Systematic Literature Review: Challenges of Open-Source Software Distributions

MASTER THESIS

Abdul Waseh Khawaja

Submitted on 15 November 2021

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open-Source-Software

Supervisor:
Dr.-Ing. Nikolay Harutyunyan
Prof. Dr. Dirk Riehle, M.B.A.

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 15 November 2021

License

Erlangen, 15 November 2021

Abstract

Open-source software (OSS) is one of the most debated phenomena in the software industry today, both theoretically and empirically. OSS allows users to modify, copy, distribute, and improve the software for any given purpose. It is a method of sharing software through source code and open-source licenses. It requires a high level of responsibility, and good configuration management practices are essential. An OSS can have multiple distributions, for example, a copy of an OS project, designed and maintained independently from the main project and other distributions. The OSS ecosystem is very critical for many technology companies today. Although the OS industry has progressed, most open-source distributions still face challenges, such as technical problems, development, security, license, and performance. The fundamental purpose of this literature review is to understand the challenges of OS distributions. The aim is to identify the critical features in an OS project, the motive for adopting the strategies used, and the challenges users face. This paper attempts to determine how organizations adopt and implement open-source distributions, focusing on configuration management and other engineering challenges. To overcome these challenges, it is necessary to understand the software components' involved. Research on OS distributions and their challenges will show how practitioners and researchers are familiar with the open-source ecosystem and once it is clear how it works, organizations can learn from best practices and apply them to internal projects. Finally, we examine the current state of the OS distributions and their process through a systematic literature review that outlines OS distributions and their performance in recent years.

Contents

Acronyms

OSS Open-Source Software

OS Open-Source

IT Information Technology

SLR Systematic Literature Review

GNU GNU not Unix

ROS Robotic Operating System

API Application Programming Interface

GUI Graphical User Interface

CM Change Management

Dpkg Debian Package Manager

CVS Concurrent Versions System

BSD Berkeley Software Distribution

SLA Service Level Agreement

OEM Original Equipment Manufacturer

RPM Red hat Package Manager

SCI System Call Interface

1 Introduction

There is currently no systematic literature review present of open-source software (OSS) distribution challenges. However, we believe that a review can help practitioners fully understand potential challenges and take appropriate actions to address them. In addition, researchers can use the results to deliberate and discuss possible causes and appropriate strategies for the challenges identified. This statement motivated us to undertake an in-depth literature review to identify and incorporate the OSS distribution challenges.

In recent years, open-source software has become a key component, playing an essential role in information technology for business and education, and providing companies such as Red Hat, Novell, and IBM with billions of dollars in service industries (Acuna et al., 2012). The increasing importance of OSS has prompted researchers to start studying the differences between open-source processes and traditional software development methods. This research includes many features of open-source development, such as different engineering challenges, stakeholder motivation, management style and requirements. In addition, the open-source software community now has a vast number of members and users. For example, the Linux/GNU operating system has multiple distributions, with millions of users worldwide, and its developers can reach thousands (Stol & Ali Babar, 2010).

In this systematic literature review, our primary focus is on Linux, ROS, Apache, Kubernetes and OpenStack software distributions. A distribution is a collection of packages configured to work together as a single program. The OS movement has been trying to downplay the fact that OS distributions usually do not charge license fees, which is not always accurate as it depends on the type of distribution (Ven et al., 2008). This study clearly shows that lower costs help promote OS distributions, and organisations appreciate that cost-effectiveness is a considerable advantage. However, this view may be misleading as it may not always be cheaper than proprietary software. Although OS project organisations may seem a little anarchistic and generally considered complicated in configuration management, there is no doubt that the OSS projects produce high-quality software. However, because OS products are primarily free for users, and the OS distributions encourage extensive collaboration, there are other costs and chal-

lenges. One of the main features of open-source software distributions is how diverse and extensive the developer community is, contributing to the project. It is a compelling argument to prove the value of open-source to any organisation. However, the larger the community and the larger the pool of contributors, the more chance there exists for issues or potential security risks (Hauge et al., 2010).

The adoption of open-source development has increased, and many companies have incorporated it into heterogeneous development, creating products by combining software from many sources and creating many different processes (van der Linden et al., 2009). To be more profitable, heterogeneous development must close the gap between traditional industrial software and OS distributions. In addition, the software ecosystem adds value by integrating projects related to a specific domain, such as a Linux distribution that integrates open-source upstream projects or an Android ecosystem for mobile applications (Foundjem & Adams, 2021). However, since each project in the ecosystem may have its release cycle and road map, this puts a significant burden on users who have difficulty manually identifying and installing project-supported versions. In terms of development speed, the parallel collaboration of globally distributed code developers makes many OS products' development speed surpasses traditional software. In terms of quality, many OS products are known for their high reliability, efficiency, and strength standards (Asklund & Bendix, 2001). In open-source software development, the source code, which is human-readable instructions of a computer program, is publicly available and widely used on the Internet (Hertel et al., 2003). Any internet user with sufficient knowledge can join the project at any time, such as downloading the source code and working on extensions or corrections. Software developers usually contribute to OSS projects for free, whether as a hobby or during regular working hours, even if OSS development is not part of their everyday work. However, some companies have recently started sponsoring OS distribution development and paying developers to continue working on OSS projects. Although this changes the voluntary nature of OSS development, it does not affect the general OS principle that the source code is available to everyone (Morgan & Finnegan, 2007).

In terms of organisational structure, membership, leadership, participation policies, and quality assurance, open-source development projects are usually organised differently from traditional software. Simple, distributed, and often informal processes make it easier to start or participate in OS projects and isolate the projects from market pressures, so many of them end or disappear. Behind the success of the OS projects and distributions is its community, whose members range from developers to passive users (Kaur et al., 2020).

The critical element that defines OS distributions is its license, which must meet many essential requirements. The main difference between many software licenses is how they relate to derivative software, as some contain terms that make

them available as open-source, while others allow greater flexibility. Therefore, choosing a suitable license for a new open-source project is as crucial as studying its license before integrating the open-source project into a proprietary system (Androutsellis Theotokis, 2010).

Nowadays, many businesses are based on open-source distribution models. Whether it is the provision of products or related services, revenue can be generated by providing support, training, subscriptions and advertising. However, the shift to OS hides opportunities related to marketing, innovation, the risks of lost profits and lowered barriers to competition (Clark, 2004). It also requires a new organisational structure, increased investment in the short term, and the further development of the OS ecosystem. Open-source distributions can be reused as products, adaptive components or codes, and other elements converted into another system. Open-source projects are increasingly forming a complete stack and use it as an infrastructure for other applications. Like web applications, the acceptance rate of operating systems is close to or even higher than that of proprietary products. The impact and consequences of open-source implementation affects the organisation's productivity, management, software quality, and development process (Cesar Brandão Gomes da Silva et al., 2017). The reuse of software components, whether closed or open-source, is considered one of the essential practices in software development because it reduces development costs and improves software quality. The context of the OS distribution release allows us to learn how to integrate into a multi-component OS environment (Napoleao et al., 2020).

GNU/Linux-based distributions are created and managed by developers, whose role is unparalleled in the traditional software development model (Mancinelli et al., 2006). The distribution publisher is responsible for compiling the available OS components and building a coherent and usable structure around them. Generally, software engineers track and compile the components, test them through integration, and resolve all necessary relationships like in dependency management (German et al., 2010). Such distributions are bundled with many upstream software components, including operating system kernels, libraries, build tools, and custom software such as desktop applications and web browsers (Adams, 2016). Therefore, many ecosystems, such as Linux distributions, provide complex, well-integrated products to end-users. The reasons for choosing Linux as one of the cores for our research according to (Yu, 2007) is:

- Linux is one of the most active OS projects and has released over 600 versions, providing extensive version history data.
- The availability of scientific, mathematical, and engineering functions in the repositories of most distributions. The most academic software available is written for Linux.

- Linux runs almost all supercomputers, many of which are clusters of Linux servers configured to run distributed parallel programs.

1.1 Types of software

According to (Stallman & Free Software Foundation (Cambridge, 2015), software types can be classified as:

Proprietary or commercial software is usually distributed as closed source and in binary form only, where it is not available to the public. Users need to buy the product, and the terms are rigorous, and modification or redistribution is strictly not allowed.

Public domain software is on the other end of the spectrum. However, there is no copyright issue, and the source code can be modified or redistributed free of charge and treated as own work (Riehle, 2009).

Free and shareware software do not require prepayment and can usually be copied, but changes are not allowed because the source code is not distributed with the product. The difference is that only limited product use is allowed without payment for shareware, whether during a fixed period or with limited functionality. Thus, it is often seen as a marketing concept and not a licensing option.

Open-source software is the method of distribution and licensing relevant for this study. The basic properties are described as being free of cost, as there is usually no license fee for this type of software, and availability of source code, as it is distributed with the product (Randhawa, 2008). Software users can modify the source code to create derivative software products or reuse the source code for another product. However, this may be subject to certain restrictions imposed by the operating license used. OSS products are protected by copyright and distributed under specific licenses that describe the conditions of use. There are many ways to license OS distributions, which differ in terms of qualification levels (Osterloh & Rota, 2007).

1.2 Classification of Distributions

There are several commercial Linux distributions available, such as Red Hat Enterprise Linux and SUSE. These products include additional services for commercial customers, such as Linux certification for specific hardware, software updates, and access to support services (Weikert & Riehle, 2013).

Some organizations are satisfied with the free Linux distributions, while others prefer the enterprise version. The enterprise version of the software provides

technical support and updates. The term 'Enterprise Edition' is mainly used when a free version is available for students or individual users. Organizations can make these decisions based on systems internal capabilities and the strategic value of the target system. Some vendors may require commercial distributions, for example, SAP (Systems, Applications and Products in Data Processing). Therefore, to obtain support from SAP, companies must purchase one of these Linux distributions. Another situation where one can charge for using OS distribution is to use software from a provider that uses a dual-licensing business model, such as MySQL (Ven et al., 2008). In recent years, many OS distributions have been developed professionally and released, which means that commercial software companies are investing a lot of capital (Khan & UrRehman, 2012). The following graphic displays the life-cycle based evolving stages of an open-source distribution.

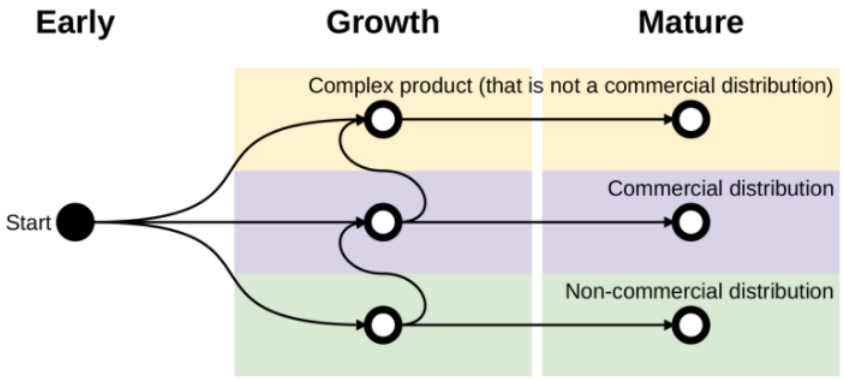

Figure 1.1: The evolving stages of a software distribution (Riehle, 2021)

Distributions generally can be classified into 'Growth' and 'Mature' stages and further classified into complex products, commercial, and non-commercial distributions. The commercial distribution is usually not free of cost, whereas the non-commercial distribution, on the other hand, is primarily free. The difference between the two instances is the configuration and not their functional programming codes. A commercial distribution can be a complex product, but a complex product may not always be a distribution and can be found in any of the stages, growth or mature (Riehle, 2021).

