Exploring the Role of Linux in Accelerating Time-to-Market for Embedded Systems

A Mixed Methods Approach

PAPER WITHIN: *Computer Science, Embedded Systems*
AUTHORS: Jesper Persson, Josua Alexandersson
TUTOR: *Jérõme Landré*
JÖNKÖPING May 2023

This exam work has been carried out at the School of Engineering in Jönköping in the embedded systems program. The work is a part of the three-year Bachelor of Science in computer engineering. The authors take full responsibility for opinions, conclusions and findings presented.

Examiner: Ragnar Nohre

Supervisor: Jérôme Landré

Scope: 15 credits

Date: 2023-05-31

Fel! Använd fliken Start om du vill tillämpa Heading 1 för texten som ska visas här.

1 Abstract

How can Linux reduce the time required for development in embedded systems, and what makes Linux appealing to embedded developers despite the loss in overall control? Through qualitative interviews with industry professionals and a systematic literature review, challenges and benefits of using Linux in embedded systems development were identified and discussed. Three hypotheses were formulated based on recurring topic agreement among the interview subjects: Reduced development time through the use of open-source solutions, struggles with real-time and security requirements, and challenges within troubleshooting and dependency management. The empirical data observed primarily aligned with the professional perception indicating the potential for development time reduction leveraging resources properly. However also highlighting additional challenges that are not present in traditional embedded system development. Several trade-offs were observed from the findings, including increased overhead and licensing concerns. Further research is required to fully understand the advantages, challenges and limits associated with Linux in an embedded system environment. This study provides valuable insights for future exploration within the field.

Keywords

Embedded System, Linux Operating System, Real-time Operating System, Software Development, Development Time Reduction

Fel! Använd fliken Start om du vill tillämpa Heading 1 för texten som ska visas här.

2 Contents

1 Abstract 1

2 Contents 2

3 Summary 4

4 Introduction 5

4.1 Background 5

4.2 Purpose and research questions 5

4.3 Scope 6

4.4 Delimitations 6

4.5 Report Disposition 7

5 Theoretical background 8

5.1 Grounded Theory 8

5.2 Qualitative Interviews 9

5.3 Systematic Literature Review 9

6 Method and implementation 10

6.1 The Mixed Methods Approach 10

6.2 Preliminary Interviews and Hypothesis Generation 10

6.3 Member Checking Process 11

7 Findings and analysis 12

7.1 Interview Findings 12

7.2 Hypothesis Generation 15

7.3 Systematic Literature Analysis of Hypotheses 16

7.3.1 Development Time and Access to Open-Source Software 16

7.3.2 Real-Time Requirements and Safety 17

7.3.3 Troubleshooting and Debugging in Linux 18

8 Discussion and Further Research 20

8.1 Comparison Between Interview Findings and Literature 20

8.1.1 Libraries 20
8.1.2 Real-Time Requirements 21
8.1.3 Troubleshooting and Debugging 21
8.2 IMPLICATIONS FOR FUTURE RESEARCH 22
8.2.1 Libraries and Open-Source Software 22
8.2.2 Real-Time Requirements 23
8.2.3 Troubleshooting and Debugging 24
9 Conclusions 25
10 References 26
11 Appendices 28
11.1 INTERVIEW GUIDELINE 28
11.2 SYSTEMATIC LITERATURE REVIEW SEARCH STRATEGIES 30

3 Summary

This thesis is organized into five main sections to provide a comprehensive understanding on the research conducted

The introduction chapter presents the reader with the scope, purpose and background of the study, as well as its research questions. It also contains an overview of the disposition, detailing subsequent chapters.

The theoretical background chapter explores relevant existing literature and theory that laid the foundation to the research questions and study design. It discusses the grounded theory approach and its suitability to explore processes and complex aspects. Additionally, the use of qualitative interviews is highlighted as its open-ended and explorative nature naturally aligns with grounded theory. Finally, this chapter introduces the use of a systematic literature review and its role of examining the extent to which empirical evidence can support or contradict the hypotheses that were extracted from the interviews.

The method and implementation chapter delves into the mixed-method approach employed in the study. It explains the use of semi-structured interviews with industry professionals to gather data and generate hypotheses. Thematic analysis and grounded theory are employed as analytical methods to derive hypotheses from the interview data. Lastly this chapter touches on the member checking process which is held to improve the validity and accuracy of the authors' interpretation of the interview responses.

The findings and analysis chapter presents the outcome of the semi-structured interviews and the systematic literature reviews applied to the hypotheses presented. It provides a comprehensive analysis of the data gathered and offers insights to help answer the research questions.

Finally, the discussion and conclusion chapters explore potential challenges and implication for future research in regards to the development of embedded Linux systems. The chapters reflect on gaps in understanding and proposes directions for further research within the field. The conclusion chapter summarizes the study's findings and presents the answers to the research questions presented in the introduction.

4 Introduction

The study is using mixed methods of qualitative and quantitative data analysis. The qualitative data will be collected through semi-structured interviews and coded into themes through thematic analysis. Which will lay the foundation for generating the hypotheses through grounded theory. The hypotheses will then be validated through empirical data found in a systematic literature review. Concluding, the study will investigate if Linux has the potential to reduce the time to market for new embedded systems. It will also highlight shortcomings that prevent Linux from being used within the industry.

4.1 Background

The demand for advanced functionality in embedded systems has increased in recent years. Many of these systems still use traditional microcontrollers in their development (VDC Reasearch, 2018).

Developing advanced functionality on a microcontroller-based system can be a more time consuming and costly process compared to developing it on a Linux based system. Another potential drawback of traditional microcontroller-designs is the inherent use of low-level languages. These languages are considered to require a more time-consuming process to produce functionality compared to high-level languages that would be available in a Linux system, low-level languages also offer more limited access to third party libraries.

An embedded Linux system also presents drawbacks however, the complexity of the system makes it less reliable for real-time requirements and many developers are already hesitant to use third party software because of licensing issues.

To address these issues, this research aims to explore the potential of Linux in improving time-to-market speed for embedded systems. The objective is to identify and analyze benefits and challenges associated with the utilization of Linux within embedded real-time systems. Contributing valuable insights to the existing knowledge base of embedded Linux systems.

