On the Use of Package Managers by the C++ Open-Source Community

André Miranda
Department of Computer Engineering
University of Pernambuco
Recife, Brazil
aldm@ecomp.poli.br

João Pimentel
Universidade Federal Rural de Pernambuco
Cabo de Santo Agostinho, Brazil
joao.hcpimentel@ufrpe.br

ABSTRACT

THIS IS A PRE-PRINT COPY

The use of package managers is commonplace for software developers working with programming languages such as Ruby, Python, and JavaScript. This is not the case for C++ developers, which present a low adoption rate of package managers.

The goal of this study is to understand what is preventing C++ developers from adopting package managers in the context of open-source software (OSS) projects. In order to achieve this goal, we performed a questionnaire survey with 343 developers from 42 OSS projects. The survey participants answered a questionnaire with 29 questions.

After the analysis of the collected data, we could conclude that most participants are not reluctant to use C++ package managers and that Open-Source licensing, High Availability of Libraries, Good Documentation, and Ease of Configuration can be considered crucial factors for the successful adoption of C++ dependency management via language-specific package managers.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**; *Software configuration management and version control systems*; • **Information systems** → *Open source software*;

KEYWORDS

Open-source software; Software maintenance; Software tools; Empirical study; Software engineering

ACM Reference Format:
André Miranda and João Pimentel. 2018. On the Use of Package Managers by the C++ Open-Source Community. In *Proceedings of ACM SAC Conference (SAC'18).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3167132.3167290

1 INTRODUCTION

Since the introduction of CTAN and CPAN as early language-specific package managers [10], developers of many programming languages have been taking advantage of this type of tool in order to facilitate the handling of dependencies.

Package managers promote reuse by providing easy access to software packages, which may range from simple utility routines to full-blown frameworks. One of the most popular package managers, NPM, hosts more than 350.000 packages on its official registry, serving more than 2.5 billion package downloads per week as of July 2017[1].

[1]https://www.npmjs.com

In some cases, the package manager is introduced in tandem with a language, which leads to design decisions of the language being made with the specific purpose of supporting the management of code units, generally known as packages [17].

Unlike other languages such as Go and Rust, C++ appeared many years before the first language-specific package managers, which helps explain the lack of a standard package format, the lack of precise semantics for modules, and the lack of support for the large amount of existing tooling and legacy code that was developed prior to the initial package management proposals.

The potential benefits of using third-party libraries are already known: lower costs, reduced time to market, and higher quality [13][7]. However, the lack of support for C++ modules and the heterogeneous landscape of the C++ environment[2] featuring: multiple compilers for different architectures, a handful of build systems, and dependencies on operating systems make the development and use of C++ package managers challenging. This complexity is also noted by Neitsch et al. claiming that "C/C++ software has complicated configuration and construction phases because of the need to support enormous platform variation." [11].

Additionally, Dos Reis and Hall [4] argue that the absence of direct language support for componentization of C++ libraries and programs has led to serious impediments regarding compile-time scalability and programmer productivity.

These characteristics have brought to our attention the case of open-source C++ projects. In spite of being written in one of the most popular programming languages [1], dependency management in these projects is handled using several non-standard methods, oftentimes resorting to system package managers for each supported operational system. This is the case even though there are at least nine language-specific package managers available for C++. Hence, the goal of this study is to **identify what are the factors that prevent the adoption of language-specific package management tools in open-source C++ projects**, from the point of view of software developers.

In order to obtain information with the open-source community, a 29-questions questionnaire was designed and applied as an online survey, centered on the following research questions:

RQ1 Do developers use the existing C++ package managers?
RQ2 What obstacles prevent the use of such tools?

[2]The C++ development environments are heterogeneous and this has been discussed by developers throughout the web, some of them argue that the availability of several tools, none of them being a de facto standard, as well as the large legacy code base, make package management for C++ much harder than most languages. Here is a discussion from a prominent Q&A software engineering forum: http://softwareengineering.stackexchange.com/q/170679

RQ3 Currently what are the preferred methods to handle dependencies in C++ projects?
RQ4 What is expected of a C++ package manager?
RQ5 What is the community's opinion towards C++ package managers?