1.2.1 The early/growth stage

An open-source project in an early stage is often not mature enough to be turned into a distribution. A project has to be in its mature stages to be classified as a distribution.

When a distribution is in its growth stages, it is usually a commercial distribution, with few exceptions like ROS (Robotic Operating System). ROS has multiple distributions released today, such as ROS Noetic Ninjemys and ROS Melodic Morena. Once a distribution enters the mature stage, distributions may become non-commercial. This paper also discusses challenges of non-commercial distributions such as Debian, OpenSUSE and Fedora by Linux, along with other python and OpenJDK distributions (Riehle, 2021).

In addition, when it comes to commercial distributions in growth stages, there are Kubernetes and OpenStack with multiple distributions. For example, some of the most known Kubernetes distributions are Mesosphere Kubernetes Engine (D2IQ), Docker Kubernetes Service (DKS), and OpenShift (Red hat). Likewise, some known OpenStack distributions are Red Hat OpenStack (RHOPS) and Mirantis OpenStack (MOS).

	Growth Stage		Mature Stage	
Complex products	**Kafka:** Confluent Platform (Confluent)	**Lucene/Solr:** Elasticsearch (Elastic)	**Hadoop:** Cloudera Distribution (Cloudera)	**Drupal:** Acquia Lightning (Acquia)
Commercial distributions	**Kubernetes:** OpenShift (Red Hat), Mesosphere Kubernetes Engine (D2IQ), Kubermatic Kubernetes Platform (Kubermatic)	**OpenStack:** VMware Integrated OpenStack (VMware), Mirantis Cloud Platform (Mirantis), Fusionsphere OpenStack (Huawei)	**Linux:** Red Hat Enterprise Linux (Red Hat), SUSE Linux Enterprise Server (SUSE), Univention Corporate Server (Univention)	**OpenJDK:** Oracle JDK, Zulu (Azul Systems) **Python:** ActivePython (ActiveState), Anaconda (Continuum Analytics)
Non-commercial distributions	**ROS:** ROS		**TeX:** TeX Live, MacTeX, MiKTeX **Linux:** Debian, Fedora, OpenSUSE	**OpenJDK:** Amazon Corretto **Python:** CPython, WinPython

Figure 1.2: Examples of distributions in growth and mature stages (Riehle, 2021)

1.2.2 The mature stage

When it comes to distributions in the mature stage, the most common commercial distributions are Red Hat Enterprise Linux (Red Hat) and SUSE Linux Enterprise Server (SUSE). However, open-source projects like Linux, OpenJDK and Python have both commercial and non-commercial distributions. Examples of these non-commercial distributions are Debian by Linux, Oracle JDK and Python Anaconda.

Moreover, some projects started as open-source, but due to limited component configuration complexity became just regular products, for example, Kafka and Lucene from the Apache software foundation. They are usually free but have

paid versions from which earnings are made. Another example of such a product is Apache Hadoop, with multiple distributions such as Cloudera, MapR and Hortonworks, but in mature stages (Riehle, 2021).

1.2.3 Importance of Commercial Distributions

This section discusses the importance of commercial distributions when compared with free of cost, non-commercial distributions.

Commercial distributions offer guaranteed, tested and proven quality constructions for the business. There is much passion on the part of the developers working on a particular OS project. However, those working on the project are not interested in the specific issues related to business and its needs. Furthermore, commercial vendors often provide that added value to make OS work for the business. Commercial distributions are quality assured, examined and verified. The following reason to consider commercial distributions is the service level agreements (SLAs) you get with them. While a user can often turn to the OS community for help, if a critical application is running, a user may need urgent support, which is always provided by commercial distributions. A user may not always get it with the OS community, which is one reason to partner with commercial distribution.

Commercial distributions are pre-built and ready to use, with a one-click solution. Often with OS distributions, you can find repositories, but there are many different branches of information to digest and include for your project. It is not easy to consume, and often businesses need a straightforward way to consume OS support and maintenance of previous versions. Commercial distributions also support and maintain older versions to avoid any complications. They do not want to be updated with the latest and fastest like the community does. Thus, a commercial distribution will provide this support on previous versions. For example, when the HeartBleed bug hit, several companies used an older version of ActivePerl and ActivePython, and as they were commercial distributions, they were immediately able to deal with it very quickly (Copeland, 2016). The non-commercial distribution communities also fixed the issue but within a longer time frame. Finally, large companies have many servers and wish to standardise on a single distribution that works on the different platforms with which they work.

Another reason why organisations prefer commercial distribution is that many companies have compliance requirements. If they use OS internally, they must have a third-party commercial vendor providing support. For example, finance, health, government sectors have mandatory rules in this regard. As a result, they cannot run OS distributions on critical applications without third party assistance. Integrating distributions into their products can help them reach the market faster and provide the risk-free reliability their customers demand.

However, ignoring the OS license terms can be dangerous. This can cost lawsuits for intellectual property infringement, heavy legal bills and damage to reputation. Partnering with a commercial supplier helps them obtain turnkey redistribution licenses and eliminate legal risks (Copeland, 2016).

2 Research Method

In order to identify all relevant research studies, we performed a comprehensive literature search based on guidelines for conducting a systematic literature review (SLR), as presented by Kitchenham (Kitchenham & Charters, 2007). This section outlines the procedure of the review protocol consisting of the research questions, literature review plan, selected digital libraries, and the qualitative data analysis techniques. The protocol development was performed under the supervision of a professor and researcher at Professorship for Open Source Software at the Friedrich-Alexander University of Erlangen-Nurnberg.

2.1 Research Questions

In this section, we define our research questions. As explained in the introduction, the goal is to analyse the theoretical framework and to answer this, we study the following research questions in relation to OS distributions.

RQ1: What are state-of-the-art engineering challenges of open-source software distributions?

The OSS ecosystem is very critical for many technology companies today, as it helps developers become more productive and structured in how they manage the software their businesses rely on. Therefore, understanding the software ecosystem, its engineering challenges and the elements of each software can help companies create and optimise organisational products.

RQ2: What are the issues organisations face when implementing open-source distributions, and how does configuration management play a role?

According to the survey in (Hecht & Clark, 2018), OSS plays an important role in how best practices are adopted by organizations. This research question helps understand those practices, configuration management and the complexity organisations face when adopting OS distributions.

2.2 Literature Review Plan

Research method and strategy is the key to any systematic literature review. This section discusses the method and tools used to perform the survey, collect literature, and perform qualitative data analysis (QDA) to write a systematic literature review.

The whole SLR process is summarised in Figure 2.1 below, thus presenting a general idea of the adopted process. In addition, it provides a comprehensive view of the general stages of SLR. There are three main stages of the literature review: Planning the review, conducting the review, and reporting the review (Staples & Niazi, 2006).

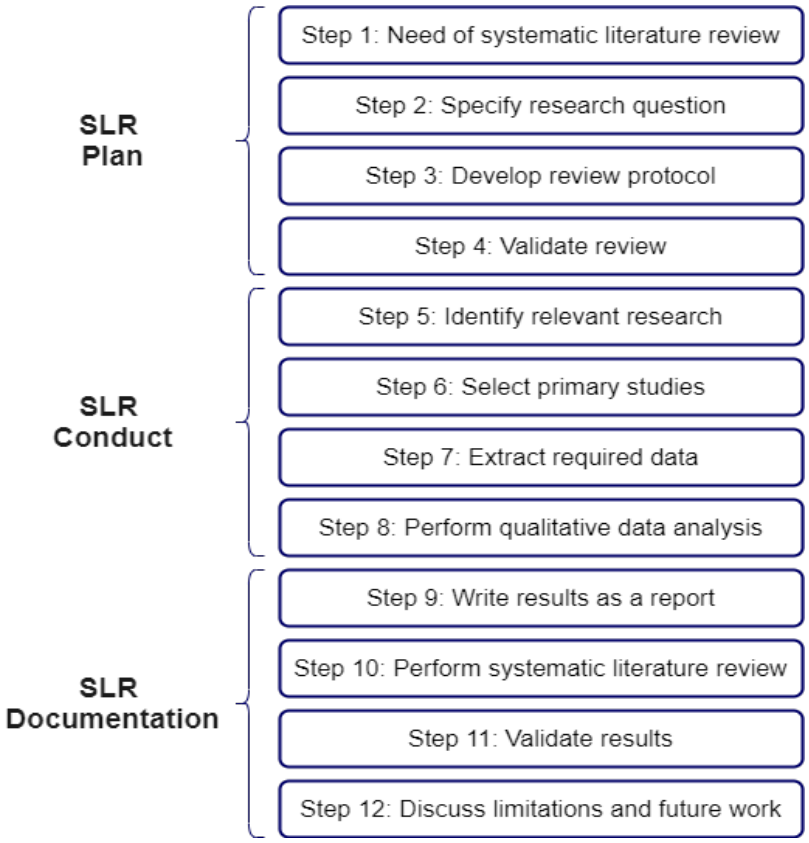

Figure 2.1: Phases and steps of a SLR on the (Kitchenham & Charters, 2007) guideline by (Nasserifar, 2016).

The choice of the correct scientific databases and the modification of the search string are also documented. The inclusion and exclusion criteria help authors through the selection stages and, therefore, the identification of relevant literature may be less unbiased. Subsequently, the design of quality assessment and data

extraction according to the research objectives and questions is also carried out in the planning phase. In planning, the external validity is mainly respected by the authors and the protocol. The other problem is finding the appropriate answers to research questions and is known as internal validity. It can happen during document selection and rating in the quality assessment activity or data extraction processes (Nasserifar, 2016).

2.3 Literature survey

Collection of literature was performed using search strings or keywords. Following is the list of different digital libraries and databases used:

- Google Scholar
- Web of Science
- ACM Digital Library
- ScienceDirect
- SpringerLink
- Scopus
- IEEE Xplore

The digital libraries and database mentioned above were chosen because they are related to the source of software engineering publications and are the most known scientific libraries due to their credibility in information technology and computer science.

2.3.1 Search Process

At this step, we search for literature based on the databases and digital libraries and search string had to be modified accordingly for better results. However, many papers or journals are not accessible, so we used the university (Friedrich Alexander University Erlangen-Nurnberg) VPN to access these digital libraries.

Starting with an automated search, then manual search to identify potentially relevant papers and grey literature, and then apply the inclusion/exclusion criteria. The search is performed using the specific syntax, considering only the title, keywords, and abstract. In addition, each repository is configured with a search to select only the journals completed in a particular period to ensure relevance and quality. Manual search supplements automatic search to obtain grey literature and scientific journals.