4.2 Purpose and research questions

The primary goal of this study is to understand how access to Linux can reduce the development time of an embedded system. Linux has both the benefit and drawback of being a more complex system than embedded development is normally utilizing. A benefit that comes with the complexity is that it has access to a larger suit of potential tools to ease the development, a drawback that comes with the complexity is that provides less low-level control of the system and additional overhead to handle during

development. The study aims to investigate the feasibility of Linux as a real-time operating system and the underlying reasons as to why it is increasingly adopted in embedded systems (Market Reports World, 2023).

Previous research by Ionescu & Enescu (2020) suggests that utilizing higher-level programming languages can lead to shorter development times, although it may come at the cost of reduced system performance. This research seeks to explore the possibility of minimizing development time through the incorporation of more versatile libraries available in higher-level languages while also examining the potential impact on system performance.

To achieve these goals, the following research questions will be investigated:

- How can Linux be used to reduce the time required for development in embedded systems
- What factors make Linux an appealing or less appealing choice for embedded system development?

4.3 Scope

The scope of this research is to investigate the feasibility of reducing development time of embedded system projects in commercial and industrial environments using Linux. Specifically, this study will explore the following aspects:

- The effectiveness of Linux in the development process, including its flexibility and ease of use which is facilitated by the available tools and resources within the Linux community.
- The suitability of Linux as a real-time operating system with respect to its reliability and performance.
- The experiences and perspectives of professional Linux and/or embedded developers to identify strengths, weaknesses, and potential improvements when using Linux as a real-time operating system.
- The current state of the art knowledge within the field through systematic literature analysis of the hypotheses built upon interview findings.

4.4 Delimitations

This research is subject to limitations that should be taken into consideration:

- The study will be limited to professionals with experience of using Linux for software development in industrial or commercial settings. This will include Linux developers and embedded developers. This limitation might obfuscate the generalizability of the results.
- The interviews will be conducted with a small sample size of professionals and may therefore not saturate the full range of perspectives for how Linux can be used to reduce development time.
    - The sample group consisted of a 100% male group
    - Age varied between 25-50+ years old
    - Experience is displayed in Table 2 below
- The study will be limited to the current state of Linux and will not account for any future developments of the technology.
- The study will not cover the entire development process but will focus only on the potential of using Linux to reduce development time.
- The research will not examine the economic or financial aspects of using Linux as an operating system.
- The study will not investigate the impact of hardware or firmware on the performance of Linux as a real-time operating system.

4.5 Report Disposition

This report is structured into five main sections. The introduction chapter provides an overview of the study's scope, purpose, background, and research questions.

The subsequent chapter, theoretical background, delves into the relevant existing literature and theories that form the foundation of the research questions and study design.

The method section outlines the mixed-methods approach employed, encompassing the interviewing process, theory generation, and systematic literature review.

The findings and analysis chapter presents the results obtained from the interviews, systematic literature review, and overall study findings.

Lastly, the final chapter explores these results in greater depth, discussing potential challenges, implications for future research, and the benefits of Linux in development.

5 Theoretical background

5.1 Grounded Theory

The research must contain several characteristics to be a grounded theory approach according to Tarozzi (2020).

- *To generate a theory or conceptual framework*, using empirical data. To develop a theory by adding data and at the same time take a step back and see how the data connects.
- *To explore a process*, where a process is things that happen in order, start to finish.
    - The research questions at hand are connected with the development process, and gaining a comprehensive understanding of its intricacies and potential variations is a key objective that will be facilitated through the conducted interviews.
- T*o employ theoretical sampling*, this is about trying to fill voids in your theory while you develop it.
    - Through the utilization of qualitative interviews in the research, the potential for new questions to arise during the analysis of each interview is introduced. These questions can be further explored in subsequent interviews, and if left unanswered, they can be subjected to additional investigation in the subsequent stages of the research.
- *To collect data and analyse them simultaneously*, this is required to follow the theoretical sampling characteristic. You cannot collect all data and then analyse it.
    - Each subsequent interview is influenced and shaped by the insights gained from the previous interviews, contributing to the iterative development and refinement of the research approach.
- *To use the method of constant comparison of every level of the analysis*, this is about questioning the data and connecting it during the analysis to make sure your understanding of the concept you are investigating is progressing.
    - In the context of conducting and analyzing interviews, constant comparison becomes a natural and essential part of the process. As interviews provide rich and diverse data, continually comparing the responses from different participants, looking for similarities, differences, and emerging themes comes naturally.
- *Research questions using sensitizing concepts*. While moving forward with grounded theory you should create your own categories, while adding to your theory, do not get stuck in what is already established.
    - This issue will be mitigated by developing an independent understanding based on the interview findings before establishing a comparison with existing knowledge.
- *Conceptualization versus description*. It is about developing a theory and not describing how something works. Take another step based on your findings after you have gathered and analysed data.

- *Production of memos and diagrams.* To follow grounded theory, you should be writing notes, later on in your research you will be able to see how your theory was developed. If you are able to show the data in different stages of the process it will further show the development of your theory.

The nature of the research is exploratory, and the final outcomes remain dynamic and subject to evolution throughout the interviews. While striving to fulfill the criteria of a grounded theory approach, it is acknowledged that further investigation is necessary to definitively classify the study as such.

Grounded theory is useful for answering research questions related to processes and complex actions (Tarozzi, 2020). Which is what this research aims to achieve by answering the research questions.

5.2 Qualitative Interviews

Qualitative interviews are well-suited for grounded theory methodology due to their inherent characteristics of being open-ended yet directed (Charmaz, 2014). Grounded theory aims to develop theories that emerge from the data itself, while qualitative interviews provide a rich source of data that allows for in-depth exploration of the topic. The interviews have prepared questions but the answers from each participant can take the discussion in different directions. This fits the purpose since even if there exists prior experience in the area, the interviews aim to learn as much as possible from the participants and the insights that are gleaned from the answers. Also, the interviews aim to provoke the participants into asking their own questions around the subject in order to establish research gaps within the field.

Kommenterad [A1]: Nicely written.

5.3 Systematic Literature Review

"Systematic literature reviews can be undertaken to examine the extent to which empirical evidence supports/contradicts theoretical hypotheses" (Kitchenham & Charters, 2007, para. 12).
From the answers gathered from the interviews, hypotheses are created from the most distinguished topics in the interviews.
Systematic literature reviews are then employed to explore how well the hypotheses hold up against empirical research.

