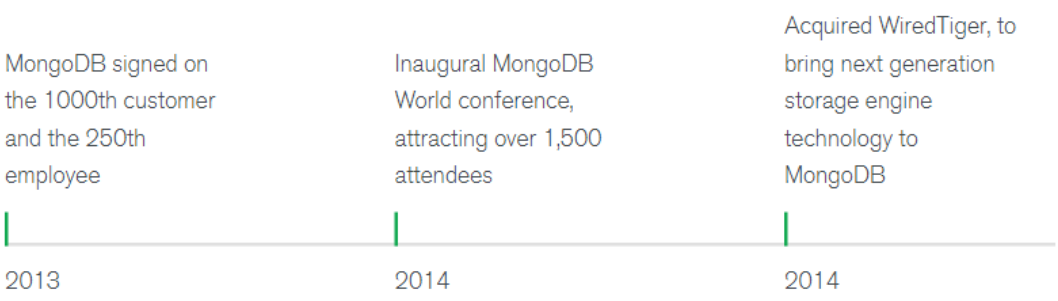

Figure 8.4.: Milestones from the MongoDB website's about page

had managed to establish a great reputation in the open source community and their product was being praised. In 2011, 10Gen launched their first cloud service, a year after they hired their 100th employee and started to provide 24/7 support for their customers. Word of MongoDB spread like wildfire, and only a year later 10Gen signed their 1000th customer and 250th employee. They also realised that it was the name MongoDB that was on everybody's lips, while the name 10Gen were rarely spoken in the same context. Therefore, they decided to rename the company MongoDB Inc. to relate closer to their product. They launched their first MongoDB conference called MongoDB World in 2014 attracting over 1500 attendees. In the same year, MongoDB Inc. acquired the company WiredTiger who make a storage engine of the same name, which was also open source. MongoDB has great performance and scalability for applications that relied primarily on database reads. However, their current storage engine was not performing nearly as well for writing. They had plans to build a new engine that would solve these problems for them. However, the developers over at WiredTiger had gotten word of their troubles and decided to fork MongoDB, implement their own engine in their system, send it back and say "Hey, will this work?". WiredTiger had achieved much better scalability for database writing, implemented it in MongoDB, and MongoDB did not have to lift a finger. This would allow MongoDB to be a good choice for customers in IoT with lots of sensor data being written to databases, data logging systems and other write-heavy scenarios. And so WiredTiger was acquired [89]. MongoDB was awarded one of the best places to work by Glassdoor that same year, and later in 2016, they hired their 500th employee [92].

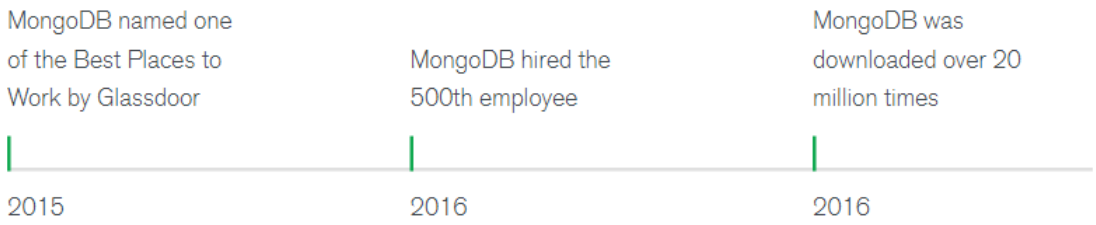

Figure 8.5.: Milestones from the MongoDB website's about page

8.8. MongoDB Today

MongoDB is still strong believers in open source and provides a community edition which is completely free of charge, as well as an enterprise edition which is under a commercial licence but is free to try out for evaluation purposes and non-commercial development. Large enterprises are often very concerned about security and data breaches so the enterprise edition of MongoDB provides additional security features on top of the community edition to satisfy these concerns. This additional layer of security and support is what

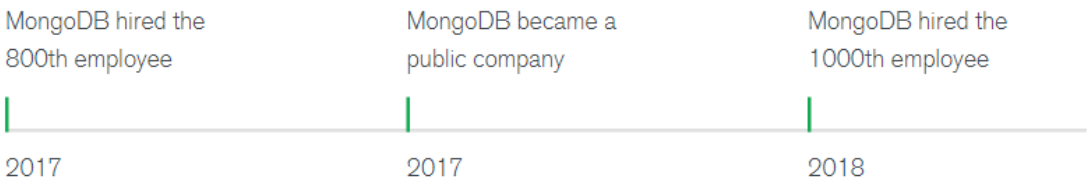

Figure 8.6.: Milestones from the MongoDB website's about page

MongoDB monetises on primarily, because of the commercial licence. This model is called *Open Core* and refers to the fact that the core part of the system is open source, but there are additional proprietary features on top which had to be paid for [93].

Even though MongoDB's success escalated when they dropped their initial plans for a complete platform and to solely provide the database, in recent years MongoDB has moved back towards being a full-blown platform in itself once again, which according to Horowitz is what they always envisioned MongoDB to be. The important thing to note is that every single expansion, new service or improvement has one common purpose, to make data handling as easy as possible for all developers. MongoDB is still all about making the database a problem of the past [89].

Today, MongoDB offers several services that target specific parts of the development stack as they move closer and closer to the application layer as well as expanding the stack. These tools include MongoDB Stitch for front-end developers to use a serverless service for their database needs and MongoDB Atlas which is a database service in the cloud which allow developers to focus more on their application development rather than database management. MongoDB Inc. has also recently (as of spring 2020) acquired Realm an open source database system which is one of the most popular ways of working with data on mobile devices. This new fusion, called MongoDB Realm which is soon to be released, allows real-time data synchronisation between desktop and mobile devices even with different operating systems. As Horowitz stated in the announcement of MongoDB Realm at MongoDB World 2019, "MongoDB will be the best way to build data-intensive applications anywhere" [94].

The task of building a general-purpose database that will fit every need is not to be taken lightly. Along the way, the battle between SQL and NoSQL still waged and still does today. Even though much of the SQL based technology was outdated when MongoDB entered the scene, by no means was it the end of relational databases. Also on the open source scene, tools like PostgreSQL had constantly been improving, and other NoSQL tools like Amazons own DynamoDB, Oracle NoSQL and CouchDB just to name a few are still trying to fill some of the use cases where MongoDB can be bested. However, in an interview, the current CEO, Dev Ittycheria, stated MongoDB was way ahead of their competitors [95].

I think three, four years ago, it wasn't clear who the winner of the NoSQL

Figure 8.7.: Numbers from the MongoDB website's about page

> *market was going to be. We were ahead, but there were other people who were within shouting distance. I think that's become very clear who the winner of the market is.* - Dev Ittycheria [95]

MongoDB is constantly expanding its product farm as well as improving their existing solutions. The fact that MongoDB today is the champion in the NoSQL marked can have many reasons, some financial, but also in regards to how the company is structured, how they interact with the open source community of developers and their interaction with their potential and existing customers which is what will be discussed in more detail in the next following sections.

9 | Organization

To understand how the success MongoDB could come to pass we have to take a closer look inside at how the gears were, and are, turning. This section will look at the organisational structure of the company, what tools they use to organise development both internally and externally towards outside contributors, and finally how the developer forums where the main interactions between MongoDB and its community go down.

9.1. The Company

Some of the big names in MongoDB has already been mentioned, namely Dwight Merriman and Eliot Horowitz. As of today, they both still work at MongoDB as Co-founders, and Horowitz is also the CTO of the company. Several of the other top employees at MongoDB is listed on their website under the leadership section [96], including also previously mentioned CEO Dev Ittycheria. Since the beginning, the company has grown from a small team of 4-5 developers to having over 1800 employees. The vision of the original team was always to expand and make the product available to as many users and customers as possible, thus the need for great marketing teams, support teams, security teams and developer teams.