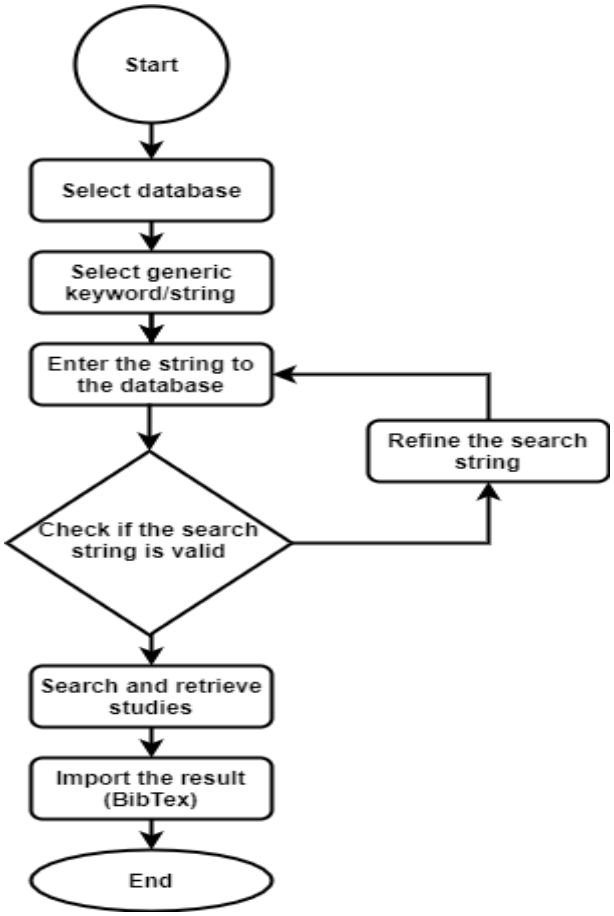

Figure 2.2: SLR search process proposed by (Unterkalmsteiner et al., 2011).

Grey literature is the material and research produced by organisations outside traditional or academic publishing. Common examples of grey literature include reports, government documents, white papers, and presentation reviews.

The literature search was done using a forward and backward reference search method. Forward reference search is when a researcher identifies an article of interest that cites an original journal. Links are provided to get access to the article from the database. It helps the researcher expand knowledge on a topic by looking at follow-up studies and identifying new developments. Whereas, backward referencing search method works in the opposite direction. Starting with an article of interest, and straight away go to the references cited and use them to further look for appropriate literature. It helps understand the origins and development of a theory or model of interest. It can also assist in identifying institutions or organisations that specialise in a topic of research (Library, 2021). This research process yielded around 90 papers in total. As the scope was defined, few papers were excluded in the initial stages. In the end, after performing

qualitative data analysis and refining the scope, we shortlisted approximately 70 research papers and grey literature for this literature review.

In data extraction process, all relevant information for each study is documented as it helps aggregate the data and link it to the source. For example, reading the title and abstract to identify potentially relevant literature. It is based on the analysis of the title, abstract and conclusion. Not so relevant literature is then discarded. If there is any doubt about whether a paper or journal is relevant to the topic, it is then considered later on if needed. The selection criteria were relatively straightforward. The publication must be a scientific journal or any grey literature. The work suggests empirical research related to our topic of open-source distributions. The most recent one was considered if multiple journals, articles, or reports on the same study were found. To make the search more precise, literature directly around at least one research question was targeted. Similarly, exclusion criteria consisted of not paying attention to tutorials and ensuring that the research is based only on expert opinions and there is no convincing evidence, except for when looking for grey literature. Any publication of an earlier version of a recently published work was also excluded.

As this is not a very well researched topic, the focus was on the grey literature, particularly after publications and academic papers. Nevertheless, some exciting blogs, white papers and presentation reviews were found on the challenges of various distributions. The following data was extracted from the research:

- Name and Author
- Article type (journals, articles, white papers)
- Research question and purpose
- Results and conclusions
- Limitations

2.4 Qualitative Data Analysis (QDA)

As qualitative data analysis was performed, the quality standard of the literature was also a key element, especially when looking for grey literature. For example, analysing whether the article is based on research or just a report based on expert opinion. It was integral that the research objective was defined with a complete description of the investigation's background.

The next step was preparing tools for data collection and analysis, for which software MAXQDA was practised. It enables researchers to conduct research using various analytical methods, such as those used in evidence-based theory, qualitative data analysis, and case studies (MAXQDA, 2021).

2.4.1 Code System

MAXQDA is a tool that also helps classify the literature into different categories, as it is imperative to make sure that the research is headed in the correct direction. Therefore, the next step was creating a code system on MAXQDA.

Next, the research identified and targeted open-source software distributions and made a list of them. The following step was to find out and look for literature concerning each distribution to decide whether to include or exclude it from research. Being one of the most common and popular open-source software with multiple distributions, Linux was the first automatic choice, followed by ROS. During this time, some other open-source software distributions were also discovered, like Kubernetes, Apache, Eclipse and OpenStack.

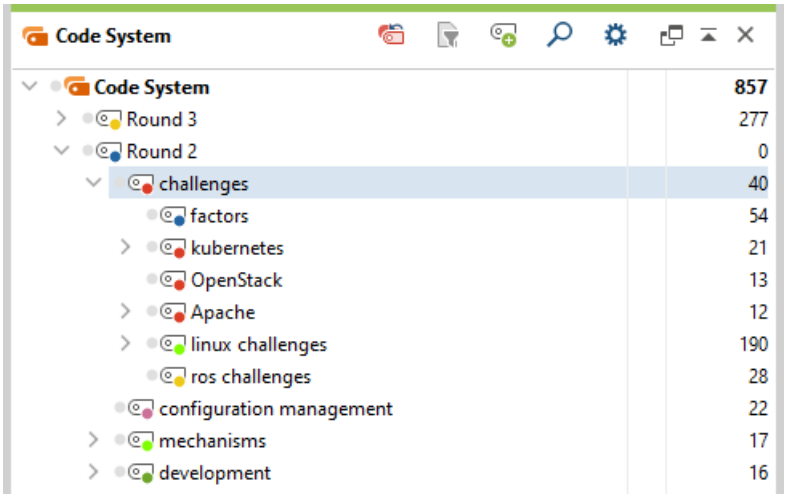

Figure 2.3: Initial code system with respect to distributions.

A broader approach was employed, to begin with, and literature was classified into different categories and sub-categories. The analysis began with going through different challenges concerning different distributions. Similarly, a code system was built of distributions in the first phase with its engineering challenges and limitations. As it was done for all the distributions, phase two was about studying those challenges of individual distributions and verifying if there was any common element between all distributions.

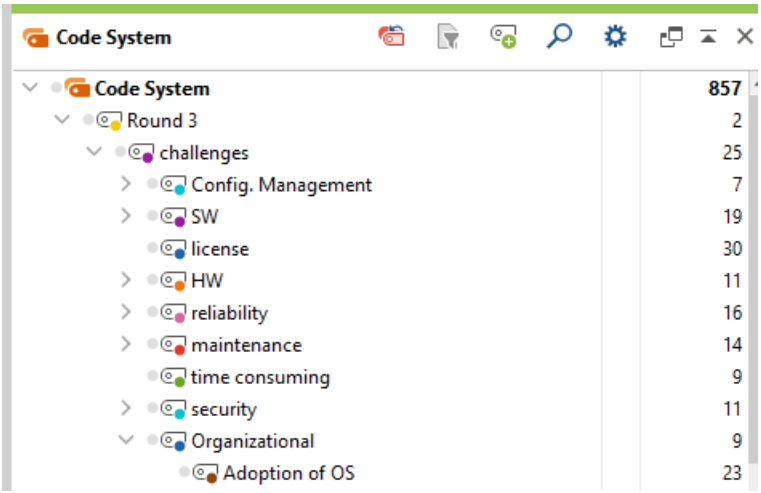

Figure 2.4: Code system of different challenges found.

As the results suggest, there were quite a few common challenges, and few were quite distribution-specific, with challenges arising due to the software architecture of a particular distribution or any changes found in the newly released version. Following this procedure, a much deeper approach was adopted in the next phase, and the code system was adapted to classify common challenges and divide them into further sub-categories. All the challenges in grey literature, case studies and academic journals were studied.

3 Results and Discussion

Before discussing the challenges, it would be interesting to know why organisations would be inclined towards adopting open-source software distributions.

Straightforward to acquire. The procurement process is not too long and relatively straightforward to adopt open-source distribution and can be integrated into an organisation. However, although it is easy to obtain, organisations generally do not want to integrate OS distributions into their applications unless they follow the appropriate procedures within the company (Copeland, 2015).

Quality of the software. The quality of the open-source software distribution is constantly improving, as anyone in the software community can improve the code, so this ripple effect works. The code is constantly worked on, and it does not get stagnant. Furthermore, when there are any specific code issues or challenges, the open-source community can be called for help.

3.1 Challenges of OS Distributions

This section answers the research questions and discusses our literature review's state-of-the-art engineering challenges and configuration management framework.

3.1.1 Software

The major problem with numerous open-source distributions is that there is no unified configuration system for device management. This issue could be solved manually in network settings, but no installer/package manager is tracking the overall distribution from development to release (Boender, 2012). The distribution repository does not contain all the open-source software available when installing any software by downloading the required package. Furthermore, different distributions can use different versions of the library and different compiler flags, which results in much confusion for the organisations and their users (Silakov, 2008). For instance, in Linux distributions, the two most popular open-source desktops, KDE and Gnome, can only configure some settings themselves,

so each release builds its manual application to configure bootloader or firewall and group services. As a result, firms can enforce uniform distribution for everyone in the organisation, but it will never be possible if the customers are free to choose (Diener, 2018).

Support

Open-source distributions alone can not make a difference until different servers, and operating system supports are not present and compatible with every release. When talking about Linux distributions, we find many different servers that should work efficiently for new releases. Similarly, the X.org server is one of the most popular servers around. It is a desktop infrastructure that provides an interface between the hardware and graphical software (Wikipedia: Xorg, 2021). Unfortunately, the X.org server is outdated for modern-day PCs and multiple packages. It can be precarious, with no standard API for growing graphical user interface.

When discussing software support as a challenge for many open-source distributions, the X.org server no longer guides unique scaling modes for special monitors and presently has no manner of completely storing and restoring settings modified with the aid of the consumer. Moreover, X.org does not transfer high resolutions in laptops when complete display software with custom editing is available for the video and gaming industry. X.org server is not multi-threaded either and does not control multiplied brightness settings and permits programs to solely take hold of keyboard and mouse input. If such programs are not efficient enough, a user is left with an unmanageable device that can not transfer to textual content terminals. Keyboard management is also a unique challenge for virtual machine packages with the X.org server as it does not support some keys in the latest versions (Tashkinov, 2021).

After introducing the impact of the server, the following vital support comes from drivers. The NVIDIA Driver is used to install GPU on the PC and communicate from the operating system to the device. This driver is required in most cases for the hardware device to function correctly. In addition, it designs graphics processing units (GPUs) for video editing, the gaming industry, mobile and automotive market (Wikipedia: NVIDIA, 2021). These proprietary drivers each require custom utilities and specific packages with extensive graphical operations that can also cause older version laptops to hang at times (Tashkinov, 2021).

Font rendering is also another issue noticed in few OS distributions. As we discussed different software challenges, we encountered some font rendering issues, mainly in Linux distributions. For example, *ClearType* is a font smoothing technology designed to smooth the fonts on the screen to be more readable on LCDs. Unfortunately, ClearType fonts (through GUI libraries) are not supported quite

well in a few Linux distributions. Moreover, even though the font rendering technology is now supported, there is no approach of well tuning it, and as a consequence, ClearType fonts from Windows appear not so good. Quite regularly, default fonts also do not appear the best because of default font configuration settings.