6 Method and implementation

6.1 The Mixed Methods Approach

In this study, the mixed methods approach was used to combine qualitative and quantitative research methods. This approach allowed for gathering of in-depth insights from interviews with industry professionals and validate them against empirical research through systematic literature review.
According to Hesse-Biber (2010) when using the mixed-methods approach the researcher is looking to converge all the data collected by the different methods to enhance the credibility of the research findings. To achieve this enhanced credibility methods triangulation is used to study the same research question.
Another aspect of mixed method research that Hesse-Biber (2010) brings up is complementarity. Which lets the researchers gain a more complete understanding of the research problem and helps clarify the given result.

6.2 Preliminary Interviews and Hypothesis Generation

The qualitative data in this study was obtained from semi-structured interviews with industry experts who have multiple years of professional experience in Linux, embedded systems, or both. Semi-structured interviews were conducted with six experts, each lasting between twenty minutes to an hour and covering a range of open-ended questions designed to elicit insights into their experiences and perspectives. The interview guideline used for the interviews can be found in the appendices: section 11.1. All interviews were held in Swedish and were then translated to English.

The data was analyzed using the thematic analysis method to identify patterns and organize the collected information into manageable parts. Braun & Clarke (2006) describe thematic analysis as a relatively straightforward form of qualitative analysis and explain that it is a suitable technique for researchers new to qualitative methods. They also frequently comment on the flexibility of the method, which increases the adaptability and broadens the range of insights that can be obtained from the data. Given the authors' limited experience with qualitative data analysis and the flexible nature of the method, thematic analysis was chosen.

In addition to thematic analysis, grounded theory was employed for generation of hypotheses. Grounded theory is a systematic methodology that develops theory from the analysis of qualitative data. This approach is also noted for its flexibility. Charmaz (2006) states that even Glaser and Strauss, the widely recognized founders of grounded theory, encouraged readers to use the method flexibly. Charmaz (2006) characterizes grounded theory as a set of principles and practices rather than a rigid set of methodological rules.

This aspect of grounded theory had great appeal, as the semi-structured interviews were designed to elicit open-ended and unpredictable responses from interviewees.

Grounded theory provided the necessary flexibility to generate hypotheses based on the thematic analysis of the interview data.

6.3 Member Checking Process

To ensure validity and accuracy in the understanding of the interview responses, member checking was conducted with the participants. Member checking involves "*asking the participants to go over transcripts of their interviews to comment on accuracy or even to review the researcher's interpretation"* (Josselson, 2013, p.178.). Full transcripts of the interviews were not sent as they are very long, in an attempt to reduce the burden on, and also increased the likelihood the interview subjects would read and respond to the request. Instead extracted codes from the interview were sent, along with a quote from their answer where the code was extracted from. The question that prompted the quoted answer was also. A response was then requested from the interviewees to ensure that the authors interpreted their answers correctly. This was to validate that the codes of their answers fairly represent their point of view.

6.4 Purposive and Snowball Sampling

The interview participants for this study have been selected through purposive and snowball sampling. Purposive sampling means the authors have reached out to professionals they believe have experience and expertise that can contribute to the study. Snowballing sampling means that the interview participants are asked to contribute with suggestions of more potential participants they think can help contribute to the study even further.

7 Findings and analysis

7.1 Interview Findings

The table below (Table 1) summarized the codes extracted from the interviews, capturing the key themes as the code with a short description of the authors' definition and interpretation. An example quote is presented to further understand the meaning behind the codes and give the reader a chance to interpret the interview subjects. The frequency of occurrence shows in how many interviews the theme was observed. The aim of the table is to present a concise and structured overview of the key points raised from participants and establish a foundation to the subsequent analysis and discussion chapters.

Below the key themes is a table (Table 2) providing experience background of the interview participants in order to strengthen the validity of the respondents' answers.

Table 1. *Key Themes and Findings from Semi-Structured Interviews.*

Code	Description	Example Quote	Frequency
Third-Party Libraries	Extending product lifespan, requiring careful management, and potentially affecting product pricing.	"We have used proprietary libraries for some projects, and you are then dependent on the supplier being around and not stopping support. We have libraries that have been compatible with certain versions of another operating system, but they have stopped supporting it for new generations, and we have not been able to upgrade. That's where open-source comes in because you have other options, often with a large community ensuring it stays up to date." - Axel	6/6
Performance	Challenges in optimizing Linux for performance in real-time applications, compared to dedicated real-time/headless OSs.	"We had to have a bunch of real sly Linux foxes trying to optimize everything so that the Linux system would be sufficiently performance-efficient. That was a challenge with it" (using Linux) - Erik	6/6
Safety	Concerns about Linux's security and execution guarantees, weighed against performance and real-time capabilities.	"Yes, there were some question marks from the industry regarding Linux, it was in the automotive industry and there were various safety aspects of it that they thought were very questionable to use Linux and that you could guarantee that things happen in the same way using a Linux system." - Erik	6/6

Development and debugging	Increased complexity in hardware debugging, increased software debugging capabilities in Linux, terminal advantages, and external problems with Linux.	"Usually, it's not the entire Linux system but rather your application, that goes a long way (in debugging). But if there are really advanced bugs, it could be the entire system or the operating system that is a factor. This increases the complexity of troubleshooting, as you have more parameters that can cause issues, like losing a file every now and then." - Henrik	6/6
Challenges with Linux	Troubleshooting difficulties, potential compatibility issues, and challenges managing dependencies.	"There are pros and cons. One disadvantage is that more things happen under the hood that you don't have full control over. If you have some custom stuff, the drivers become a bit more cumbersome, because in Linux, everything is built as a file, including drivers for much hardware, and it becomes a bit unintuitive when developing for hardware compared to a hard embedded system. In an embedded system, you might be able to create a driver that is more structured based on what it does and not how the operating system expects it to look." - Axel	5/6
Real-time requirements for Linux	Difficulties using Linux for strict real-time demands due to performance and resource management constraints.	"So, it's also about safety. We must not do anything that is dangerous. So, there is that aspect, and also the real-time aspect. I don't think we could currently switch to an entirely Linux-based node that does everything in a product, at least in our world, without having other nodes to ensure that we stay safe. That the product is not dangerous." - Henrik	5/6
Advantages of Linux	Access to extensive libraries, ready-made solutions, and ease of use for certain tasks.	"Yes, but it was very easy to get everything started, got started quite easily and, as I said, there is a lot of support available. You never have to reinvent the wheel in the same way. So, you can just download a library." - Erik	4/6
Open-Source	Open access to source code, less desirable for proprietary code, and potential discontinuation of projects.	"I think that to the greatest extent possible; we will write everything that is needed in a mower ourselves, and we will not want to share that with open source." - Gustav	4/6