9.1.1. Development

Even though MongoDB is an open source project and is strong believers in the open source and free software movements, the way their products are developed are rather unusual for an open source project, at least from what we usually think of as open source. MongoDB has a more closed approach than other projects by having most of the development of their main products, like the core database, done by employees of the company. Ittycheria himself has, in fact, stated that MongoDB is not like a traditional open source project because their rationale for going open source was never about getting a lot of help and contributions from the community. It was to "drive adoption" [97]. By being completely transparent as well as providing a quality product, MongoDB views their community not as contributors but as users. He compares MongoDB to other projects which larger companies have open sourced some of their systems under a very permissive licence to create developer engagement. This is possible for companies where the software they release is not the core of their business. The examples used are Yahoo's Hadoop and Facebook's Cassandra, both also being database systems.

> *[...] MongoDB was built by MongoDB. [...] We open sourced as a freemium strategy; to drive adoption.* - Dev Ittycheria [97]

A freemium strategy is to provide the product for free but to require payment for certain extra features or support [98].

MongoDB has however still benefited from community contributions. Seeking to become a pure general-purpose product, the database had to work with a wide set of tools and programming languages. Many of the drivers for integration with different programming languages were originally developed by the community. By focusing on developing a very attractive core product, once the community found MongoDB to be the most compelling database on the market, the developer power at the ready to develop drivers for the newest popular framework is really what drove their wide adaption. Horowitz has stated that they never had the time to even consider other competitors in the market because they were so focused on making other developers successful and productive [89].

9.1.2. Products

The current product farm of MongoDB is huge [99]. It has grown along with the company and is the result of analysing the customer's needs. Some of these products are inspired by community developed products which are deprecated or lacked core functionality.

MongoDB Community Edition

Their primary product is, of course, the open source database itself which is free to download and use. This version of the database is called the community edition and has all the core features of the MongoDB platform.

MongoDB Enterprise Edition

Next, we have the enterprise edition of MongoDB. It is built on top of the community edition with extra layers of features and security which can be tailored to each company's needs. 24/7 support services also come with the enterprise edition. Some of the security features include a role-based privileges system and Kerberos authentication.

Other Software Tools

In addition to these two editions of the database, other tools such as Charts and Compass are built to provide visualisations of your data and a GUI to interact with it.

Cloud Services

MongoDB offers a bunch of cloud services as well which integrates with large cloud providers like Amazon, Google and Microsoft. Their service called Atlas is a pay-as-you-go service where you can scale the database to suit your needs and only pay for what you actually use. Another service called Stitch allows MongoDB queries to be written on the frontend, execute certain events in response to actions in the web application and lets you interact with your data in Atlas through the client application code. The much anticipated MongoDB Realm is also on the horizon which will combine Stitch with the newly acquired Realm and make data synchronisation across multiple platforms easy.

9.1.3. Open Source Licence

Before October 2018, MongoDB has been releasing its products under the GNU Affero General Public License V3.0 (AGPL) [100]. This licence is published by the free software

foundation and is of the strong copyleft type. It is very similar to the normal GPL licence with one key addition that handles modified code that is run on a server but never distributed.

> *[...] suppose the program is mainly useful on servers. When (developer) D modifies the program, he might very likely run it on his own server and never release copies. Then you would never get a copy of the source code of his version, so you would never have the chance to include his changes in your version. [...] Using the GNU Affero GPL avoids that outcome. If D runs his version on a server that everyone can use, you too can use it. Assuming he has followed the license requirement to let the server's users download the source code of his version, you can do so, and then you can incorporate his changes into your version. [...]* - The Free Software Foundation [100]

As MongoDB, and many of its users, provide services that are in fact server based like this, the AGPL allows transparency in all potential version of the MongoDB database system as well as the ability for those who wish to download the source code and build their own version.

On October 16th 2018, MongoDB decided to abandon the AGPL licence in favour of their own licence; the Server Side Public License (SSPL) [101]. The reason is according to MongoDB based on the importance of open development and that the AGPL is not specific enough regarding vendors who provide software as a service, and that large cloud vendors are testing the limits of the AGPL by trying to monetise on all the value of open source projects like MongoDB without giving back to the communities.

> *Rather than litigating this issue in the courts, we are issuing a new license to eliminate any confusion about the specific conditions of offering a publicly available MongoDB as a service. This change is also designed to make sure that companies who do run a publicly available MongoDB as a service, or any software subject to the SSPL, are giving back to the community.* - Server Side Public License FAQ [102]

This new licence is based on the GPL, but is issued by MongoDB and has not yet been approved by the Open Source Initiative. In fact, MongoDB decided to withdraw the SSPL from the OSI review process, which has sparked a lot of discussion of the definition of open source. In a public email, Horowitz stated that it was out of respect for the time and resources of the OSI board and the rest of the free software and open source community.

> *We continue to believe that the SSPL complies with the Open Source Definition and the four essential software freedoms. However, based on its reception by the members of this list and the greater open source community, the community consensus required to support OSI approval does not currently appear to exist regarding the copyleft provision of SSPL. Thus, to be respectful of the time and efforts of the OSI board and this list's members, we are hereby withdrawing the SSPL from OSI consideration. [...] In the meantime, current and future versions of MongoDB Community will continue to be offered under the SSPL. Over the coming days, we will update the messaging on our website to make it clear that the SSPL has not been approved under the OSI's definition of "open source." However, MongoDB remains free to use and source available under the SSPL, meaning users are free to review, modify, and distribute the software or*

redistribute modifications to the software in compliance with the license. - Eliot Horowitz [103]

This exception is from the mail thread exchange between MongoDB, and OSI and its community from march 2019. The discussion goes on for several threads, each weighing in both in favour and opposition to the licence. The discussion boils down to the need for extension of the term strong copyleft. As of today, the licence is still not approved but it has not been resubmitted either [102].

9.2. The Community

Even though most of the development seems to be conducted by employees at MongoDB, they do have a community of developers that contribute to the code base from all around the world. Since MongoDB as a company has grown so large in the last years and began offering several other products in addition to just the database, the community has also spilt following each different branch of the MongoDB products. To organize the community, there is a code of conduct that must be followed if you as a developer wish to contribute to either of the projects.

9.2.1. Code of Conduct

MongoDB's code of conduct is available on their website and are neatly presented as two-word sentences that can be elaborated on by simply opening an accordion element associated with that two-word sentence. What is important to note is this code of conduct who the targeted readers are, which are not simply external developers. MongoDB's code of conduct is for everyone to follow which includes employees, customers, partners and leaders as well as those who make up the "external" part of the community. I have put the word external in quotation marks because MongoDB strives for their community to be one as every community members participation is validated and appreciated regardless of being an employee or external developer. The code of conduct is split into two parts, one for behaviour in general and one specifically regarding the use of the forums. A question asked should be specific and strive to not go off-topic. They specifically state that the community consists of many volunteers and that answers to a question might take some time, so it is important to be patient. They encourage to share experiences on the public forums instead of closed email or other sources so that people can learn from the successes and failures of others. And finally not to spam the forums with links that violate their terms and conditions [104].

9.2.2. Key Contributors

From the original release in 2009, the MongoDB ecosystem has grown quite substantially. I asked Horowitz if he could name any key persons in the community. To his knowledge, there were not singular people that stood out in the community.

> *No individual person, I think it's really been a mix of things. [...] The core database has not a massive amount of individual contributors working on it. Which makes sense, databases are not something people just go in an goof off on in the weekends, because they are big and complicated and messy.* - Eliot Horowitz A.2

He then states that where the community thrives are all the areas around the core database. This includes things such as drivers for different languages, or connectors to services like Kafka. This statement is justified by the fact that most contributors to the core database project on GitHub, as well as some of the most used drivers, are in fact Mongo employees [105]. The development of more niche drivers and tools done by the community in the MongoDB ecosystem seems to be one of the company's greatest strengths. By having MongoDB be a choice in almost every developer scenario allows very broad adoption.

9.2.3. Applications Built by the Community

As MongoDB has grown through the years, more and more of the community developed drivers has been taken over by the company and developed by them. They are still open source, and contributions are welcome and MongoDB can confidently put their name on the quality of the product. Today, almost every driver for commonly used languages are officially developed by MongoDB and the more obscure ones remain developed by the community. However, they are linked through the official MongoDB website but MongoDB makes sure to specify that they are not supported officially in any means [106].