RQ1 examines how much this development community is aware of the available tools. RQ2 aims to identify the most relevant factors that play a negative role in the low adoption of C++ package managers. RQ3 investigates the current methods employed in C++ projects to handle external libraries. RQ4 summarizes the most wanted features of a C++ package manager. Finally, RQ5 surveys the opinion of the developers hoping to learn if they are favorable or against this kind of package managers and how has been their experience while using them.

In the next section, an overview of related work is presented. The methodology of this study is described in Section 3. The results are presented in Section 4 and discussed in Section 5. Threats to its validity are presented in Section 6. Lastly, Section 7 presents conclusions and future work.

2 RELATED STUDIES

Muhammad et al. [10] have briefly surveyed the history of package managers to compare and explain some design decisions for LuaRocks, the standard package manager for Lua programming language. Whereas that paper provides an overview on previously developed package managers, here the focus is on the current opinion of potential users of such tools.

A modular architecture for package managers is proposed in [2], which might prove useful for a package manager dealing with the complexities required by C++ projects. Whereas that paper may help with the technical challenges of developing package managers, it does not contribute to the understanding of adoption factors.

We were able to find numerous surveys about open-source software development, primarily focused on social aspects and motivation [8] [18]. Whereas not directly related to package managers, these papers provide insights on the motivation of open-source software developers and can assist the developers of package managers to understand their potential users further. Additionally, Sojer and Henkel conducted an extensive survey[16] in order to understand how open-source code is reused. There, the focus is on *why* code is reused, whereas here we analyze *how* code is reused.

The study by Krafft et al. [9] identified a set of 15 factors that influence tools adoption by open-source developers: sedimentation, marketing, 'peercolation', first impressions, elegance, resistance, sustainability, quality documentation and examples, trialability and scalability, compatibility and genericity, modularity and transparency, maturity, network effects, consensus, and standards and uniformity. Albeit not directly related to language-specific package managers, these factors are likely to be relevant in this context as well.

The work by Wu et al. [20] reports on an empirical study that analyzed the usage of packages from the C++ standard library on open source projects. Since that work is focused on the standard library, it is not concerned with the mechanisms used to manage these dependencies.

Lastly, some studies (for instance, [3] and [6]) describe the architecture of operating systems package managers. While their discussion of technical challenges can be helpful for package managers developers, they do not address the topic of software adoption.

3 METHODOLOGY

In this study, we intend to understand how dependencies are currently handled in C++ projects and what is preventing C++ developers from adopting C++ package managers.

This research was organized in four phases, as follows. In the first phase, we mapped out the existing C++ package managers tools, so that we know which tools are currently available for C++ developers. In Phase 2 we identified candidate open-source C++ projects to be studied. We also analyzed how those projects handle their dependencies and identified their contributors. In Phase 3 we designed and ran a survey pilot, collecting data and feedback to improve the questionnaire. In Phase 4 the final survey was executed, and data was gathered. Finally, in Phase 5 we processed and analyzed the data gathered during the survey. These phases are further described in the following subsections.

3.1 Existing C++ Package Managers

Before starting the survey proper, we searched for language-specific package managers meant for C++ projects on GitHub, SourceForge, projects forums and on programming-related social networks, as well as through general web search engines. The following projects were identified: Biicode, Build2, Conan, CPM, Hunter, Mason, Meson, Pacm, and Yotta.

All of these projects, except Biicode and CPM, were in active development during the year previous to the survey (July 15, 2015, to July 15, 2016). Nonetheless, these two tools were also included in the survey questionnaire, as we would like to hear from respondents their experience with C++ package management tools in general.

3.2 Projects Selection and Current Practices

Nowadays, there are a diverse range of open-source software (OSS) projects, from embedded systems to large, distributed systems. Many of these projects are developed and managed openly on online repositories, such as GitHub and SourceForge. In order to have a sample of projects that is representative of this diversity, the projects selection was stratified into the following categories: Audio & Video, Database, Development Tools, Games, Graphics, Home & Education, Internet & Communications, Library, Science & Engineering, and Utility.

In order to filter projects for participation in the study, the following criteria were adopted:

(1) Open-Source License
(2) Mostly written in C++
(3) Active development during the year previous to the survey (July 15, 2015, to July 15, 2016)
(4) Make use of at least 3 external libraries

The criterion of using at least three external libraries was adopted since projects with little or no dependencies would most likely not benefit from package management tools, and thus they are not the target of our study.