Development Process	Bring-up challenges, increased Linux development process steps, and differences between Linux and other systems.	"Why do we use Linux? Well, I think that to a large extent it is about having dependencies on suppliers and manufacturers who have different libraries that they had available for Linux, which made it easiest to do the development in Linux as well, so you could just add their code." - Erik	4/6
Maintenance and Security	Maintenance and security vulnerability concerns for Linux systems compared to custom real-time OSs.	"Having this heavier operating system, more expensive processor, more expensive components, and a kind of maintenance as well. Because, I mean, you still need a team, probably to maintain a Linux distribution over time. There are many security vulnerabilities, especially if you have a connected product. You basically need to be able to update it if there's a very serious security breach around the kernel or something like heart bleed or what has been in the SSH/SSL libraries, for example. So, you need to be ready to deal with that, and it's a cost." - Henrik	3/6

Table 2. *Interview Respondents Experience Prior to the Interviews.*

Name	Expertise	Years of experience
Henrik	Software engineer	10+
Axel	Embedded systems	20+
Erik	Software engineer	10+
Gustav	Embedded systems	20+
Felix	C++ and game development	5+
Bertil	C++ and backend development	2+

Note. The expertise noted is the main field that the respondents have worked with in a professional setting and is in no way an exhaustive list of their skills or experience. Names are pseudonyms.

7.2 Hypothesis Generation

Here the hypotheses generated by the qualitative data collected during the semi-structured interviews will be presented. The transcripts were thematically analyzed into patterns and key themes where grounded theory was then employed to generate the following hypotheses.

1. Proper utilization of available libraries and open-source solutions for Linux leads to reduced development time

The most prominently mentioned aspect within the interviews was Linux's vast access to libraries and ready-made solutions. This suggests that development time can potentially be shortened by proper leveraging of these resources rather than creating all components from scratch.
The interview subjects highlighted the potential benefits of the open-source nature of Linux and its community driven development. Which is beneficial when it comes to maintenance and patching security flaws in software components. However, license agreements and ethics might come into consideration when companies want to preserve their codebase and keep it proprietary. Thus, making open-source aspects less desirable.

2. Linux struggles to meet strict real-time and security requirements in embedded systems development.

The concern for Linux ability to meet real-time requirements was expressed by most interview participants. Especially in situations where precise timing is crucial. In situations where strict real-time demands are in place Linux may not be able to meet those requirements due to its limitations in resource management and potential overhead aspects.
During the interviews, participants mentioned the security aspect of Linux. As a more general-purpose OS Linux cannot guarantee the exact execution of processes and is likely to suffer more security vulnerabilities as it is severely complex in comparison to that of any traditional or custom-made real-time operating system.

3. Linux presents challenges in terms of troubleshooting and managing dependencies.

Interview participants expressed the concern for raised difficulty in troubleshooting errors in Linux systems. The participants also mentioned the potential compatibility issues between software and hardware and maintaining and managing dependencies when using external libraries.

7.3 Systematic Literature Analysis of Hypotheses

This subchapter will investigate how the hypotheses derived from the semi-structured interviews hold up against pre-existing research in the field. The aim is to validate and challenge themes and trends from the qualitative analysis through a systematic literature analysis.

Each hypothesis will be reviewed individually and the search strategies can be found in the appendices: section 11.2.

7.3.1 Development Time and Access to Open-Source Software

The most asked questions regarding open-source software (OSS) licenses on Stack Exchange sites are not general about licenses, but users seem to have questions regarding a specific license according to a study by Papoutsoglou et al. (2022). The same study found that posts about how linking with OSS affected the licenses, got a lot of attention but seemed to be harder to answer as they were viewed a lot but had less comments.
In a survey conducted by Almeida et al. (2018), they try to get a better insight into developers understanding of different types of OSS licenses. They ask the respondents about three different licenses that differ in their restrictions. With the responses from their survey, they could observe that when only one license is involved, developers usually understand how the license works. But they also observed that developers did not have a good understanding of how licenses interact when there is more than one involved.

Lundell, Lings, & Syberfeldt (2011) conducted a study where they analysed the views of professionals in the embedded development industry who were familiar with OSS. They found that professionals found it easier to maintain long term maintenance with OSS instead of proprietary software. In embedded development the systems usually need to be maintained over a long period of time. This makes the risk, that a commercial vendor supplying a proprietary software leaving the market higher, simply because they have a longer window of time to leave. OSS provides at least some control over the software. They also learned that professionals expressed that it was easier to find consultants that could help and support with OSS than with proprietary software. They also learned that many professionals viewed OSS as low risk in licensing perspective, as they often had experienced problems with licensing when using proprietary software. However, they also learned that in the embedded domain, long term maintenance of OSS needed both an involved commercial player as well as contributing volunteers to be effective.

An analysis of Android smartphone manufacturers and their contribution to the development of the Android OS, showed that companies with the most contributions also had the shortest time to market after each release of the Android OS versions (Shiu & Yasumoto, 2016).

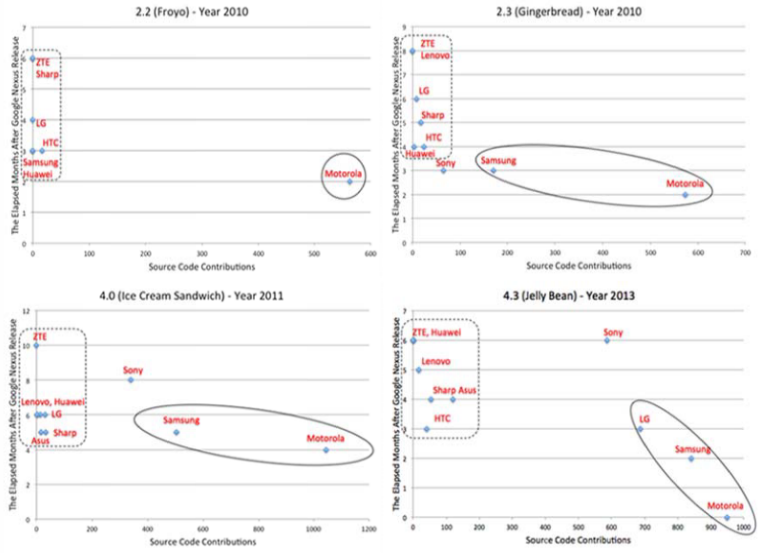

Figure 1: *Source code contribution related to product release.* Source: Shiu & Yasumoto (2016).