In addition to drivers, there are tools built in the MongoDB ecosystem to make certain tasks easier or more approachable for developers. Mongoose is such a tool meant to make it easier for developers to model their application data into something that can be saved in MongoDB. Robo 3T (formerly Robomongo) is another such tool which provides the users with a GUI to interact with their MongoDB database and perform simple as well as advanced operations without having to write everything in a terminal.

9.3. Forums

As mentioned in the code of conduct, it is encouraged to share experiences with the products, not only from a developer perspective but also from a product owner or product manager perspective to evaluate if MongoDB is the right solution for their business or start-up. In other words, there are parts of the forums dedicated to several levels of technical competence. To narrow this down a bit, one can see the forums as containing two groups of people, those who use MongoDB as a product in their own business or project and those who help develop the product that is MongoDB.

9.3.1. User Forums

In the user forums, one can see the code of conduct shining through as people are usually very friendly in their phrasing, both the person asking as well as the respondents.

After digging through both of the forums, I have found a pattern that new people in the community tends to introduce themselves with a post stating who they are and what their experience with MongoDB is. This is not written in the code of conduct as some mandatory step but is simply a norm. An example from the newest *welcome*-post at the time of writing is as follows. The names of the developers have been anonymised in this thesis because I have not gotten their explicit consent to use their name, even though it is currently available on the public community forums at the time of writing.

> *Hi all, my name is Developer#1 and I've been working with MongoDB off and on since the 1.8 days. In my current role, I don't do much database work, but I*

do act as an internal consultant for teams running MongoDB and Cassandra.

Fun facts:

- *I spent two years working with the education team as a TA for several of the online University courses.*
- *I was a part of the MongoDB Masters program.*
- *I was awarded the first MongoDB Administrator's Certification.*
- *I'm a Colorado native and have never been downhill skiing*
- *I hope to share what knowledge I can and look forward to learning from everyone.*

I hope to share what knowledge I can and look forward to learning from everyone.

This kind of posts falls under a tag in the forums called *welcome*, and every single post gets responses from other developers in the community wishing them welcome, pointing them in the right direction where their expertise can be most useful, and some of the existing developers might actually be looking for this particular person as they have the exact experience they are looking for, thrilled to see them join. Some of those who write these welcome-posts are returning developers, as is the case of Developer#1 presented above. This sometimes leads to old colleagues or students/teachers getting in contact. The comment below describes this.

Hi Developer#1, I remember when you were a teaching assistant on one of the MongoDB introductory courses that I took. Maybe more than one? I'm still a MongoDB user. And I really ought to sign up for additional classes soon – and I need to finish the aggregation course I had started long ago but somehow paused.

Thanks so much

Developer#2

The forums have high activity levels and most questions get answers or comments within a few days, either as answers to solutions or pointing the questioner to another post where that same question has been answered before [107].

9.3.2. Developer Forums

As mentioned, there is another forum primarily for those who wish to contribute by developing the products themselves. To be able to contribute any developer must first sign a contributor's agreement. I am no lawyer, but to my understanding, in short, you allow MongoDB to claim the rights of the work you do, being adding new code or modifying existing code in the repositories. It also makes sure that the code you write and contribute is not owned by some other third parties which can become a legal issue. After the agreement has been approved, you can register for an account that lets you access the MongoDB Jira board. Jira is a tool to track the progress of a project through rows and columns with small Post-It-like notes on them containing details of a specific part of the system to be implemented or fixed. This is the primary place for developers to interact as they can pick an already existing issue or feature, discuss possible solutions and, once ready, submit pull requests connected to a specific Jira task [108].

Some also tend to use a Google Groups forum to submit technical questions that are not necessarily suggesting improvements or changes, but pointing out details in the code that are not easy to understand at first glance. Occasional improvements suggestions or bug reports do exist in this forum, but the authors are encouraged to take it to the Jira board and submit tickets there. In addition to these technical questions, the official release notes of different builds of the core server are also published in this forum.

To try and centralise all of the community activity, being it usage related or development-related questions, MongoDB released their own community forum on their developer hub platform. A single hub for all things related to developing, learning, sharing and discussing MongoDB [109]. The developer hub itself launched in march of 2020.

> *The Developer Hub is the place to learn about all things MongoDB* [109]

9.3.3. MongoDB World

MongoDB's size and broad usage have also inspired the company to host their own annual conference with keynotes and workshops focused on MongoDB. This is also the arena where the company reveal big announcements like acquisitions or partnerships, new products or other announcements worthy news like new leadership. In 2019 MongoDB World had over 2000 attendees, over 100 talks, 200 Ask the Experts sessions and a Hackathon with 900 participants from around the world. MongoDB World is not only for the company to show off their products, but also an arena for developers to meet and discuss their experiences with MongoDB related development. This allows for community members that usually talk over the forums to have an excuse to meet up in person. As a result of the COVID-19 pandemic, many such events as MongoDB World has been cancelled or postponed to next year. However, MongoDB has been quick and organised a virtual alternative called MongoDB .Live held later in June [110].

9.4. Community Acknowledgements

MongoDB has many ways of engaging their community in different activities as well as having several different achievements which individual can attain.

9.4.1. Community Badges

One of such ways is community badges. In order to use the MongoDB online forums, you have to register an account which identifies you when you interact by asking or answering questions and commenting. These profiles are visible to all who also has got an account registered on the forums. The community badges one has obtained is shown here. Some badges are very long hanging fruit to engage people from the very beginning. There are badges for receiving your first like on a post you have written, for sharing your first post, and for simply filling out your profile information just to name a few. There is even an exclusive Star Wars badge for people who logged in on the fourth of May, which is international Star Wars day. At the other end of the scale, there are badges which are much harder to obtain and which are used to identify the most active users on the forum. One badge requires you to visit the forums every single day for a year, another one requires you to have minimum 5 likes on over 300 posts. In addition to obtainable badges, MongoDB staff have their own badge to highlight the presence of MongoDB employees among the rest of the community [111].

9.4.2. Certifications

The badges obtained on the community forums are only visible there as well. However, MongoDB offers several educational programs which aim to provide developers with the skills of what they consider a certified MongoDB professional developer. As proof of their competence, developers can take certification exams which, if passed, gives them the right to use certification badges on other sites than the MongoDB forums. These certifications are meant to help developers land jobs in startups and larger companies which already use MongoDB or plan to switch over to it and require skilled engineers who know the ins and outs of MongoDB. They provide both developer certificates which focuses on integrating existing software and MongoDB, and database administrator certificates who focuses on maintaining and scaling the database to suit the company needs [112].

9.4.3. MongoDB University

As a measure to prepare developers and administrators for these certificates, MongoDB has launched MongoDB University which is a free educational platform for anyone who wants to be proficient with MongoDB. Several courses are ranging from very beginner-friendly introductory courses all the way to advanced topics like database security. Over 1.5 million people have registered from over 196 countries [113].

9.4.4. Innovation Award

As part of MongoDB World (and now .Live), annual innovation awards are being handed out. The award can be won by any individual, group or larger organisation who has built or is building something with innovative potential. Among the prizes are an actual, physical award, recognition at the global MongoDB World (or .Live), and a featured story on the project on the MongoDB blog. The innovation award spans over several different categories to recognise all kinds of accomplishments reached with the help of MongoDB, ranging from honouring a single certified professional who has done a great deal for the community, to companies who has a significant increase in their revenue thanks to MongoDB [114].

Part IV.

Analysis

10 | Discussion

Open source has indeed made a huge impact in the tech industry in the last few decades with projects ranging from operating system distributions to electric car batteries. Lots of research has been conducted on the topics of licences, developer motivations and business models [43] [10]. The open source model is known for producing highly innovative products that are high quality and thoroughly reviewed and improved by a large community of peers [16] [17]. The innovative power is intriguing to many who wish to start a business developing a software product.

Often associated with being free of charge, many believe that open source is not a sustainable way to run a company from a financial perspective. To combat this scepticism, many business models for open source development has been tried and found relatively effective. With platforms for crowdfunding, making donations to projects, recurring or one time, are also becoming much easier. These platforms allow developers and other customers to donate to the projects they see value in and care about [73] [54].