Quite often, distributions are unusable because they do not support new hardware, mainly GPUs (in addition to Wi-Fi adapters, NICs, sound cards, and external drivers). Furthermore, because of previous libraries, one cannot use a new software program in specific distributions. Another drawback is that most distributions are made so that their core components (like kernels, Glibc, Xorg and Mesa drivers) can not be improved without upgrading the entire system. Also, modern-day hardware typically can not deploy the latest version of distributions since most do not contain the kernel release. In order to fix this and deploy kernel, it is required to appoint numerous hacks (Tashkinov, 2021).

HWInfo64 is a simple application that scans the components of a PC and displays basic information about the operating system, memory size, and RAM. It also displays other technical data such as logical processors, memory speed, and battery consumption rate. In most open-source distributions, hardware sensor support is not enough. For instance, HWiNFO64 detects and suggests hardware sensor assets on a standard laptop and over multiple sensors. This scenario does not work too well on laptops, as now and then, the best readings from sensors are CPU cores' temperatures (Tashkinov, 2021). There is no idea of drivers in Linux distributions, for instance, other than proprietary drivers for NVIDIA or AMD GPUs that are separate packages. Furthermore, nearly all drivers are already both withinside the kernel or numerous complimentary packages. Thus, the consumer cannot recognise whether or not the hardware is undoubtedly supported or not all of the required drivers are indeed established and operating well.

Numerous open-source distributions today do not have adequate documentation and lack appropriate supporting manuals. Furthermore, no unified broadly used device for applications signing and verification; as a result, it becomes an increasing number of intricate to verify applications that are not covered. Furthermore, most OS distributions are no longer audited due to this. As a result, the quality of an OS distribution is compromised. To summarise, there is no unified configuration system and no backwards and forward compatibility due to volatile and continuously developing kernel APIs (Kahani et al., 2016).

Installation Challenges. For instance, deploying some of the distributions, such as OpenStack and Kubernetes, is not very straightforward and could be challenging if a user experiences it for the first time. Moreover, it is particularly complex during the installation phase because it is a series of packages, involves much manual scripting, and each of these applications must be configured according to the user's needs (Lehmann, 2017). For example, an engineer should create a Kubernetes script in YAML or JSON format and write commands to configure an application. However, it becomes a daunting task when the goal is to run multiple configurations per day (Tozzi, 2020). It means that the installation should be carried out by a specialist, or ideally by a company of several specialists to cover the range of additional skills required for an optimum configuration (Thomas, 2019). As a result, more often than not, organisations end up using scripts and commands that slow deployment. The simplest way to overcome this problem is to find a vendor who can provide a complete package, including software, hardware, and initial configuration. It reduces the need for companies to hire more technical and qualified staff, making the process more straightforward, but it is still essential to do due diligence to make sure experts know as much as they say they know (Constantinescu, 2019).

Upgrade. The upgrade process is one of the most interesting catches in modern hosting. One of the primary goals of the cloud-based infrastructure is to provide both high reliability and availability. Unfortunately, updating a distribution is not always easy. In fact, due to its complex nature and approach to multi-project development, downtime is sometimes unavoidable. Unfortunately, this process usually relies on installing updates regularly, but developers do not have much incentive to provide support and updates for an open-source project. To make matters worse, OpenStack, for example, has officially discontinued support for some of its versions. As a result, it can make updating OpenStack more complex than updating an alternative (Shiozaki, 2016).

3.1.2 Hardware

One of the critical challenges of open-source distributions can be classified under the heading of hardware challenges. In order to make organisations adapt to open-source distributions, it is essential to eliminate these hardware challenges. In many versions of OS distributions, peripheral devices and gadgets are either poorly supported or not supported at all. For example, hardware such as Broadcom wifi adapters can not be used without a running Internet connection and must be manually configured. Thus, it can be understood that it is not the easiest of tasks to integrate hardware to open-source distributions as new hardware constantly requires support. Furthermore, specialised software to manage gadgets like printers, scanners, cameras, webcams, audio players, and smartphones does not exist or must be manually set up. To sum it up, hardware is incom-

patible with a few open-source distributions, so it needs to rely upon third-party applications.

Software regression is a type of software error in which a previously working function stops working. This may happen after an event, such as after a system update. The fix is often included in the software patch. Regression is a user-visible change in kernel behaviour between two versions or releases. Regression testing is significant to evaluate the functionality of the new program code. It ensures that the new code will not disrupt the existing code functionality and guarantee no defects after installing the software update. In addition, it allows retesting of existing software after changing the application. Hence, regressions were also noticed in the kernels when some hardware stops working in the new version, which is a big problem for users and organisations.

Peripheral Devices

When users decide to switch to a new OS distribution, they assume everything will work out well, but they do not research, and a lot of the time, printers, scanners and other devices face compatibility issues. The latest problem users face in the tech industry was setting up hardware like 'touchpads' alongside wifi cards or USB wifi adapters (based on Realtek chips) as not all versions are supported under a few distributions. It was also noticed that few power-saving modes did not work too efficiently, and battery life was not the same under numerous OS distributions. Moreover, new portable devices can not be used as compatibility packages take time to install under different distributions (Tashkinov, 2021).

Video Accelerators

Most machines use the NVIDIA technology that does not work too well in open-source distributions. Users struggle with screen tearing and new kernel releases. NVIDIA driver is comparatively slower due to power and fan speed management, as it does not provide the required firmware. Most of the complex game releases are accompanied by a matching driver release from AMD or NVIDIA, as the open-source community does not have the resources. Most of the time, drivers in these OS distributions require manual configuration for non-standard and very high display resolutions. In industries, it is noted that setting up multi-monitor configurations with multiple GPUs running can be a significant task as well (Tashkinov, 2021).

Audio Subsystem

It is also noticed that advanced configuration is available only by editing some text files in the console under a root session in the audio subsystem environment. PulseAudio is not suitable for multiplayer mode, and there is no reliable echo

cancellation. In addition, various audio effects like volume normalisation are not included or enabled by default in most OS distributions (Tashkinov, 2021).

3.1.3 Maintenance

Maintenance is key to any open-source platform, be it a commercial or non-commercial distribution, and distributions must be maintained and improved in new releases or updates.

Implementing peer review. Peer review evaluates the work done by people with similar abilities as the developer and qualified professionals. With the development of open-source projects, it has become increasingly difficult for a limited number of senior contributors to review each code request. This becomes the bottleneck of the entire project and slows down the progress and growth of the software distribution. Peer review is the most common way to solve this bottleneck. In addition, this process requires other developers to understand the mission of the project and the quality that everyone should achieve (Hurley, 2014).

Development dependency. Complexity in managing dependencies between large amounts of modules or packages that make up the distribution. Editors need to be updated with recent source code changes by developers, which is to be done manually and is an error-prone task. This could be very time consuming, as it leads to backtracking to cater changes to the modules. Users expect to find an upgrade path that does not disrupt the system when moving from an old version to a new distribution version. Binary compatibility is a huge challenge, as differences between distribution versions mean that every application should be recompiled for every particular system (Silakov, 2008).

Unstable API and lack of compatibility. It is challenging to adapt older versions of OS software in the latest distributions. In addition, backward compatibility makes building closed source applications for most open-source distributions extremely difficult and expensive. Open-source distributions without active developers and maintainers are discarded when their dependencies cannot be satisfied because the old library is outdated or unavailable. For example, for this reason, many applications are not available on modern distributions. WinSxS is a folder that stores main components and keeps older versions if the user has to switch back to the previous Windows version and multiple distributions today have no WinSxS equivalent, so there is no easy way to install conflicting libraries (Tashkinov, 2021).

Lack of collaboration. There is no primary authority to organize the development of different parts of the open-source stack, which often leads to changes made by one project disrupting other projects. Although the open-source movement lacks a workforce, various OS distributions seek sufficient resources to

branch projects. Most distributions have a short update/release cycle (in some cases only six months), which keeps changing. Moreover, due to applying retention policies and there is usually no officially approved method for installing advanced applications, long-term support distribution is not suitable for desktop users in most cases (Adams, 2016).

Appropriate IT Support. Like any server-dependent content, OS distribution requires hosting and maintenance of the code, platform, version control, and related software engineering tools. While often overlooked, IT support is crucial for maintaining the new versions, running scheduled backups and recovery when necessary, and hardware maintenance (Martin & Hoffman, 2007). Not surprisingly, most OSS members are classified as active or passive users. They publish bug fixes and inquiries from time to time, subscribe to mailing lists, read news, and mainly use the software (Hauge et al., 2007).

Data Loss. For instance, few Apache and Kubernetes distributions indicate that some arguments are under-replicated. The data is not replicated, and these warnings indicate a potentially serious problem as the likelihood of data loss increases. It can happen entirely unexpectedly, even if the user does not do anything on his end. It usually happens when clients affect the data volume, and a spike in the data volume causes the package broker to save the message conversion (D2iQ, 2021).

If working with large data sets, using some distributions to archive them can cause several problems. The main problem is that some distributions, for example, Kafka by Apache, store redundant copies of data. As a result, it can affect performance, but more importantly, it dramatically increases storage costs. The best solution would be to use Kafka only to store data for a short period and migrate the data to a relational or non-relational database, depending on specific needs. As we discuss the issues with the long term storage solution, we highlight an additional issue related to it. Downstream customers often have completely unpredictable data demand patterns, and as a result, settings are a bit of a problem. For example, distributions stores messages, which can take up a lot of disk space and download the data; one must set the retention period or configurable size. If data retention settings are not correctly adjusted, there is a risk of rendering data useless or paying too much for storage.

The volume of data flows can go in both directions, and that is why it is essential to choose a distributed messaging platform that is easy to scale. With some distributions, this is a problem due to balancing things manually to reduce resource bottlenecks. The user has to do this whenever there is a significant change in the data flow.

3.1.4 Reliability

Coding standards are a set of guidelines or rules that open-source projects expect from all code submissions. Code standards are usually simple procedures to ensure that any code submitted looks the same and the system feels like one piece of software when combined (Ven et al., 2008).

Lack of stability, bugs, and regressions. There are many regressions in kernel and userspace applications that somehow ruin the initial code, due to which kernel at times does not support new hardware. Some regressions can even cause data loss, as discussed earlier. Unfortunately, most open-source distribution projects have almost no regression testing. For example, Microsoft reports that Windows 8 has undergone 1.24 billion hours of testing, while the new Linux kernel, to be specific, has less than 10,000 hours of testing, and each version of the Linux kernel is comparable to the new version of Windows. As a result, the staging version of the kernel usually regresses, although most distribution developers insist that such a version be updated immediately (Tashkinov, 2021).

Lack of standardisation. Many open-source distributions are incompatible with different configurations, packaging systems and libraries (Ansari & Chaubey, 2014). A console application is used to configure computer settings. If we talk about Linux, Debian-based distributions use the plain text utility 'dpkg-reconfigure' to perform specific system maintenance tasks, which can be challenging for a new user. Dpkg is a tool to install, build, remove and manage Debian packages. It is the software package management system in Debian distribution and its numerous derivatives (Boender, 2012).

Automation. Application-oriented companies strive to automate repetitive network operations and eliminate manual operations that may cause errors. To achieve flexibility through continuous integration and continuous delivery, firms must deliver applications quickly (Rajagopalan, 2020). First, the application web service must be an API because storage in a hybrid cloud environment is complex, and many different products or tools can implement storage management in this environment. Secondly, because each application has different business requirements, it is not always adequate to apply templates to other applications (Pritchett, 2020).