The search was conducted on GitHub and SourceForge, two of the largest public repositories of source code, resulting in 45 projects matching the criteria[3]. The list of selected projects, along with the number of contributors active during the year previous to the survey, is presented in Table 1.

Through an extensive analysis of these projects' source code and documentation, we observed that none of the projects were using a C++ package manager. Instead, they used eight different methods for dependency management:

(1) Header-only libraries
(2) Source code amalgamation
(3) Library sources in the repository
(4) Git submodules
(5) System package manager
(6) Manually compile and install libraries
(7) Custom script to fetch and install libraries
(8) Precompiled packages

Some projects adopt multiple dependency management methods. This may be due to, among other reasons, differences across the supported platforms. For instance, Microsoft Windows does not contain an official package manager, unlike most Linux distributions. Another possible reason is a difference of preferences between contributors.

3.3 Survey Design & Pilot Questionnaire

Despite the relatively high number of tools for C++ package management, we have observed in our initial research that these tools are seldom adopted. Thus, we set out to understand what factors influence the adoption of such tools. To answer the research questions of this study, we performed a questionnaire survey with developers of C++ open-source projects. Besides basing the survey design on other existing surveys, we followed guidelines from The Survey Kit [5] as well as from Pfleeger and Kitchenham's series on principles of survey research (which started on [12]).

The questionnaire was crafted on LimeSurvey [14], an open-source survey tool. An early version of this questionnaire was discussed with authors of C++ package management tools. The questionnaire was further refined through multiple reviews by C++ development experts, as well as by experts on empirical studies.

Once the questionnaire was validated by experts, a pilot study was executed with approximately 10% of the selected developers' sample, kept available online for ten days. During this phase, 207 developers were contacted yielding 53 responses (25.60%), with 23 incomplete (11.10%) and 30 full responses (14.50%). After reviewing the feedback and analyzing the generated data, we have performed adjustments before reaching the final version of the questionnaire, which is available in a static form at GitHub[4].

The final version of the questionnaire is composed of 29 questions grouped into the following categories:

Characterization (Questions 1-4): We ask about the respondents' age, if their main job requires programming skills and how many years of experience with C++ they have, as well as with other languages.

[3]Originally, the Chromium project was also included in this study, but it was later excluded since our contact attempts were blocked by spam filters.
[4]https://github.com/andreldm/cpp-survey

Table 1: Projects Summary

Project	Contributors / Respondents	Category
Audacity	22 / 4	Audio & Video
Clementine	36 / 8	
Hydrogen	11 / 1	
Kodi	115 / 14	
Mixxx	38 / 10	
Tomahawk	13 / 1	
EventQL	4 / 2	Database
MongoDB	89 / 9	
RethinkDB	54 / 8	
Scylla	21 / 4	
LiveCode	27 / 2	Development
Mapnik	19 / 4	
Redis Desktop Manager	8 / 2	
Torque2D	8 / 2	
TortoiseGit	10 / 3	
Minetest	150 / 16	Games
Simultrans	3 / 1	
StepMania	22 / 4	
SuperTuxKart	24 / 3	
Warzone2100	7 / 0	
Aseprite	12 / 4	Graphics
FreeCAD	50 / 7	
Natron	6 / 2	
OpenSCAD	18 / 6	
SolveSpace	6 / 1	
Synfig	5 / 1	
GoldenDict	3 / 1	Home & Education
Kiwix	12 / 2	
LibreOffice	265 / 39	
Lyx	19 / 3	
Firefox	834 / 73	Internet & Communications
Miranda NG	17 / 5	
PSI	5 / 2	
CEGUI	9 / 2	Library
NanoGUI	12 / 3	
OpenCV	176 / 47	
OpenImageIO	19 / 7	
Bosen	6 / 2	Science & Engineering
Ceph	226 / 21	
DeepDetect	2 / 0	
Tesseract	27 / 7	
CuteMarkEd	5 / 1	Utility
GParted	40 / 6	
lnav	6 / 0	
Newsbeuter	16 / 4	
Total 2477/343		