Many projects rely on a large community of contributors to keep the project alive and attractive in the market. Over time, as projects age, the is always the risk of contributors losing interest in the projects. In the modern open source world, it is just as much about keeping projects alive as publishing them in the first place. The excitement around a product can be short-lived if not nurtured properly. There are many studies conducted on the incentives and motivations behind contributors, such as community renown and in some explicit rewards [78] [72].

In light of the cathedral and the bazaar models of developing software, one could argue that a more cathedral-like approach greatly reduces the risk of project death by contributor loss because the project is mainly developed by a team of developers high up in a hierarchy. By managing to build a company with sustainable income around such a group of developers, the project is more likely to sustain at the expense of having rapid growth through outside developer power [5] [93].

By adopting a model that is somewhat in between the cathedral and the bazaar, as well as finding the right incentives for growing a community mainly consisting of users and reviewers before contributors, companies could arguably gain market advantage through rapid development, financial gain and adaption. In this chapter, I will discuss my observations of MongoDB and see if growth has been a product of adopting such a model, and see if previous research on developer motivation and incentives also apply to a community of users instead of contributors.

10.1. MongoDB's "Itch"

Writing databases is as Horowitz has stated not an easy task and can quickly become quite complex, thus requiring special skills, talent and experience. Horowitz and Merriman both fulfilled these requirements as developers even before the work on MongoDB began. Regarding the lesson from Raymond [5] that every piece of great software begins as a developer's itch, it is certainly true in the case of MongoDB as seen by great continuous success. Not only were Horowitz and Merriman scratching their own itch by solving their own big data issues, but they were reaching out to scratch an itch in the market as well [5].

Raymond [5] also points out the value in reusing previously written code. Just like the approach that the GNU/Linux project did by replacing parts of a complete OS one by one with free software components, MongoDB took a lot of its inspiration from the already powerful relational database technologies. Instead of reinventing everything from scratch, they simply made changes to the main parts of relational databases that need to be improved; easier data modelling for developers and scaling horizontally. This implies that great ideas can come from taking existing solutions and find ways to improve it instead of reinventing the wheel [89]. Providing source code can speed up this process even further as the cogs and gears of a program can be studied in great detail by the community [4].

The original MongoDB team were seemingly not salespersons or great at marketing. All they did was write about their project on their blogs and talk out it at developer meetups. However, the word of mouth spreads exponentially. Being open source allows developers to try it out without having to pay or apply for a trial version. Even more impactful is the "mouth" of someone famous. Many companies were looking for an easier way to store data, preferably in a document-based manner just like MongoDB [92]. The popular open source hub SourceForge was one of the first stakeholders in MongoDB and gave a great review of a version of MongoDB that was not even at release yet.

MongoDB was arguably at the right place at the right time, as the new generation of developers that came around in the early 2010s were interested in rapid development and frequent releases. With relational databases, as applications grow the complexity usually grows with it and the relations becomes very hard to maintain. A document-based database like MongoDB was answering the call of these developers for a simpler solution. Even though MongoDB might have been lucky releasing at the right time, they are still growing larger and larger even today, so something else has to have driven their popularity as well. After all, the hardest part of open source seems to be sustainability [4].

10.2. Community Engagement

Research has shown that the interaction between community and project owners is the quintessential part of open source [78] [43]. Usually, when talking about open source communities, the term refers to contributors who are directly engaged in projects development by contributing code. In the MongoDB camp, they seem to view the term differently as they see their community mostly as users rather than contributors. They see them as a mean drive growth by adoption and word of mouth. When the community has this different role, how does that affect their motivation and what incentives, if any, do they expect from MongoDB?

10.2.1. Mobilising

Mobilising the community early one was something Horowitz and Merriman seemed to excel at. Driven by pure excitement they were showing their product to everyone they knew. They attended hackathons showing off what their product could do to hacker communities. Since NoSQL technologies were not that common at the time, it could seem that there was ambiguity around exactly what problems MongoDB could solve. But the topic on everybody's lips was how to tackle big data and people became quite interested when MongoDB was described to be "easy to use and setup, and great at scaling horizontally". Since it was written with JavaScript like notation it was also labelled as JavaScript for databases. It is clear by the lengths they went to advertise and share their product, no sign of the "field of dreams" was present [74].

10.2.2. First Contact

Many developers dislike being flamed on public forums for asking "stupid questions" or not abiding by the norms of that particular community forums [4]. MongoDB seeks to tackle this issue with several means. One is their code of conduct which encourages nice attitudes towards other community members, as well as serving as a written version of the norms of the forum. Being precise when asking questions and providing enough details being a few of those norms. The new developer forum seems to have many ways of encouraging new community members to be active participants. New *welcome* posts are still popping up in the forum every day. With this, MongoDB has lowered to bar for community participation which appeals to developers need to belong [76].

Another factor that was present in the earlier days of MongoDB was that Horowitz was tearing down the hierarchy in the community. He went to great lengths to answer every single question on the forums, and when the consumer and the CTO communicate directly like this the consumer feels heard and valued as an important part of the community [81]. He was even able to talk to me in his otherwise busy day.

10.2.3. Gifting Culture

In traditional projects where contributions are welcome, the gifting culture is much more visible. By abiding by the bazaar model, the developers set expectations for the community to give something back to the project. In these cases, the gift of code is returned by the gift of code (or documentation etc.). The point is that the gift is given and the one received is at seemingly equal value, or at least exists in the same domain [10].

By closing off the project for contributions, MongoDB sends the signal that they do not want anything in return. Instead, they just want the community to reap the benefits of using an open source tool that is easy to just start playing with right out of the box without having to worry about licensing and other legal issues. However, there is no such thing as a free lunch, and MongoDB does, in fact, seem to want something in return but it is not something that the individual user can give. They want broad adaption, they want a reputation, they want great reviews and a solid word of mouth. By using the community as advocates MongoDB has become the first choice of many developers needing to choose a database [10] [83].

A solid adaption not only grows MongoDB as a single product but draws the database market's focus towards the realm of NoSQL in favour of other technologies. It allows

projects like MongoDB to control the technology scene. And a bigger market means more potential sales. As an example, when Tesla open sourced their patents, more cars can use similar batteries and require similar chargers. This allows Tesla to have other companies build chargers that are compatible with their cars as well, saving a lot of time and resources that would have been spent providing charging stations all over the globe [115].

10.2.4. Users over Contributors

Even though MongoDB does not seek contributions as gifts, they still value their product adaption very highly. They believe that through engagement and simply reaching many peoples lips their product will always be in circulation among options for databases when building software. The recent aggregation of their developer and user forums into the developer hub have in essence made the potential number of users that can interact with each other the union of all the forums. This means that people who usually stuck to one of the forums can discuss and interact with the people who stuck to one of the other forums, thus extending the network of possible interactions [10] [107].

This aggregation also means that MongoDB can better monitor the feedback the community provide. Their vision has always been aligned with the agile principles of appealing to the customer's needs and really taking in their feedback [62].

10.2.5. Virtual Incentives with Value

MongoDB provides its users with many forms of virtual achievements and other incentives of value. Their badge system has low tier badges which are easy to obtain as well as higher-tier badges which are harder to obtain. They can serve the purpose of satisfying personal accomplishment as well as renown in the community. It seems that it also satisfies the developers need to belong as merit systems tend to encourage too more interactions in the forums [76] [78].

Perhaps the most valuable obtainable asset in the community is the professional certifications which can be taken through MongoDB University. These certificates serve as proof that developers have the skills needed to handle MongoDB implementations and management. By taking a glance at the curriculum it does not seem to take a very long time to complete. This is also a means of including the community since the sense of accomplishment with MongoDB technologies are low hanging fruit [43].

I do not have data which can verify the existence of monetary incentives driving contributions to MongoDB but due to its size and wide adaption among larger companies, it seems possible that they exist.

10.3. The MongoDB Ecosystem

The idea of having the core product more restrictive towards accepting contributions seems to have lead to the community seeking expiration in applications around the product, thus expanding the platform ecosystem.