Similarly, Nagle (2014) found that all companies could increase their productivity by using more OSS, although it did also show that the benefit for companies in the service industry was larger than for companies in the production industry. Nagle (2014), unlike Shiu & Yasumoto (2016) was unable to confirm that a company's input to the development of the OSS would correlate with the output they would get from OSS.

7.3.2 Real-Time Requirements and Safety

In a journal article, Dodiu et al. (2010) examined the real-time capabilities of a Linux system. Their experiments measured processor load and found that the amount of jitter observed during a CPU task period varied depending on the load. The task period was 300µs, and their worst-case measurements were 22µs and 100µs of jitter for minimum and maximum CPU loads, respectively. This equates to 7.4% and 33% of the main task period. This result was considered unsatisfactory, as it could potentially delay critical tasks of other processes in the system.

Dodiu et al. (2010) concluded that while the Linux system might not be suitable for short duration, high-frequency tasks, for longer task periods the jitter that occurs in the CPU becomes more acceptable. Ultimately, depending on the application, Linux could be used as a hard real-time controller as long as the jitter does not prevent the functionality of the application.

In a series of research experiments conducted by Adam (2021) the response times of three real-time Linux systems were measured. These systems were using the PREEMPT_RT patch, a patch aimed to increase real-time performance of Linux systems. A majority of the latencies measured fell below 50µs and the maximum

latencies averaged across the systems was 160µs. Adam (2021) points out these latencies are significantly lower than that of the standard Linux kernel and provides a deterministic system as long as the upper bound latency is an acceptable safety margin. In another study, Adam et al. (2020) made use of the PREEMPT_RT patch to successfully use a COTS device with a Linux operating system as a real-time control system. The focus was to find efficient system solutions for real-time applications, specifically on COTS devices. One of the conclusions was the possibility of development time reduction as proclaimed in this quote: "Another outcome of this research is that such a COTS-based approach can reduce the development time and ease the implementation of low-cost experimental testbed systems for further research in real-time control based on COTS components" (Adam et al., 2020, p. 14)
Bruzzone et al. (2009) researched the feasibility of using Linux for controlling embedded real-time applications. The research experiments compared two versions of Linux using different threading properties while also investigating the impact of the ability to preemptively interrupt scheduled tasks. Table 2 below shows that with preemption activated Linux can for the vast majority operate within 50µs of requested periods. However, there are outlier cases where the requested periods may be close to 300µs delayed

Table 3. *Timing parameters between LinuxThreads/NPTL libraries and preemption option activated/deactivated* Source: Bruzzone et al. (2009)

Library	Preemption	T_{max} (µs)	$T_{99,99\%}$ (µs)	$T_{99,9999\%}$ (µs)
LinuxThreads	No	752	44	114
LinuxThreads	Yes	281	39	112
NPTL	No	495	39	114
NPTL	Yes	243	18	106

Note. T_{max} refers to the maximum discrepancy measured on the requested period of the task scheduler. $T_{99,99\%}$ represents the 99,99-percentile threshold and $T_{99,9999\%}$ the 99,9999-percentile threshold of the measurements.

7.3.3 Troubleshooting and Debugging in Linux

Based on the findings from Wei et al. (2021) it is evident that a significant portion of security bugs in open-source projects are related to memory operations, authentication requests, resource management, and security configuration. Notably almost half of all security bugs are categorized as memory operation bugs. Wei et al (2021) suggest that developers add more tests to identify such bugs while further considering the feasibility and rationale behind resource references while developing, in order to reduce the occurrence of such bugs.

In a journal Article from Spear et al. (2012) the authors highlight that the increased parallelism and complexity of multi-core systems presents daunting problems for

developers. In order to alleviate the problems, visualization of the system's behavior was desired. In the Linux community there existed several tracing technologies that used incompatible data formats and there was also a desire to share analysis tooling. Ericsson and the Embedded Linux Forum financed the endeavor to create the common trace format (CTF). CTF offers tracing capabilities across applications, architectures, and programming languages. Spear et al. (2012) believes that the availability of open-source components will enable developers to add tracing to their applications and grant an increased understanding in multi-core systems' behavior.

The advancements within tracing mechanisms have not only brought on improved system performance, but it also presents potential new benefits for the debugging process. In a study by Beamonte & Dagenais (2015) the authors presented efficiency improvements using the Linux Trace Toolkit: next generation (LTTng). Primarily targeting reduced latency and improved determinism. The study showed maximum latencies at 7 and 6 µs respectively for the standard Linux Kernel and the PREEMPT_RT patched kernel. Through the reduced latencies the tracing mechanism became increasingly stable with less variance in duration. According to Beamonte & Dagenais (2015) this allowed LTTng to support tracing even for very demanding real-time applications.

8 Discussion and Further Research

8.1 Comparison Between Interview Findings and Literature

This chapter presents a comprehensive comparison between the findings and insights obtained from the semi-structured interviews with existing peer-reviewed literature. From juxtaposing the findings with literature this chapter aims to reveal gaps in understanding of the topics as well as strengthen or question the validity of the findings. The aim is to explore novel perspectives that emerges from the interviews in order to provide valuable additions to the existing body of knowledge.

8.1.1 Libraries

In both the interviews and the systematic literature review it showed that the biggest concern for developers when using OSS is licensing (Almeida et al. 2018; Papoutsoglou et al. 2022). The literature review will provide more detail on how it becomes more complicated to understand how licenses interact when more than one is involved (Almeida et al. 2018). The literature review also revealed that many developers had a problem to understand how linking the OSS to proprietary code can affect how proprietary code will fall under the OSS license, this can also indicate that many developers are inclined to keep their own code proprietary.
The literature review suggested that in embedded there is a hesitancy to use proprietary software because of the possibility of running into a longevity issue (Lundell, Lings, & Syberfeldt, 2011). The interview participants spoke of third-party software as one of the large benefits of having a Linux system over a pure embedded. If a developer does not want to spend resources developing the software, OSS seems to be preferable to proprietary in many cases. Not just the cost aspect but that licensing issue does not necessarily get easier with proprietary software (Lundell, Lings, & Syberfeldt, 2011). For effective maintenance of OSS in the embedded domain it is best to have both a big commercial player and volunteers to contribute (Lundell, Lings, & Syberfeldt, 2011). Shiu & Yasumoto (2016) claims that companies that contribute more to development of OSS, in many cases will also have a shorter time to market after the software is released. They showed that more companies over time started to contribute more to the development of Android OS, the OSS their study were looking at. This was not something that appeared during the interviews, the possibility that the company could be involved in the development of the OSS.