The reliability of the software requires release delays to allow sufficient time for testing (Cesar Brandão Gomes da Silva et al., 2017). The lack of a transparent business model that is attractive to a broader range of industries has not been widely known or valued. The critical issue also involves the standardisation of OS distributions. Many participants believe that this is urgently needed, but it may be too early. The main disadvantage of these distributions is that it is not very easy to get accustomed to. There is a lack of applications that run both on open-source and proprietary software; therefore, switching to open-source involves

a compatibility analysis of all the other software used that run on proprietary platforms (Agerfalk et al., 2005).

Time complexity. Some distributions present challenges, not in cash flow, but in the developer experience and the time required to quickly and seamlessly deploy the first system without spending too much effort. In addition, once the OS repository reaches a specific size, finding and browsing various items and personnel details becomes time-consuming (Shiozaki, 2016). Therefore, proper search and navigation infrastructure is critical to the success of OS distributions. In addition, a short reporting cycle and shorter running time mean less time for testing. Application service resilience is more critical in container-based applications because containers can be activated or deactivated faster than traditional infrastructure. Applications in the cloud or data centre can take advantage of this flexibility at the computing level, but network services can still cause bottlenecks (Rajagopalan, 2020). The service dependencies of applications migrated to the hybrid cloud need to be modified to support these applications. For example, if a single service in a particular location becomes a bottleneck, all hybrid cloud applications will depend on that service. Therefore, a single-site failure mode is introduced (Pritchett, 2020).

3.1.5 Organisational

Today, most companies have a hierarchical organisational structure, making it difficult to share code and product release plans with different stakeholders. This section answers our second research question, discussing challenges and issues faced by organisations when adopting open-source distributions.

Implementation. Incompatible platforms, as discussed before, are a significant challenge, and due to the lack of support for new technologies and hardware, this is one of the major reasons why industries even today are reluctant when it comes to migrating to OSS distributions. A study concluded that the traditional separation between operators and developers seen in many organisations is the main obstacle to launching applications quickly and frequently (Tozzi, 2020).

Developer briefing. The most crucial aspect of software development is the skills of each developer. This requires developers to understand the tools that adopt established coding standards concerning the source code. It is one of the main organisational and management issues (Ven & Mannaert, 2008).

Leadership. The open-source model is based on at least one product owner of specific software modules and a team of developers. For example, a system provides good guidance for a given software module in the open market, and the problem occurs when a software module manager decides to leave, and several projects rely heavily on this module. Alternatively, even worse, when there is no specific conductor for a specific module.

Adoption of open-source distributions

The adoption of open-source distributions in organisations represents a significant paradigm shift in improving and managing fundamental structures (Chau & Tam, 1997). However, for organisations, many technologies are too large and complex to be understood using human cognitive abilities or typically deployed in the arbitrary jurisdiction of the organisation (Dedrick & West, 2004).

When considering open-source software distributions, there are significant differences from the previously more traditional platforms. First, the research and development, sales, revenues, and support are the liabilities of a clearly defined commercial business to profit from its products. Open-source uses collaborative research and development and cooperation with companies whose roles are less relevant or much less described. Second, the fundamental difference between open-source software is that the provided code is widely available to everyone, and thus the deployment teams have a chance to tailor the program to the individual needs (Morgan & Finnegan, 2007).

What are the main factors influencing the use of the open-source platform? Organisations use open-source distributions for various functions. However, the size of the equipment and the costs to install makes it a rare choice to choose a new platform. Furthermore, one of them resulted in a vast search overhead to choose the best alternative. Therefore, we observed two models when considering to adopt OS distributions:

- Organisations that choose specific hardware first.
- Organisations that prefer operating systems, that suit their business needs.

Some organisations are open to adopting everything that comes with an OS distribution, whereas some are more reluctant and can not afford to compromise. As the models suggest, some organisations are dependent on the hardware, so they want to choose any distribution that it would support easily. On the other hand, the other model suggests some prefer operating systems based on API support and check if business-critical applications run on a specific operating system.

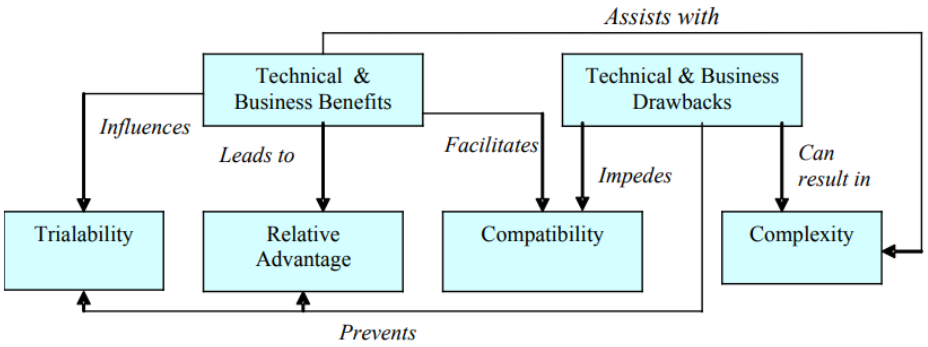

Figure 3.1: The impact of OSS adoption in technological context (Morgan & Finnegan, 2007).

Here in the graphic above, we notice the impact of the benefits and drawbacks on OSS adoption in a technological context. It is observed that the adoption of OS distributions has both, drawbacks and benefits. Many technical and business benefits can influence compatibility, and it can also increase complexity which is a significant drawback.

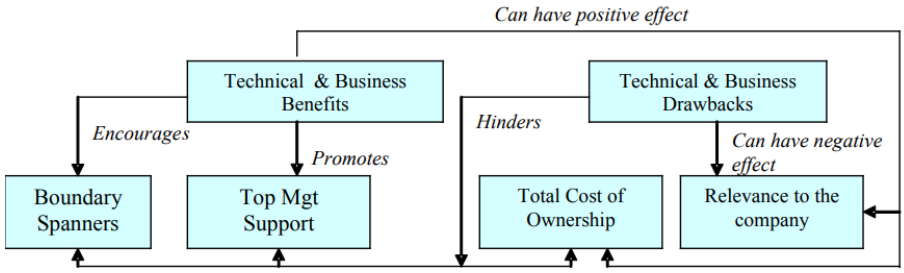

Figure 3.2: The impact of OSS adoption in organisational context (Morgan & Finnegan, 2007).

When discussing the impact in an organisational context, we again observe many advantages, but at the same time, a few drawbacks and challenges, which are essential for our research and this paper. It is observed that these benefits can also cause considerable challenges in organisations. When adoption of OS distributions encourages spanning boundaries and promotes management support, this can also be transformed into a drawback where it may harm the company's relevance and increase the total cost of ownership.

An organisation can choose to use OS distribution without further development. Alternatively, free software becomes more efficient if the organisation actively participates in developing a community. However, this requires additional resources.

The amount of resources required depends on the level of involvement and the types suggested by (Bonaccorsi et al., 2007):

- Project coordination
- Code delivery
- Development

A realistic concern that organisations have is regarding the future or longevity of OS distributions. For example, if a particular OS distribution is adopted, users do not like to find themselves in a situation where the community that supports that product disappears. If this happens, it means that there is no support or update for that particular product. In this case, an organisation can decide to take care of the maintenance of the project. However, this would involve additional maintenance efforts and could distract the organisation from its core business (Stol & Ali Babar, 2010).

3.1.6 License

Open-source software distribution is a method by which software is available through source code and open-source licenses. Here, the owner grants the right to modification and distribution of the software for any purpose. The number of people using or implementing open-source software in the public sector is increasing. The use of open-source software needs to be understood from different angles. It is essential to understand different terms that refer to or have the same meaning but are used in different contexts. This chapter will focus on licensing issues and address their challenges.

Some OS licenses only allow the reuse of the software when the derivative works are also under the same license, but the definition of the derivative works may not always be clear enough to specify how to use the OS distribution. Therefore, a part of the generated software product is based on the operating system code of other parties, and the first one is used as the operating system license, and the second one is used as a proprietary license (Perens et al., 1998). Developers pay more attention to the functional features of the code rather than the user interface and usability. They usually have no training in handling such issues, and the various licensing options of different software and the risks of combining them present challenges for application developers. Choosing an open-source software licensing program is a significant issue that requires a deeper technical understanding of various licensing types. First, the user should know the rights to the source code that is used. The software license is not necessarily a contract, but there is still copyright to protect the user's work (Kogut, 2001).

Accepting general licenses. Open-source software distribution can be used, modified or redistributed for free but is subject to certain restrictions on copyright

and its status protection (von Krogh & von Hippel, 2006). OS projects must always have software licenses. It defines the distribution strategies and methods by which other people can use the software. A vital step to consider when allowing developers to contribute to the code is the license applied to the proposed code. It is important because the developer must know and agree with the type of license chosen by the project stakeholders. For example, some open-source projects require signatures to identify the license type of any code submitted (Peters, 2004).

The software license embodies these rights and restrictions, which is the contract between the proprietary software (the licensor) and its potential users (the licensee). OSS licenses come in many different types, but they usually aim to provide source code. Thus, the OSS licensor (usually the owner or author) can be an individual developer, development team or organisation, and owns copyrighted software. On the other hand, the licensee is the end-user of the operating system or the person who integrates it into their product or application and then distributes (Androutsellis Theotokis, 2010).

Characteristics

They are free licenses for software distribution, and they include the condition that the source code must be provided to the licensee. Thereby giving up any concept of ownership and being free from copyright issues. Open-source licenses can be extended to 'copyleft' licenses. It refers to a license that permits to copy any work and requires that the work use the same license as the primary work. Copyleft is open-source licensing that grants the right to copy, adapt, or distribute software. However, it restricts all derivative works from being published under the same license. Thus, it shows how software and the freedom that comes with it becomes inseparable (Alam & Soomro, 2016).

Concerns and risks

One of the concerns about the adoption of various OSS licensing systems is that the use of licenses that combine open-source and proprietary software effectively undermines the concept felt by the supporters of the OSS movement. In addition, commercial software development firms may feel that there is a risk of including OS distribution codes in their products due to the ambiguity of specific definitions (Sardina, 2019). Free and open-source software is the same as long as they serve their purpose, but the license terms are different. Any software that uses free code should also be free, but not the same as open-source distribution is not necessarily free.

The most popular licenses according to (Khan & UrRehman, 2012) are:

- GNU General Public License
- Free BSD License
- Mozilla Public License
- Apache License
- X11 License (also known as MIT License)

Several studies have pointed to the complex OS licensing situation as a problem. For example, studies have reported a lack of consistency between license agreements and little guidance on the interpretation of open-source licenses. It is therefore not surprising that OS licensing is seen as a complex issue. As a result, research efforts have been made to resolve this problem (Stol & Ali Babar, 2010).

3.1.7 Security

More contributors means more risks. This is a severe problem as the community of developers who contribute code and solve problems, continues to grow and there is a need to develop some guidelines for everyone involved. For example, developing a presenting code requires a standard license, and introducing peer review is a good practice in projects. Some believe that the larger open-source projects are, the more vulnerable they are to security threats and dangers posed by different participants. However, in the unique and controllable environment of closed source companies, the benefits of open-source far outweigh the perceived risks. In addition, through thoughtful community organization methods, these risks can be managed (Hurley, 2014).

Visibility. Application visibility is significant for container-based applications. Both application developers and operations teams need to see the interactions between services to identify false interactions, security breaches, and potential delays (Rajagopalan, 2020). Developers can unintentionally access the product's internal structure, that is, the source code and design. Usually, they at least want to know who is accessing their source code and for what purpose. It depends on the project, from fragile security requirements to very confidential company information. Therefore, open-source requires proper authentication, authorization, and verification mechanisms to control access to source code, which is in contrast to OS mechanisms that may be accessible to everyone (Kohgadai, 2020).