10.3.1. Focus on Platform

This seems to be a deliberate choice by MongoDB as driving the idea of general-purpose requires a solid collection of interfaces, in this case, drivers. In the modern software market, single products can no longer gain a competitive advantage. Platform ecosystems enable customer lock-ins which again leads to wider adaption and even more customer lock-ins [71]. By growing a platform ecosystem thus their product's general-purpose, smaller micro-segments of markets can be penetrated which are usually inaccessible by companies who solely provide single products or services. MongoDB has by doing this been able to capture the long-tail; the niche markets.

10.3.2. Rapid Development

Today the company has over 1900 employees and most of the drivers are now being officially developed by employees at MongoDB. Before the company was this large, the community was still huge due to the adaption of MongoDB grew faster than they managed to hire more employees. The sheer size of the community has enabled extremely rapid development of new drivers once certain technologies suddenly become the new craze in the tech world. This was the case with Node.js, where a community developed driver was released just a few days after the launch of Node.js [89].

Being omniscient in an ever-changing tech world is close to impossible but having a community look at problems, and solutions, from any possible angle, can help you as a company see things that you would otherwise never discover. This seems to be the case when MongoDB acquired the WiredTiger engine. Since MongoDB was open source and transparent, the folks at WiredTiger downloaded the code and built their own system into MongoDB, sent it back to Horowitz and the rest of the team and asked for their evaluation. This event led to MongoDB acquiring WiredTiger without MongoDB having to lift a finger. Open source allows smaller startups like WiredTiger to perform actions like these in pursuit of being bought up, which is a quite common goal in the entrepreneur world [89].

10.3.3. Acquiring Community Projects

Many of the drivers for MongoDB has been built by the community. However, as the company has grown over the years more and more of these drivers have been fused into the company from the community. More and more of the drivers are now *official* drivers, meaning that MongoDB employees are developing them. Some of them simply got an official MongoDB sticker smacked onto them claiming them as they were, others had to be rewritten from scratch to abide with the driver guidelines developed by the MongoDB staff. By issuing a guideline for how every single driver should behave, they strive to make MongoDB a familiar, seamless experience regardless of what programming language is being used.

The community's reaction to these fusions has varied but more than often been positive. They are still open sourced under the same licence as before, so contributions are still welcome just as they were when the drivers were still community-based.

10.4. Business Model

MongoDB seems to have been flexible with their business model, utilising different strategies at different stages in the projects lifetime. Their current business model seems to be something that not many other competitors apply, so it might be a key to their current success and continuous growth.

10.4.1. Project Success

MongoDB has satisfied many of the factors of a successful IT project [47]. Horowitz and Merriman were both skilled developers with great motivation for their project, and going to the lengths of answering every question on the forums as well as being present when demonstrating the product has built superior customer relationships. MongoDB is very agile and adapts very easily to the tech market, and listens to their customers' needs. Their product is pursuing a simple design, is regularly delivered with frequent releases, and they certainly developed the most important features first; their core database.

As stated by Linus Torvalds, modularity is the key to open source. MongoDB has been very modular with their products allowing easy expansion of tools in the ecosystem. The core database might be complex because of the very nature of databases but since their focus never was to have developers contribute that much to that repository, complexity is not an issue as an intrinsic cue [68].

Regarding the extrinsic cues, MongoDB used the AGPL licence for a long time which incorporates strong copyleft. These types of licences are greatly supported in the free software camp as well as open source, allowing the choice of licence to be a string extrinsic cue. They did, however, change it to a non-OSI approved licence which has had its amounts of consequences (see section 10.5). The user base and developer base are both huge and has been one of the main focus areas of MongoDB, expanding these bases. It is safe to say that MongoDB has satisfied a lot of indicators of project success, technical as well as market success.

10.4.2. Multiple Value Propositions

Stated many times already, finding the ultimate business model for open source projects can be hard. That said, even though such a method would be discovered, adopting the business model to the state of the project is key to its economic growth. Horowitz has stated that in the early phases of development, they were developing something they wanted, or at least thought they wanted, as it was quite experimental. They were not sure if anyone else wanted their software. MongoDB can be seen as a NoSQL movement more than a market, as it seems they were not sure if their product would even get to a release candidate outside of beta [89].

Today, MongoDB provides a collection of different solutions to provide its customers with the experience they want. The community edition of the database is, of course, free and fully open source for developers to download and use freely. They also provide the enterprise edition under a commercial licence built on top of the community edition with several extra proprietary features and tools. This allows MongoDB to keep a low and attractive price point because the core of their product is still free of charge. In addition to the features, they also provide support either from a distance or by sending employees over

to help customers everything up and facilitate the implementation phase. Companies find this to be very competitive versus other proprietary vendor's high price tags. This business model has been applied by many other open source companies and in itself does not provide the company with the profit it needs to grow as MongoDB has done. However, MongoDB has heavily invested in its cloud platform Atlas which allows customers to outsource all of the work of setting up the database and managing it to MongoDB. All they have to do is provide the data they want to store and how they wish to organise it. Their cloud service allows customers to interact with their product without ever having to speak to MongoDB employees, which is desired in some cases [89] [71].

By having both these models, MongoDB appeals to both the larger enterprises with more complex systems that require the help from trained support staff and the smaller companies that just need something up and running fast to test their proof of concept through the very low price point cloud services.

10.5. Choice of Licence

MongoDB's switch from the AGPL licence to their own SSPL licence has got a lot of attention from the open source community [103]. They tried to get both the first version and the second version of the licence approved by OSI but the first one was rejected and MongoDB withdrew the second one from the review process. This second move was arguable because the process was taking up a lot of time and resources from OSI but that MongoDB also meant that the whole review process is faulty.

10.5.1. Debate

The controversy of the SSPL licence has sparked a debate in the open source community [116]. The purpose of the licence is to hinder corporate giants to monetise on MongoDB's technology without having to pay MongoDB anything, thus potentially "stealing" profits from MongoDB. However, this can also impact smaller startups. If someone running a much smaller company or start upbuilds something really cool and that their product heavily relies on MongoDB as a database, then eventually MongoDB could come knocking at the door and demand that they buy a commercial licence of MongoDB or open source their own product. The community sees this as an issue as open sourcing your product might rob them of revenue from proprietary sales. However, the purpose of this thesis is to find ways for open source to be the most lucrative it can be. In the future we might see the SSPL licence become open source compliant according to OSI, or that OSI verification will no longer have the same status in the community.

10.6. Refining Commercial Open Source

The SSPL seems to be another step MongoDB has taken in the right direction towards refining commercial open source. In the early days of GNU and Linux, open source was an anomaly. A small thing that no one really understood and acknowledged as a commercial opportunity. Today open source is at the heart of the most popular mobile system in the world [12]. The main challenge that commercial open source faces today seems to be finding the sweet spot in between a closed project and a completely open project. How wide should the doors be opened for the community to feel that their contributions and feedback matter while you as a company still remain in control? MongoDB has done many things right.

By building their ecosystem of very open projects (drivers and third-party tools) around a core that is more closed, the excitement and passion for the product are exercised by expanding the platform ecosystem. The more diverse ecosystem, the more general-purpose the product becomes which is key to drive adaption along with mobilisation tactics early on in the project's life. Regarding their business model, MongoDB's approach of combining several different value propositions allows them to compete in multiple markets at the same time and provide an attractive alternative in all of them. Being too focused on applying only one model seems to be an economic bottleneck in other projects.

11 | Conclusion

The problem definition of this thesis seeks to answer how MongoDB has grown into a profitable company through their open source products. The answers from the findings are not clear but can imply where the key to their financial success might lie, which is mainly in three areas.

By having their core product free of charge, they have eliminated the high price tag that usually comes with proprietary software. This is a common characteristic of open source. On top of this, they have utilised several business models, both as SaaS through the cloud as well as more tailored solutions for larger enterprises. In this way, they offer their services at very attractive price points and two different markets.

By having the doors to their core product more closed off for contributions which reminiscent of a cathedral style approach with slightly open doors, MongoDB's has directed their community's skills and itches towards building an ecosystem around the core software. This has allowed them to become a viable solution in many different scenarios, in other words, *general purpose*. It is not to shove under a rock that the founders of MongoDB were skilled developers and managed to build a product of great quality early on but their growth is a consequence of their involvement with the community and listening to their users and customers, providing them with the features they want. It seems that MongoDB has utilised the community formed by being open source to drive interest and adoption of their product.