Nagle (2014) claims that using OSS will increase a company's productivity, this aligns with the views of the interview subjects. As most of them came from the embedded world, a Linux system would give them access to more OSS and that would mean that they would need to develop less software themselves, and that would lead to less development costs and shorter time to market.

There seems to be a consensus that developers can shorten their time to market with OSS, but at the same time developers seem to want to keep software proprietary. There also seems to be a restraint to use OSS simply because developers do not understand the license it is under and are worried the software being developed will fall under OSS license. If companies were willing to make even a part of the company's software OSS, they could benefit from better maintenance and faster development of the software, the companies would not need to worry as much about licenses either. But that would mean losing proprietary rights to the software.

8.1.2 Real-Time Requirements

Findings from the systematic literature analysis and the interview findings support the hypothesis that Linux struggles to meet strict real-time and security requirements within embedded systems. Dodiu et al. (2010) observed jitter that could potentially delay critical task of other processes. While Linux may be more suitable for longer task periods, during short duration and high frequency tasks it was deemed unsatisfactory. Similarly, Adam (2021) measured response times that could provide a deterministic system as long as the upper latency bound was satisfactory. Bruzzone et al. (2009) Also echoes that within the vast majority of cases Linux could operate within requested periods, but there are outlying cases where the deterministic behaviors fail. Worth noting though is that the largest outlier was 243μs for the Native POSIX Threads Library (NPTL) which is the current standard for Linux systems today. While this is a notably large discrepancy from the 99,99 percentiles; 18μs, it may still be sufficiently small for many applications.

The interview participants justifiably echoed concerns regarding Linux's ability to meet strict real-time requirements. They expressed doubts regarding the precision of Linux while highlighting its limitations of resource-management and potential overhead aspects. The complexity of Linux's general-purpose oriented nature raised concerns for security vulnerabilities and participants believed that designated real-time or custom operating systems granted better guarantees for exact process execution and minimization of security vulnerabilities.

Overall, the comparison reveals a consensus that Linux faces challenges in meeting strict real-time requirements in embedded systems development. While patches such as the PREEMPT_RT patch show improvements, there are still occasional delays or timing precision errors. This coupled with the inherent complexity of its general-purpose nature prevents Linux from providing the same level of guarantees as a dedicated real-time operating system.

8.1.3 Troubleshooting and Debugging

Spear et al. (2012) emphasized the need for improved understanding of multi-core system behavior through tracing technology. The creation and open access to a protocol such as CTF. As discussed by Spear et al. (2012), enables developers to share analysis tooling and gain a better understanding of multi-core systems behavior. This resonates with the advantages of access to extensive libraries and ready-made solutions that was emphasized by the interview participants.
Additionally, Beamonte & Dagenais (2015) discuss the efficiency improvement that tracing technologies have achieved. The reduced latencies and improved determinism offered by the tracing tools enables powerful debugging tools even for powerful real-time systems. Addressing some concerns, the interviewees had related to the ability of troubleshooting and performance constraints of Linux.
Observing the findings from Wei et al. (2021), it is evident that security bugs in open-source projects often relate to memory operations and resource management. This aligns with concerns expressed by the interview subjects regarding challenges of troubleshooting and potential compatibility issues of Linux systems.

Through comparison of the findings from the interviews and the literature analysis it is apparent that developers face challenges in troubleshooting and debugging Linux systems. Although literature presents insights of potential solutions, such as tracing and enhanced deliberation behind the rationale of writing code, these are still challenges that developers are required to solve during development of Linux systems. The increased overhead generated by a general-purpose OS might be overbearing for safety critical systems. Increasing standards on security also puts pressure on the producer of systems to maintain distributed software, which incurs additional costs that could potentially be avoided or reduced by building a more specific, custom OS.

Overall, the findings highlight the ongoing need for developers to find effective solutions to troubleshooting and debugging on Linux systems, considering the complexities and constraints of the development environment.

8.2 Implications for Future Research

The preceding discussion has shed light on several key areas that warrant further investigation under the context of developing an embedded system using Linux. These implications provide potential directions for future research that would enable a deeper understanding of the challenges and opportunities that are associated with using open-source software in in embedded system development

8.2.1 Libraries and Open-Source Software

There has been a lot of research done on OSS over the years, a lot less seems to have been done on the use of OSS in embedded development. A big part of that might be the fact that embedded development has access to a lot less OSS, so it is not used as often. It could also be that embedded developers are more protective of their

proprietary rights. Some might argue that embedded development benefits greatly from OSS, especially when it comes to maintenance of software, since longevity of a product is important to embedded developers. Combine this with the fact that being involved in development of OSS can create a head start when the product is ready, and it seems it would be valuable to dive deeper into how companies can reach better productivity if they make some of their software OSS, especially in the embedded domain. This would also make it easier to handle licenses, since the product is already OSS there is no worry about it becoming OSS. There is great value in keeping software proprietary, but perhaps there is a worthy trade-off at some point.

There seems to be a lot of confusion among developers about OSS licenses. While there is already extensive research done on the subject, why the confusion remains and how it can be mitigated can also be an interesting subject to explore.

The use of Linux in embedded development would give access to a lot more OSS, exploring the difference in available OSS more in depth and quantify it could maybe help developers see the advantages or disadvantages of using Linux for embedded development more clearly.

8.2.2 Real-Time Requirements

The challenges of meeting strict-real time requirements with Linux controlling embedded systems have been highlighted in both the literature analysis and the interview findings. However, further research is required to deepen the understanding of this issue and explore potential solutions.