3.2 Configuration Management

Configuration management is the art of classifying, organizing, and managing software changes during the development process. This chapter talks about the importance of configuration management in open-source distributions from the development to the release phase (Mockus et al., 2000). The list of protocols that are the most relevant in configuration management suggested by (Asklund & Bendix, 2001) are:

Version control. Versions with different document parameters are stored, so they can be retrieved later on when needed. Therefore, it is essential to make a comparison with the latest versions.

Build management. It is used to collect all the original modules of the system and helps build and update the system with released modules.

Workspace management. Developers usually want to use configuration transparently without worrying about version control or viewing changes made by others who use the same configuration.

Change management. It revolves around support management of change requests, error reporting, implementation of these changes, technical documentation of problems and solutions.

Release management. Identification and organization of all included files and assets. The assembly manager is responsible for the correct settings and functions of the packaged products.

3.2.1 Version Control

The CVS tool is mainly used for version control in most OS projects to track all changes to project source code files. CVS is widely regarded as the best free, full-featured version control tool. It is the most popular tool for configuration management, which is constructed on a client-server architecture. Its repository stores a complete copy of files and directories, which are supposedly under version control. Usually, all these files can be copied to the working directory, which only can be accessed with CVS commands. In the change-log, the version can be marked symbolically. Typically, write access to the CVS repository is extensive, allowing hundreds of developers to add new versions. One can also submit regular patches for moderators to add to the repository. However, in this case, Linux is an exception because it does not use version control tools. Instead, it puts the code for each version in a separate directory and adds the correct one in the latest directory. The moderator can only make additions to the repository, and version history is not kept because they violate the immutability principle in version control. When a new version is created, the latest version will be copied

to the new version directory to keep it as development continues, which means the version is immutable. Meanwhile, Linux kernel development has only two branches: stable release branch and development branch. In projects that use CVS, the version is rarely used to roll back to the previous version. Instead, the version is used for historical tracking, describing how the file was developed by reading log comments and comparing versions using the different functions. Most projects target multiple platforms with significant differences. They seem to deal with changes by breaking the code into different files or directories or using conditional compilation (Raymond, 2000).

These days, each software program inside corporations generally consists of a model management system, that debugs, reports malicious programs, and that helps migrating current source code to an interface, each from a user and company perspective (Di Cerbo et al., 2007).

3.2.2 Build Management

The fact that a local workspace is created containing all the required files makes build management easier. The system model is present in these files, and all projects use or make similar compilation tools. The reconstruction project is not very time consuming as in the changed local workspace, but the initial build is an entirely new process and hence, very time-consuming. Nevertheless, it can be achieved if the object code is considered when creating the workspace. The disadvantage is that the workspace creation speed is much slower.

3.2.3 Configuration Selection

The latest version of all files is almost always used to create the configuration, so the selection is trivial. Since only one (latest) version is saved, you do not have to revert to other settings. In supporting both the staging version and the production version, they work as two separate projects. Branches are rarely used for contemporary work; therefore, no configuration selection is required. For new versions, a new configuration link is created by adding tags to the configuration version. In some projects, a large number of possible configuration options cause problems for developers. Specific changes disrupt the configuration and need feedback to solve the problem. The obvious solution is to have limited security settings instead of all possible combinations.

3.2.4 Workspace Management

The version control tool (CVS) used is optimised to support the accessibility of OSS projects. It supports a project concept that enables a single operation of creating a workspace and synchronizing the workspace with the repository.

However, the most crucial feature of CVS is that it can run in a client-server model, which reduces the burden on developers. To transfer files over the Internet, one needs to stay connected at all times. CVS can disconnect and make all changes offline and reconnect only when synced and added to the repository. In some cases, developers do not have to write access to CVS and some distribution repositories. In these cases, regular corrections are required. First, manually create and send to the host or coordinator, and then they must apply it to the repository.

3.2.5 Concurrency Control

CVS optimizes concurrency control as the file lock function is not used when copying from the repository. CVS can detect when parallel changes are made to the same file and force the final developer to add changes to resolve the conflict. In most cases, it is sufficient to use the update process to automatically merge the first change into the workspace of the second developer. Then he can ensure that the changes made before will not harm his work, as CVS cannot automatically merge. Despite the rapid development and many developers with write access to the repository, update conflicts are infrequent. It is used to raise awareness and reduce the risk of conflicts that are difficult to integrate. With concurrency control, when the host reviews the received posts, they are sorted, and if later post conflicts with an earlier change, the post will be returned, and the contributor will be prompted to resubmit the patch. Moderators only make necessary changes if the conflict can be quickly resolved.

3.2.6 Change Management

In OSS, the evaluation of proposed changes is not precise. Anyone can propose changes, and in most cases, they will not even be proposed until they are submitted directly. Thus, the project cannot give developers any tasks, and everyone is doing what they want. There are two slightly diverse methods, depending on whether there is a need to send content to the moderator or write permissions to write changes directly to the repository. However, the same general procedure is followed. The change is conceived, implemented, tested, and submitted as a fix or directly applied to the repository and finally implemented. The evaluation may result in the facilitator rejecting the fix or the coordinator checking the changes into the repository (Feller & Fitzgerald, 2000).

When we talk about open-source distributions, Linux is the best example of an OS project that has been audited. It receives submitted patches and is processed by a moderator. The fix is checked in several stages before testing and then transferred to the repository or rejected in the initial stages. If the idea is good, but the code is terrible, the post usually goes through several verification steps

before testing.

Projects like Apache, Kubernetes and OpenStack use a much hybrid approach of direct-write access by module owners and developers. The owner of the module has the right to reject the patch. In most cases, any developer with write access to the repository can change most of the code. There are no hard and fast rules; instead, each module owner sets its module's addition and modification strategy. Sometimes, the module owner can become a bottleneck when editing a post. However, in projects that work entirely through coordination, most changes seem to be accepted immediately, and few, if any, are rejected (Kafka, 2121).

Most change management problems seem to fall into the review phase. Codes are usually verified through code reviews and quick run-time tests. However, formal evidence is not always used. Sometimes developers continue to use new code after making changes. Developers who have submitted many good fixes are much more reliable, and their contributions enter the repository faster. Although change requests and error lists are supported, errors and suggested changes seem to be fixed casually.

Important CM Factors

This section will analyze some change management factors and how they affect the OS and its distributions. We divide the analysis into two categories: tools and process (Asklund & Bendix, 2001).

Tools. As with all software projects, many tools are used. CVS is used as a configuration management tool in a typical OS project and standard tools such as mail and web browsers. One server has many clients, so the server does not need to be synchronized, only the client workspace. Since all developers must learn the tool themselves, it must be easy to use and manage rather than prescribe a specific process. Tools that support this model and make it easier to update the workspace from the server are essential. Unfortunately, many tools use one model when the client is online and another when the client is offline, complicating the developer's work.

Process. The process must be simple and easy to follow. An overly rigid process may increase personal investment without increasing the developer's profits, making the whole process complex. A good example is the lengthy transaction model, which encourages frequent commitments, reduces merger conflicts, and increases awareness. Frequent submissions also mean short iterations, which seems to be a good strategy. The key is to identify the weakest link in the process so that any developer can bear social pressure to follow the process and guidelines provided for the project. The process should be suitable for the task. Instead, all development, including bug fixes and new requirements, is passed directly. Otherwise, multiple branches must be maintained, which can now be avoided.

The combination of self-directed tasks, simple processes, stimulating discussions, direct communication, and group awareness is essential for educating and motivating developers. In addition, they often find it exciting and challenging to discuss technical solutions with other well-trained developers.

4 Conclusion

To summarise the entire process, the OSS distribution system is distributed in software packages, such as RedHat/Fedora or Debian, which are built from source code, distributed under various licenses that require the installation of libraries or components (Attilio et al., 2006). An essential part of this research is to improve the conceptual framework and demonstrate the elements shown by these studies that will lead to the successful emergence of OS distributions in organisations and technology development.

Understanding the development of open-source distributions is essential, and so the author has classified distributions in different stages and types. They are followed by different challenges, such as software and hardware issues. Further challenges are related to performance, license and security risks. When we talk about software challenges, the biggest drawback of OS distributions is server and driver support. There is no unified configuration system for development and release management and no standard API. Many settings are to be done manually, and there is no package manager taking care of these things, which is challenging for a new user. In the case of organisations wanting to adopt OS distributions, the most significant resistance is that different distributions can use different versions of library and compiler flags, resulting in much confusion for organisations and their users. Even if there is the desired support, the next challenge is the compatibility issue, which is an essential factor in why many organisations are still reluctant to adopt OS distributions. This compatibility issue is one of the most critical challenges, as many peripheral devices, gadgets and tools are either poorly supported or not supported at all. In order to overcome these technical challenges, it is essential to learn from them and come up with solutions that may have a positive impact on the production of open-source projects (Reffell, 2021).

Several other issues must be considered when adopting OS in an organisation, such as licenses and security risks. Other topics include standardisation, improving good business practices, and acquiring the skills to communicate effectively in developing OSS projects. A large number of licenses, the accompanying restrictions, and the complex dependencies between packages create a situation in which

it is difficult to understand the conditions under which packages can be used and distributed. Although software distributions such as GNU Linux distributions perform their license checks, this is usually not enough.

All these technical challenges enhance the role of a unified configuration system from development to release of open-source distributions. We understand the configuration management of OS distributions, with the following goals (Panda, 2021):

- Make the change management procedures more project-specific and easy to understand.
- Communication of developers with clients simultaneously and work in a distributed and autonomous manner, rather than separate servers.
- The project coordinator protects the codebase, and the task is to have the ability to reduce the risk of bottlenecks in distribution delivery methods.

This research paper attempts to illustrate the importance of reasonable security measures, especially when releasing open-source distributions. Distributed open-source applications are very vulnerable to attacks due to poor development and configuration errors. In addition, many external threats and their attack frequency significantly impact the overall security footprint. This research provides an updated and structured understanding of software development methods and engineering challenges in OSS projects based on systematically collected literature.

5 Future work

We have identified some possible research areas for future work. First, this research needs to be repeated on a larger sample of projects, which will help increase the quality and verify these results. Finally, based on this literature review, the direction could be exploring other fields beyond the operating systems, such as large-scale commercial configurators, and comparing the results with open-source projects. This comparison could be fascinating. In addition, configuration management tools could also be integrated into more distributions and improving the current framework, such as turning it into a more integrated and user-friendly tool. As future work, we aim to apply our research results to future OSS projects and verify the proposed configuration management process from a practical perspective, including the activities of OS distributions, their characteristics, and the entire OS ecosystem.

Apart from laying down the principles of performing qualitative data analysis using MAXQDA, this paper also shares the entire literature survey method, which could support future studies regarding systematic literature review. A good understanding of OS distribution challenges can help professionals prepare to take the appropriate steps to address them. This article presents the results of our study to systematically identify and summarise the challenges that come with open-source distributions. We believe these findings can be equally valuable for practitioners and researchers. Professionals will become more aware of the potential challenges, and the research community can deliberate and discuss potential causes and solutions to the challenges summarised. In addition, references to the literature may benefit researchers interested in future research on this topic.