Their change of licence to one that is not OSI-approved has gotten a lot of attention in the open source community. It has sparked a debate around whether the open source definition is not adapted to the modern software market. The licence change directly affects MongoDB's potential profit and is therefore crucial to their continuing growth. It will be interesting to see the consequences of this debate in the future. If it is recognised by the community and OSI in the end, it would be a step in the direction of refining commercial open source.

11.1. Limitations of the Study

As presented in the method chapter, this thesis has been tossed in turned in many directions before I landed on the topic that I did. To gather more data from the community, I would like to have interviewed some random contributors and community members to hear their reflection on why they chose to join the MongoDB community over another community. The topic of the thesis becomes closer to marketing than development so more theory on branding and growth hacking could have been useful.

11.2. Further Work

The purpose of this thesis was to shed light on the way MongoDB rose to success by utilising open source in the hope that they had found the key to commercial open source. Many of the findings can be argued to be too surface level and ambiguous but the essence of their business model is clear. It would be interesting to see more research on this model used elsewhere, in projects where it already exists and to try it out on new projects. In addition, the debate of what is pure in the open source community is still unresolved and it would be exciting to read a study that focuses solely on this debate.

Bibliography

[1] N. Friedman, "The State of Open Source - GitHub Universe 2019 - YouTube." [Online]. Available: https://www.youtube.com/watch?v=jImkk1BxGa4

[2] M. Priyadarshini, "The Top & Fastest Growing Open Source Projects On GitHub In 2019," Fossbytes, Tech. Rep., 2019. [Online]. Available: https://fossbytes.com/top-fastest-growing-open-source-projects-on-github-in-2019/

[3] Red Hat Inc, "The state of Enterprise Open Source: A Red Hat report," Red Hat Inc, Tech. Rep., 2019. [Online]. Available: https://www.redhat.com/en/enterprise-open-source-report/2019

[4] B. Fitzgerald, "The Transformation of Open Source Software," MIS Quarterly, Tech. Rep. 3, 2006.

[5] E. Raymond, "The cathedral and the bazaar," *First Monday*, vol. 2, no. SPEC, pp. 23–49, 10 2005.

[6] J. Sanders, "Why open source software adoption is accelerating in the enterprise," *TechRepublic*, 2019. [Online]. Available: https://www.techrepublic.com/article/why-open-source-software-adoption-is-accelerating-in-the-enterprise/

[7] Ben Bromhead, "10 advantages of open source for the enterprise," OpenSource.com, Tech. Rep., 2017. [Online]. Available: https://opensource.com/article/17/8/enterprise-open-source-advantages

[8] J.T.S. Moore, "Revolution OS - Documentary," 2002. [Online]. Available: http://www.revolution-os.com/index.html

[9] The Linux Foundation, "Using Open Source Software to Speed Development and Gain Business Advantage," *The Linux Foundation*, 2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/02/using-open-source-software-to-speed-development-and-gain-business-advantage/

[10] M. Bergquist and J. Ljungberg, "The power of gifts: organizing social relationships in open source communities," *Information Systems Journal*, no. 11, pp. 305–320, 2001.

[11] "6 pivotal moments in open source history | Opensource.com." [Online]. Available: https://opensource.com/article/18/2/pivotal-moments-history-open-source

[12] Hostingtribunal, "111+ Linux Statistics and Facts - Linux Rocks!" 2019. [Online]. Available: https://hostingtribunal.com/blog/linux-statistics/#gref

[13] Massachusetts Institute of Technology, "MIT OpenCourseWare | Free Online Course Materials," 2020. [Online]. Available: https://ocw.mit.edu/index.htm

[14] J. Stern, "How open source software is fighting COVID-19," *OpenSource.com*, 2020. [Online]. Available: https://opensource.com/article/20/3/open-source-software-covid19

[15] GitHub, "Keynote - GitHub Satellite 2019 - YouTube," 2019. [Online]. Available: https://www.youtube.com/watch?v=sGC2rwOiaWc

[16] J. Wallen, "10 open source projects that are leading innovation," 2013. [Online]. Available: https://www.techrepublic.com/blog/10-things/10-open-source-projects-that-are-leading-innovation/

[17] ——, "Best open source projects of the year," *TechRepublic*, 2019. [Online]. Available: https://www.techrepublic.com/article/the-best-open-source-innovations-of-the-last-decade/

[18] Google, "Why Open Source?" 2020. [Online]. Available: https://opensource.google/docs/why/

[19] A. Howden, "On the value of open source," 2018. [Online]. Available: https://medium.com/sitewards/on-the-value-of-open-source-6c3075837a9f

[20] PR Newswire, "MongoDB, Inc. Announces Fourth Quarter and Full Year Fiscal 2019 Financial Results," 2019. [Online]. Available: https://www.prnewswire.com/news-releases/mongodb-inc-announces-fourth-quarter-and-full-year-fiscal-2019-financial-results-300811992.html

[21] D. Bretthauer, "DigitalCommons@UConn Open Source Software: A History," *OpenCommonsLibrary*, 2001. [Online]. Available: https://opencommons.uconn.edu/libr_pubs/7http://digitalcommons.uconn.edu/libr_pubs/7

[22] R. Finkel, "What is an operating system?" in *Computer Science Handbook, Second Edition*, 2004, pp. 80–1. [Online]. Available: https://www.howtogeek.com/361572/what-is-an-operating-system/

[23] "The GNU Operating System and the Free Software Movement." [Online]. Available: https://www.gnu.org/

[24] Fossbytes, "Linus Torvalds's Famous Email — The First Linux Announcement," 2016. [Online]. Available: https://fossbytes.com/linus-torvaldss-famous-email-first-linux-announcement/

[25] Free Software Foundation, "The GNU Operating System and the Free Software Movement," 2020. [Online]. Available: https://www.gnu.org/

[26] ——, "What is Copyleft? - GNU Project - Free Software Foundation," 1998. [Online]. Available: https://www.gnu.org/licenses/copyleft.html

[27] Tech Terms, "Copyright Definition," p. 1, 2009. [Online]. Available: https://www.investopedia.com/terms/c/copyright.asp

[28] Eric S. Raymond, "Eric S. Raymond's Home Page," 2020. [Online]. Available: http://www.catb.org/~esr/

[29] J. Hoffmann, "The History of the Browser Wars: When Netscape Met Microsoft - The History of the Web," 2017. [Online]. Available: https://thehistoryoftheweb.com/browser-wars/

[30] B. Perens, "Bruce Perens," 2020. [Online]. Available: https://perens.com/

[31] OpenSource.com, "The Open Source Definition | Open Source Initiative," 2020. [Online]. Available: https://opensource.org/osd

[32] ——, "Open Source Initiative," 2020. [Online]. Available: https://opensource.org/

[33] ——, "History of the OSI | Open Source Initiative," 2020. [Online]. Available: https://opensource.org/history

[34] Apache, "The Apache HTTP Server Project," 2020. [Online]. Available: https://httpd.apache.org/

[35] PHP, "PHP: Hypertext Preprocessor," pp. 1704–1704, 2013. [Online]. Available: https://www.php.net/

[36] "GNOME – An easy and elegant way to use your computer," 2017. [Online]. Available: https://www.gnome.org/

[37] N. R. Budhathoki and C. Haythornthwaite, "Motivation for Open Collaboration: Crowd and Community Models and the Case of OpenStreetMap," *American Behavioral Scientist*, vol. 57, no. 5, pp. 548–575, 2013.

[38] S. Jr and C. Denner, "Attractiveness of free and open source software projects," 18th European Conference on Information Systems, Tech. Rep., 2010. [Online]. Available: http://www.eclipse.org/

[39] A. Hars and S. Ou, "Working for free? Motivations for participating in open-source projects," *International Journal of Electronic Commerce*, vol. 6, no. 3, pp. 25–39, 2002.