To further advance the current understanding of real-time capabilities in Linux systems, future research could focus on investigating the effectiveness of real-time patches, such as the PREEMPT_RT patch. This research path should aim to provide valuable insights that these patches provide on reducing delays and improving timing precisions for real-time scenarios.

Specifically, the research should consider factors such as task duration, frequency, and criticality to evaluate its real-time performance in different contexts. Researchers can gain a comprehensive understanding of the possibilities and constraints inherent to real-time Linux by investigating the limitations, trade-offs, and alternative approaches associated with these patches.

The exploration of alternative solutions, such as dedicated real-time operating systems or custom-made operating systems, would provide a more nuanced understanding of the limits of real-time Linux and highlight the potential benefits of alternative approaches. A comparative analysis with this focus would enable developers to make informed decisions when selecting a real-time solution for their embedded systems.

Overall, further research within this area would address the current lack of understanding regarding the limits and trade-offs that real-time Linux has. While also providing guidance for developers seeking to leverage the other benefits of a Linux System.

8.2.3 Troubleshooting and Debugging

The challenges that developers face while troubleshooting and debugging Linux systems have been established both from the interviews and the literature analysis. Further research is required to explore potential solutions to these challenges and create strategies to address them properly.

Building on the challenges identified through the interviews, future research should investigate which type of tools are missing in the Linux development environment, compared to that of traditional embedded system environments. By identifying these gaps, the Linux community can work towards developing necessary standards such as the common trace format (CTF), in order to effectively create tools that developers need and enhance the debugging capabilities for embedded Linux systems.

Additionally, research could focus on the development of robust techniques and tools to effectively mitigate security vulnerabilities, with specific focus on memory operations and resource management. Investigation of novel approaches and strategies for ensuring secure memory operations could be invaluable for enhancing the overall security and reliability of Linux Systems.

Overall, future research should highlight the importance of missing tools within the Linux development environment and develop robust techniques for mitigating vulnerabilities. By addressing these research paths, developers can be better equipped to face the challenges posed, while ultimately enhancing the overall performance, reliability, and security of these systems.

9 Conclusions

In conclusion, this study has aimed to explore the potential of Linux reducing the development time for embedded systems and investigate the factors which make it an appealing choice for an operating system. While the discussion chapter mainly highlighted gaps of understanding and proposed research questions, it has also shed light on certain aspects related to the research questions.

- How can Linux be used to reduce the time required for development in embedded systems

The findings suggest that Linux offers a vast array of libraries and ready-made solutions, which can significantly contribute to the reduction of development time. Adoption of such resources introduces additional considerations into the development process, such as the need for continuous maintenance and compatibility issues. However, the study failed to provide concrete evidence or quantitative data to support the claims of reduced development time. Instead, it emphasizes the potential benefits and need for further exploration.

- What factors make Linux an appealing choice for embedded system development, despite the inherent trade-off of reduced overall control?

The discussion identified several factors that could lead to potential development time reduction, including the availability of extensive libraries, open-source tools, and through the use of ready-made solutions. While the potential benefits of using a such as Linux in an embedded system sound quite positive, there are also trade-offs to consider. One notable trade-off is the increased overhead associated with a general-purpose OS compared to that of dedicated real-time operating systems. Others include strict real-time requirements, troubleshooting ability, and licensing concerns.

Overall, the study has identified key areas for further research for development of embedded Linux systems. The discussion has highlighted the complexities and challenges associated with the incorporation of Linux to the embedded domain. By delving deeper into these aspects, researchers can contribute to a more comprehensive understanding of the advantages, challenges, and trade-offs associated with utilizing Linux in embedded system development.

10 References

Adam, G. K. (2021). Real-Time Performance and Response Latency Measurements of Linux Kernels on Single-Board Computers. *Computers, 10*, 64. Retrieved from https://doi.org/10.3390/computers10050064

Adam, G. K., Petrellis, N., Kontaxis, P. A., & Stylianos, T. (2020). COTS-Based Real-Time System Development: An Effective Application in Pump Motor Control. *Computers, 9*, 97. Retrieved from doi:10.3390/computers9040097

Almeida, D. A., Murphy, G. C., Wilson, G., & Hoye, M. (2019). Investigating whether and how software developers understand open source software licensing. *Empirical software engineering: an international journal, Vol.24(1)*, 211-239.

Beamonte, R., & Dagenais, M. R. (2015). Linux Low-Latency Tracing for Multicore Hard Real-Time Systems. *Advances in Computer Engineering, 2015*. Retrieved from https://doi.org/10.1155/2015/261094

Braun, V., & Clarke, V. (2006). Using thematic analysis in Psychology. *Qualitative Research In Psychology*, 3: 77-101.

Bruzzone, G., Caccia, M., Ravera, G., & Bertone, A. (2009). Standard Linux for embedded real-time robotics and manufacturing control systems. *Robotics and Computer-Integrated Manufacturing , 25*, 178–190. Retrieved from https://doi.org/10.1016/j.rcim.2007.07.016

Charmaz, K. (2006). *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis (1st Ed).* London: SAGE Publications Ltd.

Charmaz, K. (2014). *Constructing Grounded Theory (2nd Ed).* London: SAGE Publications Ltd.

Dodiu, E., Graur, A., & Gaitan, V. G. (2010). Hard-Soft Real-Time Performance Evaluation of Linux RTAI Based. *ELEKTRONIKA IR ELEKTROTECHNIKA, 8*, 51-56.

Hesse-Biber, S. N. (2010). *Mixed Methods Research: Merging Theory with Practice*. New York: The Guilford Press.

Johannesson, P., & Perjons, E. (2014). *An Introduction to Design Science.* Springer International Publishing.

Josselson, R. (2013). *Interviewing for qualitative inquiry: A relational approach.* The Guildford Press.

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering.

Lundell, B., Lings, B., & Syberfeldt, A. (2011). Practitioner perceptions of Open Source software in the embedded systems area. *The Journal of systems and software Vol.84(9)*, 1540-1549.

Market Reports World. (2023, May 19). *Embedded Linux Market Analysis Report (2023-2030)*. Retrieved from The Expresswire: https://www.theexpresswire.com/pressrelease/Embedded-Linux-Market-Analysis-Report-2023-2030_21358945

Nagle, F. (2014). *Crowdsourced digital goods and firm productivity: Evidence from free and open soure software.* Harvard Business School.