6 Limitations

Like every other research, this thesis is also subject to limitations and constraints. From the start, the central focus has been set on analysing the literature and discuss the findings. However, as an open-source phenomenon is still new to many people and firms, this topic is not very well researched, due to which many dependencies were on the grey literature. Moreover, most of the literature on this topic is very 'distribution' specific, and there are cases where challenges of every distribution differ due to their architectural framework or development of packages. As this thesis is not supposed to be about any particular software distribution, we decided to go with distributions with the most literature available. Therefore, labelling these as common challenges for all open-source distributions will not be entirely correct as some could easily differ. Instead, the author has tried to lay down a framework and knowledge to get into the topic and understand why it is necessary to dive deeper and explore the field of open-source software and its distributions.

Although we have performed a rigorous literature search, we may have inadvertently excluded studies due to the subjectivity of our inclusion and exclusion criteria. Our classification of challenges is necessarily subjective. However, our intention is not to present a definitive ranking; instead, we intend to present our findings in a structured way that can help practitioners using open-source distributions in product development and software engineering.

7 Extension Chapter

This chapter is the extension of this thesis, where we discuss the role of integrating packages in OS distributions and a summary of the Linux kernel. As Linux distributions being the primary focus of our research, it is essential to understand the Linux kernel and its characteristics.

7.1 Packages

This section discusses the role of packages in developing open-source distributions and the challenges they bring along. As the scale and scope of many open-source distributions continue to grow, variable software and hardware make it very complex and non-trivial to reuse the code. Additionally, the size of the code can also be intimidating, as the user must have in-depth knowledge of the code. Furthermore, software architectures must also support large-scale software integration efforts. To meet these challenges, many researchers have already created an extensive framework to manage complexity and facilitate rapid software prototyping for experiments, resulting in many software systems currently used in academia and industry (Breiling et al., 2017).

We see that the work of developers is substantial, so attempts to automate some of these tasks, such as automated dependency extraction tools or getting source code updates from developers, can be crucial. In our example, we consider a distribution based on Debian, with two maintainers and an RPM-based distribution with one maintainer. This consideration is since RPM-based distribution allows simultaneous installation of several packages, whereas Debian distribution does not and works in a binary manner. This distribution is divided into one or more units, compiled for the different architectures supported by the distribution and grouped into packages. Control files and data specify how the product is divided into units, how each unit should be compiled and packaged, architectures with information on dependencies, and their classification. These packages are then downloaded by this distribution's user package management software. Some users may prefer to download the sources directly from the developers, in which case they will usually run a sequence of commands like */configure* to install and

compile the software. However, they lose many of the benefits of a package management system, such as monitoring the files and automated updates (Di Cosmo et al., 2006).

Integrating a package

We studied some in-depth analysis of package integration for any open-source distribution. Integrating a package is not as easy as integrating any other software. For some users, it could be pretty challenging. First, the user has to download, for example, the Debian based operating systems package from the repository, which is the default build tool. The package must then be installed, configured and launched. Next, parameters, nodes, device identifiers, and different features must be defined to start the service successfully. Finally, once the project is running, the user can monitor the data flow and status, deciding whether the new package behaves as expected or should be reconfigured or removed (Estefo et al., 2019).

An analysis of the information gathered revealed the following challenges with regards to packages. These challenges affect the resilience of any open-source distribution:

- Package could be for an outdated version or package available for a newer version that was not compatible with the current device.
- Users could not figure out how to use it properly as reusing packages is not as easy as it seems. Bugs and lack of essential documentation is a significant drawback.
- The package is not compatible with the particular hardware, and many packages end up being abandoned by their developers due to a lack of support.
- The absence of an active package maintainer affects the normal flow of contributions.
- Many package developers are unaware of the workflow contribution.

To address these challenges, (Estefo et al., 2019) suggests some recommendations. The purpose of these guidelines is to minimise the consequences of package bottlenecks.

1. Identify and predict abandoned packages

The reuse of packages is one of the most popular assets of open-source distributions. However, according to the survey (Estefo et al., 2019), the majority of users do not reuse packages. In many cases, this is because their maintainers have discontinued packages. As a result, users can waste a lot of time and effort trying to install or configure such a package before realising it has been abandoned.

When users find packages that cannot be reused and contributions ignored, the open-source distribution ecosystem becomes less reliable.

2. Provide a repository of information packages

Popular and mature software ecosystems such as the LATEX, or programming languages such as R and JavaScript, rely on a comprehensive and informative catalogue of packages available for reuse. According to a survey (Estefo et al., 2019), users are looking for the following information about packages that can be reused:

- Purpose and characteristics of the package
- Installation process and dependencies
- Package development status
- The maintainer of the package
- Package configuration guidelines

Users also want more information about the packages available, such as troubleshooting experiences, recent development and usage activities, or alternative packages. As a result, the user will be able to make a more informed decision in choosing which packages to rely on in their projects.

3. Recommend contribution opportunities to qualified members

Recommendation systems in software engineering can help deliver information from significant data sources that one person cannot perform due to the size, heterogeneity, and processing complexity. Referral systems can designate community members who are particularly best suited to make some contribution to the ecosystem. For example, they can identify experts to answer a popular unanswered question, confirm bugs associated with hardware, or handle specific maintenance tasks from a discontinued package. However, since the open-source community is relatively minor, it is more susceptible to information scarcity issues. In addition to finding a qualified community member to suggest a contribution, we suggest fragmenting a regular contribution (e.g. bug fix, bug report, and documentation update). Such micro-tasks would cost less time and effort by reducing the barriers for contributors. A system similar to a problem tracker could track the contribution status, inform about the effort estimate, and the current status of the distribution (Estefo et al., 2019).

4. Limit substantial changes

In recent times, we consider the example of ROS Kinetic, which is a ROS distribution. It resulted in substantial changes in the build system API and some core functions that needed to be updated by maintainers. The result was that one-third of the Packages available in the previous version, ROS Jade, no longer

worked on ROS Kinetic. Such noticeable changes can seriously affect any packages already available. This situation is hazardous for packages that have already been dropped because someone is unlikely to update them. Maintainers will take a while to respond or may not respond at all. A side effect is that packages that depend on discontinued packages may be out of date if their dependencies cannot be upgraded. The need to use outdated packages was the most common reason the survey respondents could not reuse a package. Another possible scenario is the other way round, where a user is stuck with an older version, possibly because a discontinued package prevents them from upgrading to a newer version and can not use a package that works (Mancinelli et al., 2009).

Quality control. Let us now come to the problem of ensuring the quality of a distribution and its packages. This can be divided into three main tasks (Di Cosmo et al., 2006):

- Upstream monitoring ensures that the distribution package closely follows the evolution of software development.
- Testing and integration, so the program works as expected in conjunction with other packages in the distribution.
- Dependency management ensures that packages can be installed and upgraded when new versions are produced, respecting the constraints imposed by dependency metadata.

The lack of continuous quality support also leads to a decline in distribution releases, which results in users and organisations not wanting to adopt new OS distributions.

7.2 Linux kernel

A lot of the software related issues discussed in the Results chapter, are found in Linux distributions. For instance, in Linux distributions, the two most popular open-source desktops, KDE and Gnome, can only configure some settings themselves, so each release builds its manual application to configure settings. As Linux is the most commonly used, it is essential to discuss the Linux kernel as it is the fundamental component of the Linux operating system and the interface between computer hardware and its processes. It exchanges data between them and manages resources most effectively. A lot of challenges in Linux distributions are dependent on the kernel due to the architecture type of how Linux is build.

Much research has been done to study the evolution mode of Linux, including size growth, changes in kernel structure, and community characteristics. Linux is a UNIX-like operating system that provides computer users with a free system comparable to traditional UNIX systems like IBM AIX or Sun Solaris (Phillips,

2003). The script is mainly based on a selection of proprietary solutions and suitable implementation tools. Linux kernel performs the job of memory management, process management, device drivers, and system security calls (IBM, 2021). The kernel runs on its own, where it allocates memory and keeps track of where everything is stored. The applications interact with the kernel through the system call interface (Hertel et al., 2003). Since every Linux distribution contains core packages and can be compiled to meet almost any requirement, the user needs to determine if the distribution is supported by other software, and will run on the hardware. Linux also works on new Macs, but not possible to use it for some of the older ones with bus technology. Some Linux distributions can be extraordinarily tough and bulky to debug, even for those who develop them. Several regressions occur in the kernel due to the lack of ability of developers to check their modifications and test them on all viable software programs along with their updates (RedHat, 2019).

Eventually, it would have been incomplete to discuss challenges of OS distributions without understanding Linux kernel. It is also critical to understand the role of packages in development and release of open-source distributions.

Appendices

A List of figures

References

Acuna, S., Castro, J., Dieste, O. & Juristo, N. (2012). A systematic mapping study on the open source software development process. *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, 42–46. https://doi.org/10.1049/ic.2012.0005

Adams, B. (2016). An empirical study of integration activities in distributions of open source software, 42.

Agerfalk, P. J., Deverell, A., Fitzgerald, B. & Morgan, L. (2005). Assessing the role of open source software in the european secondary software sector: A voice from industry [Please cite as:Agerfalk, P, Deverell, A, Fitzgerald, B and Morgan L (2005) Assessing the Role of OSS in the European Secondary Software Sector, 1st International Conference on Open Source Software, Genoa, Italy, July 2005]. *1st International Conference on Open Source Software*. https://mural.maynoothuniversity.ie/6641/

Alam, K. & Soomro, T. R. (2016). SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES), 5.

Androutsellis Theotokis, S. (2010). Open source software: A survey from 10,000 feet. *Foundations and Trends® in Technology, Information and Operations Management*, *4*(3), 187–347. https://doi.org/10.1561/0200000026

Ansari, M. S. & Chaubey, A. K. (2014). Library automation & open source software, 15.

Asklund, U. & Bendix, L. (2001). Configuration management for open source software, 14.

Attilio, F., Di Nunzio, P., Di Gregorio, F. & Meo, A. R. (2006). A graphical installation system for the GNU/linux debian distribution [Series Title: IFIP International Federation for Information Processing]. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto & G. Succi (Eds.), *Open source systems* (pp. 337–338). Springer US. https://doi.org/10.1007/0-387-34226-5_35

Boender, J. (2012). A formal study of free software distributions, 141.

Bonaccorsi, A., Lorenzi, D., Merito, M. & Rossi-Lamastra, C. (2007). Business firms' engagement in community projects. empirical evidence and further developments of the research. *First International Workshop on Emerging*

Trends in FLOSS Research and Development, FLOSS'07, 13–13. https://doi.org/10.1109/FLOSS.2007.3

Breiling, B., Dieber, B. & Schartner, P. (2017). Secure communication for the robot operating system. *2017 Annual IEEE International Systems Conference (SysCon)*, 1–6. https://doi.org/10.1109/SYSCON.2017.7934755

Cesar Brandão Gomes da Silva, A., de Figueiredo Carneiro, G., Brito e Abreu, F. & Pessoa Monteiro, M. (2017). Frequent releases in open source software: A systematic review. *Information*, *8*(3), 109. https://doi.org/10.3390/info8030109

Chau, P. Y. K. & Tam, K. Y. (1997). Factors affecting the adoption of open systems: An exploratory study. *MIS Quarterly*, *21*(1), 1. https://doi.org/10.2307/249740

Clark, B. (2004). *Distribution of software*. https://docplayer.net/10534571-Distribution-of-software.html

Constantinescu, M. (2019). The enterprise challenge to OpenStack adoption and how to address it, 2.