[40] A. Truong, "Netscape went public 20 years ago today—a look at how it catalyzed the dot-com boom and changed the world," 2015. [Online]. Available: https://qz.com/475279/netscape-changed-the-internet-and-the-world-when-it-went-public-20-years-ago-today/

[41] Mozilla Project, "History of the Mozilla Project," 2020. [Online]. Available: https://www.mozilla.org/en-US/about/history/

[42] Mozilla, "Grants — Mozilla," 2020. [Online]. Available: https://www.mozilla.org/en-US/grants/

[43] L. Dahlander and M. Magnusson, "How do Firms Make Use of Open Source Communities?" *Long Range Planning*, vol. 41, no. 6, pp. 629–649, 12 2008.

[44] T. Kilamo, I. Hammouda, T. Mikkonen, and T. Aaltonen, "From proprietary to open source - Growing an open source ecosystem," *Journal of Systems and Software*, vol. 85, no. 7, pp. 1467–1478, 7 2012.

[45] SourceForge, "SourceForge - About," 2020. [Online]. Available: https://sourceforge.net/about

[46] Google, "Android Website," 2020. [Online]. Available: https://www.android.com/

[47] H. Taherdoost and A. Keshavarzsaleh, "A Theoretical Review on IT Project Success / Failure Factors and," *4th International Conference on Telecommunications and Informatics, Sliema, Malta*, pp. 80–88, 2015.

[48] GitHub, "GitHub Sponsors," 2020. [Online]. Available: https://github.com/sponsors

[49] E. v. Hippel and G. v. Krogh, "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science," *Organization Science*, vol. 14, no. 2, pp. 209–223, 2003. [Online]. Available: http://pubsonline.informs.orghttp//www.informs.org

[50] E. Musk, "All Our Patent Are Belong To You," pp. 1–8, 2014. [Online]. Available: https://www.tesla.com/blog/all-our-patent-are-belong-you

[51] EV Database, "Newest and upcoming electric cars in 2019 and 2020 - EV Database," 2019. [Online]. Available: https://ev-database.uk/compare/newest-upcoming-electric-vehicle

[52] A. Nowogrodzki, "How to support open-source software and stay sane," pp. 133–134, 7 2019.

[53] R. Sen, S. S. Singh, and S. Borle, "Open source software success: Measures and analysis," *Decision Support Systems*, vol. 52, no. 2, pp. 364–372, 1 2012.

[54] S. Vujovic and J. P. Ulhøi, "An Organizational Perspective on Free and Open Source Software Development," in *The Economics of Open Source Software Development*, 2006, pp. 185–205.

[55] GitHub, "Choose an open source license," 2020. [Online]. Available: https://choosealicense.com/

[56] Free Software Foundation, "The GNU General Public License v3.0," 2007. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.html

[57] Sylvain Leroux, "Open Source Licenses Comparison Guide," 2019. [Online]. Available: https://itsfoss.com/open-source-licenses-explained/

[58] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile Software Development Methods: Review and Analysis," VTT, Tech. Rep., 2002. [Online]. Available: http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf.

[59] Techrepublic, "Understanding the pros and cons of the Waterfall Model of software development," 2006. [Online]. Available: https://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/

[60] D. W. W. Royce, "Managing the Development of large Software Systems," *Ieee Wescon*, no. August, pp. 1–9, 1970.

[61] H. Kniberg, *Scrum and XP from the Trenches*. InfoQ, 2007. [Online]. Available: http://old.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf

[62] K. Beck, M. Beedle, A. V. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," p. 2006, 2001. [Online]. Available: https://agilemanifesto.org/

[63] K. Beck, M. Beedle, and a. V. Bennekum, "Principles behind the agile manifesto," *Retrieved*, pp. 2–3, 2001. [Online]. Available: https://agilemanifesto.org/principles.html

[64] I. Jacobson, P. W. Ng, P. E. McMahon, I. Spence, and S. Lidman, "The essence of software engineering: The SEMAT kernel," *Communications of the ACM*, vol. 55, no. 12, pp. 42–49, 12 2012. [Online]. Available: https://dl.acm.org/doi/10.1145/2380656.2380670

[65] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 5 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7458761/

[66] J. Wang, "Survival factors for Free Open Source Software projects: A multi-stage perspective," *European Management Journal*, vol. 30, no. 4, pp. 352–371, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.emj.2012.03.001

[67] C. M. Schweik, R. English, S. Haire, and R. Conservation, "Factors Leading to Success or Abandonment of Open Source Commons An Empirical Analysis of Sourceforge.net Projects," *The European Journal for the Informatics Professional*, no. 43, pp. 58–65, 2009.

[68] V. Midha and P. Palvia, "Factors affecting the success of Open Source Software," *Journal of Systems and Software*, vol. 85, no. 4, pp. 895–905, 4 2012.

[69] P. Bratach, "Why Do Open Source Projects Fork?" 2017. [Online]. Available: https://thenewstack.io/open-source-projects-fork/

[70] S. O'Hear, "Nokia's Forking Of Android Could Benefit Google," 2014. [Online]. Available: https://techcrunch.com/2014/02/24/nandroid/

[71] A. Tiwana, *Platform Ecosystems: Aligning Architecture, Governance, and Strategy.* Elsevier Ltd, 2014.

[72] G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, "Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development," MIS Quarterly, Tech. Rep. 2, 2012.

[73] V. Hardy, "Why Do Developers Contribute to Open Source Projects?" 2013. [Online]. Available: http://www.apache.org/foundation/,http://www.websitemagazine.com/content/blogs/posts/archive/2013/08/29/why-developers-contribute-to-open-source-projects.aspx

[74] E. B. Swanson and N. C. Ramiller, "The Organizing Vision in Information Systems Innovation," *Organization Science*, vol. 8, no. 5, pp. 458–474, 10 1997. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/orsc.8.5.458

[75] J. Reed, "Open source or bust - developer engagement, MongoDB style," 2014. [Online]. Available: https://diginomica.com/open-source-developer-engagement-mongodb-style

[76] R. F. Baumesister and M. R. Leary, "The need to belong: Desire for interpersonal attachments as a fundamental human motivation," in *Psychological Bulletin*, 1995, ch. 117, pp. 497–529. [Online]. Available: http://psychology.iresearchnet.com/social-psychology/interpersonal-relationships/need-to-belong/

[77] E. Kensinger, "New Study Suggests we Remember the Bad Times Better than the Good," 2007. [Online]. Available: https://www.psychologicalscience.org/news/releases/new-study-suggests-we-remember-the-bad-times-better-than-the-good.html

[78] I.-H. Hann, J. Roberts, S. Slaughter, and R. Fielding, "Why Do Developers Contribute to Open Source Projects? First Evidence of Economic Incentives," Carnegie Mellon University, Tech. Rep., 2002. [Online]. Available: http://www.apache.org/foundation/,

[79] M. Osterloh, S. Rota, and B. Kuster, "Trust and Commerce in Open Source — A Contradiction?" ResearchGate, Tech. Rep., 2003. [Online]. Available: www.unizh.ch/ifbf/orga

[80] M. Douglas, *Purity and Danger: An analysis of concepts of pollution and taboo*. Routledge, 1966.

[81] B. J. Oates, *Researching Information Systems and Computing*, 1st ed. SAGE Publications Ltd, 2006.

[82] A. Tjora, *Kvalitative Forskningsmetoder i Praksis*, 3rd ed. Gyldendal Norsk Forlag AS, 285.

[83] M. Keep, "MongoDB: The Most Wanted Database by Developers for the 4th Consecutive Year | MongoDB," 2019. [Online]. Available: https://www.mongodb.com/blog/post/mongodb-the-most-wanted-database-by-developers-for-the-4th-consecutive-year

[84] Strozzi Carlo, "NoSQL Relational Database Management System," 2010. [Online]. Available: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/HomePage

[85] R. Elmarsi and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2017.