Papoutsoglou, M., Kapitsaki, G. M., German, D., & Angelis, L. (2022). An analysis of open source software licensing questions in Stack Exchange sites. *The Journal of systems and software, Vol.183*, 111113.

Shiu, J., & Yasumoto, M. (2016). Benefitting from Contributions to the Android Open Source Community. *Annals of Business Administrative Science, Vol.15(5)*, 239-250.

Spear, A. , Levy, M., & Desnoyers, M. (2012). Using Tracing to Solve the multicore System Debug Problem. *Computer, 45*(12), 60-64. Retrieved from https://doi.org/10.1109/MC.2012.191

Tarozzi, M. (2020). *What is Grounded Theory?* Boomsbury Publishing.

VDC Reasearch. (2018, January 25). *IoT and Embedded Operating Systems Market to Reach 11.1B Units by 2021 According to VDC Research*. Retrieved from www.VDCResearch.com: https://www.vdcresearch.com/images/pr/2018/jan/IoT-and-Embedded-OS-01-25-18.html

Wei, Y., Sun, X., Bo, L., Cao, S., Xia, X., & Li, B. (2021). A comprehensive study on security bug characteristics. *Software: Evolution and Process, 33*(10). Retrieved from https://doi.org/10.1002/smr.2376

11 Appendices

11.1 Interview Guideline

Appendix A.
Interview Guideline

Interview Guideline

Introduction

- Introduce yourself and explain the purpose of the interview.
    - An investigation into how Linux can impact the development time of new embedded projects.
- Request permission to record the interview.
- Ask the interviewee to briefly introduce themselves and their background in embedded and Linux development.

Experience and Background

- What experience do you have in developing embedded systems and Linux-based applications?
- Have you worked on any embedded system projects that used Linux? If so, can you describe:
    - Why did the project use Linux?
    - Was Linux beneficial or a disadvantage for the project?
    - Did you encounter any challenges specifically because of the choice of operating system?

Linux in Embedded Development

- What opportunities does Linux provide for shortening Time to Market?
    - What disadvantages make Linux development longer to market?
- Are there features or tools exclusive to Linux that you find helpful for software development?
    - Are there any tools or features missing?
- Is the use of external libraries different when developing on Linux?
    - How do third-party licenses affect development?
    - How does access to open-source affect development?
    - Is access to external libraries an important part of development on Linux?
- How do you see the use of Linux affecting the development time of embedded system projects?
    - Does the availability of high-level languages make a difference in development?
    - Do high-level languages provide easier access for developers?
        - Do all developers need to have knowledge of Linux?

- How does the use of Linux affect the costs and resources required for application development?
    - Price, time, skills?
    - Do you see systems developed on Linux having access to more resources than those developed with other solutions?
        - If yes, why?
        - How does this affect the overall product cost?
- What types of embedded projects would benefit most from using Linux?

Development Process

- Do you believe there would be any significant differences in the development process depending on the OS used?
- How do you choose an operating system? Are there criteria or other factors you consider?
- Does Linux provide equal opportunities for troubleshooting in embedded systems?
- How does Linux affect the opportunities for continuous integration?
    - In embedded systems?
    - In software applications?

Technical Knowledge

- How do you feel about making changes to the Linux kernel?
    - Is it reasonable to maintain changes?
- Would Linux still provide the low-level control required for embedded development?
    - If not, what is missing?
    - If yes, why is it not used more often?

Future Directions

- How does the use of open-source and third-party libraries affect maintenance and product lifespan?
- Where do you think the future of embedded development with Linux is heading?
- What new technologies or advancements are you excited about in this area?

Conclusion

- Ask if there is anything else they would like to add or discuss.
- Are there any questions that they think were difficult to answer or unnecessary?
- Ask if the interviewee can recommend someone else for an interview.
- Thank the interviewee for their time and insights.
- End the interview.

11.2 Systematic Literature Review Search Strategies

Appendix B
Search Strategy for Libraries and Open-Source Software

Platform: Primo
Filter: Peer-Reviewed, English
Search Terms: (“open source” OR “open-source” OR “third party”) AND (“librar*”) AND (“development time” OR “development-time”) AND (“licenc*”)

Inclusion Criteria:
Studies that focused on open source in regard to development or developers in general
Studies that provide empirical evidence or theoretical analysis within the topics.
Study is Peer reviewed and written in English.
Study is published after 2007.

Exclusion Criteria:
Studies where the focus is a specific open-source software artifact
Studies that do not discuss any relevant aspects of how open source can affect the development of software
Studies that lack any empirical evidence or theoretical analysis related to the topic.
Studies that are not peer reviewed or not in English.
Study is published prior to 2007.

Appendix C
Strategy for Linux Real-Time Performance

Platform: Primo
Filter: Peer-Reviewed, English
Search Terms: ("Linux") AND ("embedded system" OR "embedded development") AND ("performance" OR "optimization" OR "real-time")

Inclusion Criteria:
Studies focused on Linux operating system in the context of embedded systems.
Studies that discuss performance, real-time application or optimization of the operating system.
Studies that provide empirical evidence or theoretical analysis on the use of Linux in embedded systems.
Peer reviewed studies in English.
Study is published after 2007.

Exclusion Criteria:

Studies where the focus is not on Linux in the context of embedded systems development.
Studies that do not discuss any relevant aspects to the topic performance of Linux in embedded systems development.
Studies that lack any empirical evidence or theoretical analysis related to the use of Linux in embedded systems development.
Studies that are not peer reviewed or not in English.
Study is published from before 2007.

Appendix D
Search Strategy for Linux Troubleshooting and Maintenance

Platform: Primo
Filter: Peer-Reviewed, English
("Linux") AND ((("dependenc*" OR "mainten*") AND ("open-source" OR "third party" OR "library")) OR ("troubleshoot*" OR "debug*" OR "error*"))

Inclusion Criteria:
Studies focused on Linux operating system.
Studies that discuss troubleshooting or debugging procedures or studies that discuss maintenance or management of dependencies.
Studies that provide empirical evidence or theoretical analysis within the topics.
Study is Peer reviewed and written in English.
Study is published after 2007.

Exclusion Criteria:
Studies where the focus is not on Linux.
Studies that do not discuss any relevant aspects to the topic troubleshooting or maintenance of a Linux system.
Studies that lack any empirical evidence or theoretical analysis related to the topic.
Studies that are not peer reviewed or not in English.
Study is published prior to 2007.