Copeland, B. (2015, December 15). *Enterprises and open source: The important role of commercial distributions*. https://www.activestate.com/blog/enterprises-and-open-source-important-role-commercial-distributions/

Copeland, B. (2016, January 6). *Enterprises and open source: The important role of commercial distributions*. https://thenewstack.io/enterprises-open-source-important-role-commercial-distributions/

D2iQ. (2021). *Limitations of kubernetes*. https://docs.d2iq.com/mesosphere/dcos/services/kubernetes/2.4.9-1.15.10/limitations/

Dedrick, J. & West, J. (2004). An exploratory study into open source platform adoption. *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, 10 pp. https://doi.org/10.1109/HICSS.2004.1265633

Di Cerbo, F., Scotto, M., Sillitti, A., Succi, G. & Vernazza, T. (2007). Toward a GNU/linux distribution for corporate environments: In S. K. Sowe, I. G. Stamelos & I. Samoladas (Eds.), *Emerging free and open source software practices* (pp. 215–236). IGI Global. https://doi.org/10.4018/978-1-59904-210-7.ch010

Di Cosmo, R., Durak, B., Leroy, X. & Mancinelli, F. (2006). Maintaining large software distributions: New challenges from the foss era.

Diener, D. (2018, September 18). *How to upgrade fedora linux*. https://www.addictivetips.com/ubuntu-linux-tips/upgrade-fedora-linux/

Estefo, P., Simmonds, J., Robbes, R. & Fabry, J. (2019). The robot operating system: Package reuse and community dynamics. *Journal of Systems and Software*, *151*, 226–242. https://doi.org/10.1016/j.jss.2019.02.024

Feller, J. & Fitzgerald, B. (2000). A FRAMEWORK ANALYSIS OF THE OPEN SOURCE DEVELOPMENT PARADIGM, 13.

Foundjem, A. & Adams, B. (2021). Release synchronization in software ecosystems, 52.

German, D. M., Di Penta, M. & Davies, J. (2010). Understanding and auditing the licensing of open source software distributions. *2010 IEEE 18th International Conference on Program Comprehension*, 84–93. https://doi.org/10.1109/ICPC.2010.48

Hauge, Ø., Ayala, C. & Conradi, R. (2010). Adoption of open source software in software-intensive organizations – a systematic literature review. *Information and Software Technology*, *52*(11), 1133–1154. https://doi.org/10.1016/j.infsof.2010.05.008

Hauge, Ø., Sørensen, C.-F. & Røsdal, A. (2007). Surveying industrial roles in open source software development [Series Title: IFIP — The International Federation for Information Processing]. In J. Feller, B. Fitzgerald, W. Scacchi & A. Sillitti (Eds.), *Open source development, adoption and innovation* (pp. 259–264). Springer US. https://doi.org/10.1007/978-0-387-72486-7_25

Hecht, L. E. & Clark, L. (2018). *Survey: Open source programs are a best practice among large companies*. https://thenewstack.io/survey-open-source-programs-are-a-best-practice-among-large-companies/

Hertel, G., Niedner, S. & Herrmann, S. (2003). Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel. *Research Policy*, *32*(7), 1159–1177. https://doi.org/10.1016/S0048-7333(03)00047-7

Hurley, D. (2014, June 24). *12 challenges for open source projects*. https://opensource.com/life/14/6/12-challenges-open-source-projects

IBM. (2021). *Major linux problems on the desktop*. https://www.tech-insider.org/linux/research/acrobat/010321

Kafka, A. (2121, April 6). *Top 10 problems when using apache kafka*. https://pandio.com/blog/top-10-problems-when-using-apache-kafka/

Kahani, N., Bagherzadeh, M., Dingel, J. & Cordy, J. R. (2016). The problems with eclipse modeling tools: A topic analysis of eclipse forums. *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, 227–237. https://doi.org/10.1145/2976767.2976773

Kaur, R., Kaur Chahal, K. & Saini, M. (2020). Understanding community participation and engagement in open source software projects: A systematic mapping study. *Journal of King Saud University - Computer and Information Sciences*, S1319157820305139. https://doi.org/10.1016/j.jksuci.2020.10.020

Khan, M. A. & UrRehman, F. (2012). Free and open source software: Evolution, benefits and characteristics. *1*(3), 7.

Kitchenham, B. A. & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (tech. rep. EBSE 2007-001).

Keele University and Durham University Joint Report. https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf

Kogut, B. (2001). Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, *17*(2), 248–264. https://doi.org/10.1093/oxrep/17.2.248

Kohgadai, A. (2020, October 1). *Four container and kubernetes security risks you should mitigate*. https://www.stackrox.com/post/2020/10/four-container-and-kubernetes-security-risks-you-should-mitigate/

Lehmann, R. (2017). 4 OpenStack monitoring challenges, tips & tricks, 4.

Library, B. U. (2021). *Guide to searching*. https://libguides.brown.edu/searching/citation

Mancinelli, F., Boender, J., di Cosmo, R., Vouillon, J., Durak, B., Leroy, X. & Treinen, R. (2006). Managing the complexity of large free and open source package-based software distributions. *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, 199–208. https://doi.org/10.1109/ASE.2006.49

Mancinelli, F., Boender, J., di Cosmo, R., Vouillon, J., Durak, B., Leroy, X. & Treinen, R. (2009). Managing the complexity of large free and open source package-based software distributions. *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, 199–208. https://doi.org/10.1109/ASE.2009.49

Martin, K. & Hoffman, B. (2007). An open source approach to developing software in a small organization. *IEEE Software*, *24*(1), 46–53. https://doi.org/10.1109/MS.2007.5

MAXQDA. (2021). *Maxqda in research*. https://www.maxqda.com/

Mockus, A., Fielding, R. T. & Herbsleb, J. (2000). A case study of open source software development: The apache server. *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, 263–272. https://doi.org/10.1145/337180.337209

Morgan, L. & Finnegan, P. (2007). How perceptions of open source software influence adoption: An exploratory study, 13.

Napoleao, B. M., Petrillo, F. & Halle, S. (2020). Open source software development process: A systematic review. *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, 135–144. https://doi.org/10.1109/EDOC49727.2020.00025

Nasserifar, J. (2016). *Open source software ecosystem: A systematic literature review* (Doctoral dissertation). https://doi.org/10.13140/RG.2.1.2254.1049

Osterloh, M. & Rota, S. (2007). Open source software development—just another case of collective invention? *Research Policy*, *36*(2), 157–171. https://doi.org/10.1016/j.respol.2006.10.004

Panda, D. (2021). The top challenges of implementing continuous delivery with kubernetes, 5.

Perens, B., Sroka, M. & Stu, M. (1998). The open source definition, 9.

Peters, T. (2004). *Distribution of software*. https://docplayer.net/7506702-Choosing-an-open-source-license.html

Phillips, D. (2003). Computer music and the linux operating system: A report from the front. *Computer Music Journal*, *27*(4), 27–42. https://doi.org/10.1162/014892603322730488

Pritchett, B. (2020). What we've learned using OpenShift container platform in a hybrid cloud environment for red hat IT, 3.

Rajagopalan, R. (2020, July 18). *Challenges and requirements for container-based applications and application services*. https://cloud.redhat.com/blog/challenges-and-requirements-for-container-based-applications-and-application-services

Randhawa, S. (2008). Open source software and libraries, 10.

Raymond, E. S. (2000). The cathedral and the bazaar, 59.

RedHat. (2019). *What is the linux kernel?* https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel

Reffell, C. (2021, March 23). *10 business sectors where top open source platforms have an impact*. https://crowdsourcingweek.com/blog/10-business-sectors-where-top-open-source-platforms-have-an-impact/

Riehle, D. (2009). The commercial open source business model [Series Title: Lecture Notes in Business Information Processing]. In M. L. Nelson, M. J. Shaw & T. J. Strader (Eds.), *Value creation in e-business management* (pp. 18–30). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03132-8_2

Riehle, D. (2021, May 29). *Open source distributions by life-cycle*. https://dirkriehle.com/2021/05/29/open-source-distributions-by-life-cycle/#more-14848

Sardina. (2019, April 16). *The enterprise challenge to openstack adoption and how to address it*. https://faun.pub/the-enterprise-challenge-to-openstack-adoption-and-how-to-address-it-dae983bbb1de

Shiozaki, M. (2016, October 14). *The top 3 openstack benefits and challenges*. https://www.stratoscale.com/blog/openstack/the-top-3-openstack-benefits-and-challenges/

Silakov, D. (2008). Linux distributions and applications analysis during linux standard base development. https://doi.org/10.15514/SYRCOSE-2008-2-2

Stallman, R. & Free Software Foundation (Cambridge, M. (2015). *Free software free society: Selected essays of richard m. stallman* [OCLC: 927962245].

Staples, M. & Niazi, M. (2006). Experiences using systematic review guidelines. https://doi.org/10.14236/ewic/EASE2006.9

Stol, K.-J. & Ali Babar, M. (2010). Challenges in using open source software in product development: A review of the literature, 17–22. https://doi.org/10.1145/1833272.1833276

Tashkinov, A. S. (2021). *Major linux problems on the desktop*. https://itvision.altervista.org/why.linux.is.not.ready.for.the.desktop.current.html

Thomas. (2019, February 19). *The benefits and challenges of building an openstack based cloud*. https://www.eurovps.com/blog/openstack-cloud-benefits-challenges/

Tozzi, C. (2020, August 5). *8 problems with the kubernetes architecture*. https://www.itprotoday.com/hybrid-cloud/8-problems-kubernetes-architecture

Unterkalmsteiner, M., Gorschek, T., Islam, A., Cheng, C., Permadi, R. & Feldt, R. (2011). Evaluation and measurement of software process improvement—a systematic literature review. *IEEE Transactions on Software Engineering*, *38*, 398–424. https://doi.org/10.1109/TSE.2011.26

van der Linden, F., Lundell, B. & Marttiin, P. (2009). Commodification of industrial software: A case for open source. *IEEE Software*, *26*(4), 77–83. https://doi.org/10.1109/MS.2009.88

Ven, K. & Mannaert, H. (2008). Challenges and strategies in the use of open source software by independent software vendors. *Inf. Softw. Technol.*, *50*(9–10), 991–1002. https://doi.org/10.1016/j.infsof.2007.09.001

Ven, K., Verelst, J. & Mannaert, H. (2008). Should you adopt open source software? *IEEE Software*, *25*(3), 54–59. https://doi.org/10.1109/MS.2008.73

von Krogh, G. & von Hippel, E. (2006). The promise of research on open source software. *Management Science*, *52*(7), 975–983. https://doi.org/10.1287/mnsc.1060.0560

Weikert, F. & Riehle, D. (2013). A model of commercial open source software product features. *Software business. from physical products to software services and solutions* (pp. 90–101). Springer.

Wikipedia: NVIDIA. (2021). Nvidia — Wikipedia, the free encyclopedia [[Online; accessed 10-October-2021]]. https://en.wikipedia.org/w/index.php?title=Nvidia&oldid=1048769084

Wikipedia: Xorg. (2021). X.org server — Wikipedia, the free encyclopedia [[Online; accessed 10-October-2021]]. https://en.wikipedia.org/w/index.php?title=X.Org_Server&oldid=1048715352

Yu, L. (2007). Understanding component co-evolution with a study on linux. *Empirical Software Engineering*, *12*(2), 123–141. https://doi.org/10.1007/s10664-006-9000-x