[86] Derrick Harris, "MongoDB co-creator explains why 'NoSQL' came to be, and why open source mastery is an elusive goal," 2016. [Online]. Available: https://medium.com/s-c-a-l-e/mongodb-co-creator-explains-why-nosql-came-to-be-and-why-open-source-mastery-is-an-elusive-goal-

[87] K. Chodorow, "History of MongoDB – Kristina Chodorow's Blog," 2010. [Online]. Available: https://kchodorow.com/2010/08/23/history-of-mongodb/

[88] Google, "An Overview of App Engine," 2020. [Online]. Available: https://cloud.google.com/appengine/docs/standard/python/an-overview-of-app-engine

[89] B. Popper, "Podcast: A chat with MongoDB's CTO, Eliot Horowitz - Stack Overflow Blog," 2020. [Online]. Available: https://stackoverflow.blog/2020/03/10/podcast-mongodb-cto-eliot-horowitz/

[90] PR Newswire, "Oversee.net Acquires ShopWiki as Anchor Property in Retail Vertical Market," 2011. [Online]. Available: https://www.prnewswire.com/news-releases/overseenet-acquires-shopwiki-as-anchor-property-in-retail-vertical-market-114789439.html

[91] Businessweek, "America's Best Young Entrepreneurs," 2006. [Online]. Available: https://web.archive.org/web/20070320085217/http://images.businessweek.com/ss/06/10/bestunder25/source/12.htm

[92] M. Inc., "About Us | MongoDB," 2020. [Online]. Available: https://www.mongodb.com/company

[93] V. Backaitis, "Open Source vs. Open Core: What's the Difference?" 2019. [Online]. Available: https://www.cmswire.com/information-management/open-source-vs-open-core-whats-the-difference/

[94] MongoDB, "MongoDB Realm unifies mobile, web, and backend development (MongoDB World Keynote, part 6) - YouTube," 2019. [Online]. Available: https://www.youtube.com/watch?v=WEL28rrG3DQ

[95] D. d. Preez, "MongoDB CEO - "I think it's clear who the winner in this market is"," 2019. [Online]. Available: https://diginomica.com/mongodb-ceo-i-think-its-clear-who-winner-market

[96] "Leadership | MongoDB." [Online]. Available: https://www.mongodb.com/leadership

[97] M. Asay, "MongoDB CEO tells hard truths about commercial open source - TechRepublic," 2019. [Online]. Available: https://www.techrepublic.com/article/mongodb-ceo-tells-hard-truths-about-commercial-open-source/

[98] Cambridge Dictionary, "Freemium definition," 2019. [Online]. Available: https://www.investopedia.com/terms/f/freemium.asp

[99] MongoDB, "MongoDB Cloud Database Solutions," 2020. [Online]. Available: https://www.mongodb.com/cloud

[100] Free Software Foundation, "GNU Affero General Public License," 2007. [Online]. Available: https://www.gnu.org/licenses/agpl-3.0.en.html

[101] MongoDB, "Server Side Public License," 2018. [Online]. Available: https://www.mongodb.com/licensing/server-side-public-license

[102] ——, "Server Side Public License FAQ," 2020. [Online]. Available: https://www.mongodb.com/licensing/server-side-public-license/faq

[103] E. Horowitz, "Email threads: Approval: Server Side Public License, Version 2 (SSPL v2)," 2019. [Online]. Available: http://lists.opensource.org/pipermail/license-review_lists.opensource.org/2019-March/003989.html

[104] MongoDB, "MongoDB Community Code of Conduct," 2020. [Online]. Available: https://www.mongodb.com/community-code-of-conduct

[105] ——, "MongoDB GitHub Repositories," 2020. [Online]. Available: https://github.com/mongodb

[106] ——, “Community Libraries — MongoDB Drivers,” 2020. [Online]. Available: https://docs.mongodb.com/drivers/community-supported-drivers

[107] ——, “MongoDB Developer Community Forums,” 2020. [Online]. Available: https://developer.mongodb.com/community/forums/

[108] ——, “MongoDB Jira Board,” 2020. [Online]. Available: https://jira.mongodb.org/plugins/servlet/samlsso?redirectTo=%2F

[109] ——, “MongoDB Developer Hub,” 2020. [Online]. Available: https://developer.mongodb.com/

[110] ——, “MongoDB World,” 2020. [Online]. Available: https://www.mongodb.com/world

[111] ——, “MongoDB Badges,” 2020. [Online]. Available: https://developer.mongodb.com/community/forums/badges

[112] ——, “MongoDB Professional Certification,” 2020. [Online]. Available: https://university.mongodb.com/certification

[113] ——, “Free MongoDB Official Courses at MongoDB University,” 2020. [Online]. Available: https://university.mongodb.com/

[114] ——, “MongoDB World is now MongoDB.live,” 2020. [Online]. Available: https://www.mongodb.com/world/innovation-awards

[115] A. Krabberød, “Tesla trenger nye patenter - Onsagers AS,” 2015. [Online]. Available: https://onsagers.no/aktuelt/tesla-patent-strategi/?fbclid=IwAR20MJuRhboWxCc9Wrsq_YgvJJNn73-a_WlQToM5WIzDxaUS6PhXUWQIc1w

[116] N. Mathur, “Red Hat drops MongoDB over concerns related to its Server Side Public License (SSPL),” 2019. [Online]. Available: https://hub.packtpub.com/red-hat-drops-mongodb-over-concerns-related-to-its-server-side-public-license-sspl/

A | Appendix

A.1. Assignment Text

Open source ('åpen kildekode') utvikling får økende oppmerksomhet, ikke minst pga. det faktum at mye av den mest interessante teknologien idag er utviklet (mer eller mindre) open source.

Et sentral kjennetegn ved Open source er det slående fraværet av vanlige metoder og verktøy for å understøtte utviklingen av teknologien.

Hvordan får Open source prosjekter strukturert og organisert utviklingen på uten disse verktøyene ? Hva (om noe) kan vanlig, kommersielt basert utvikling lære av Open source baserte metoder ?

Oppgaven vil ta utgangspunkt i nærstudier av utvalgt(e) Open source prosjekt, typisk gjennom studier av epostlister, elektroniske arkiv, irc.

Oppgaven bygger på en fortolkende forskningstradisjon til forskjell fra f.eks. spørreskjema-baserte undersøkelser.

A.2. Interview Guide

Interview Guide for CTO of MongoDB

Smalltalk:
Hi, can you hear me?
First of all, thank you
I'm a masters student at NTNU
My thesis is a literature study of the characteristics of successful open source projects. I'm then taking a deep dive into a case, MongoDB, to try and discover signs of these characteristics and other eventual discoveries.
I'm no journalist, so some of the questions might overlap.
Is it ok if I record the interview?
Everything helps, so any details you can think of is valuable to me.
If there's something you cannot answer I fully understand

Open Source:

- Why did you decide to go Open source?
- Was that the plan all along, or was it decided when releasing the database?
- Have you reflected on how not open-sourcing the project might have changed the outcome for your business?
- In what ways do you think being an open-source project has given you advantages? Are there some disadvantages?

Stakeholders and key persons:

- Were there any stakeholders that withdrew when you decided to only release the DB?
- Who are the key persons of the project, then and now?
- Who were your original targeted customers? Was it enterprises or smaller businesses? Or was it developers in general?
- What was your original business model? How were you planning to generate revenue?
- When and why did you decide to make a community edition and an enterprise edition?

Development:

- What drives you to develop your new services like Atlas and Realm? Is it the community that asks for it or is it your own innovation in internal projects that drives these new services?
- Are there any important milestones and/or obstacles that had a special impact on the company/product?

Misc.:
I am trying to find specific incidents or moments in your history that were somewhat crucial or had some kind of importance. A huge decision being taken, a serious security flaw uncovered or something similar.

One of the things I have found was that many of the drivers for MongoDB are written by the community. In this case, I am referring to the driver for Goland that was called mgo. This was created by a man called Gustavo Niemeyer back in 2011.

From what I have found, in early 2018, you (MongoDB) decided to create your own driver for Goland as you were dependent on the community driver to be updated to test out new features. Just days after the announcement of an official driver, Niemeyer updated the Readme of the repo stating that it will no longer be maintained.

Do you know anything about this, and is this the direction MongoDB is heading as it grows, replacing the community developed “pieces” with official “pieces” (in lack of better words)

NTNU

Norwegian University of
Science and Technology