Project Participation (Questions 5-8): In this section, we ask the participants which project they contribute to, whether it is related to their job or it is a hobby, their role in the project, and when was their first contribution.
Project Setup (Questions 9-14): We survey how easy it was to set up the project, the operating systems used, what types of package managers were used, and how satisfied the participants are with the way dependencies are handled in the project. There are also Likert scale questions about building and dependencies issues.
Experience with Package Managers (Questions 15-22): These questions were designed to examine the participants' experience with package managers of any language and type, their stance towards language-specific package managers and the most important positive and negative features that might influence their usage of package managers.
C++ Dependency Management (Questions 23-29): Finally, in this section we ask the participants' preferred way to add dependencies to a C++ project, their preferred build systems, if they had already used a C++ package manager, how was their experience with them, a Likert scale question regarding the eventual usage of a C++ package manager, and an open-ended question.

3.4 Survey Execution

The questionnaire was available online for 30 days. Invitations to partake in the survey were sent by email. Aiming to improve participation, we offered US$ 10 vouchers to be randomly distributed to three respondents. Two weeks after the questionnaire launch, we have sent reminders to participants — this proved to slightly increase the response rate in the final days. A total of 2447 contributors were contacted, as per Table 1.

Three projects from the selection of 45 projects have yielded no responses from any participant; they are: DeepDetect, lnav, and Warzone2100.

3.5 Data Analysis

Once the questionnaire has expired, the data was exported from LimeSurvey in CSV (comma separated values) format and then curated into a SQL database, allowing greater flexibility in analysis. In the end, there were 402 responses (15.72%), where 59 were incomplete (2.31%), and 343 were full responses (13.41%). The incomplete responses were discarded. The results of this analysis are presented in the next section and discussed in Section 5.

4 RESULTS AND FINDINGS

In this section, we present the results of our survey, grouped by research question. These findings are discussed in Section 5.

4.1 RQ1: Do developers use the existing C++ package managers?

Based on the data collected during the survey, we could grasp the usage frequency of C++ package managers. We also gathered data about usage of language-specific package managers for other languages that the developers also develop with, for comparison sake. Both questions were asked considering a 3 months period. A summary of these results is presented in Table 2 — the answers "I don't know what is a package manager" and "Never" are grouped into the "Do not use" category, and the remaining (from "Rarely" to "Always") are grouped into "Use". The four participants who claim to use only C++ were not included in the row for other languages in Table 2.

Table 2: Usage of package managers during the last 3 months prior to the survey execution

Language	Do not use	Use
C++	275 (80.17%)	68 (19.83%)
Other languages	46 (13.57%)	293 (86.43%)

We can see in Figure 1 the detailed landscape of package managers usage for other languages. The frequency is well distributed, but we can observe that most participants made some use of packagers managers.

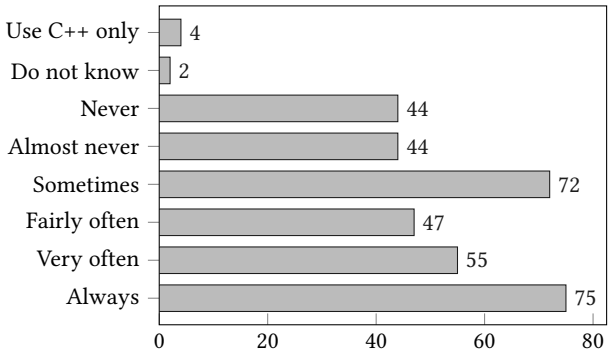

Figure 1: Frequency distribution of package managers usage for languages other than C++, during the last 3 months prior to the survey execution.

On the other hand, Figure 2 makes it clear that most participants never used a C++ package manager during the same period. With this data, we can state:

(1) 80% of participants have not used a C++ package manager in the stated timeframe, or do not know what is a package manager.
(2) About 15% of participants rarely used a C++ package manager in the stated timeframe.
(3) Only 5% of participants use a C++ package managers at least occasionally in the stated timeframe.

4.2 RQ2: What obstacles prevent the use of such tools?

To answer this research question, we have asked in the survey what are the five most important features that could negatively influence the participants while using a package manager. The answers can be seen in Figure 3.

The biggest concern of the participants regarding these tools is about their license, as Closed Software was the negative feature most selected. Probably this is due to the fact that the participants

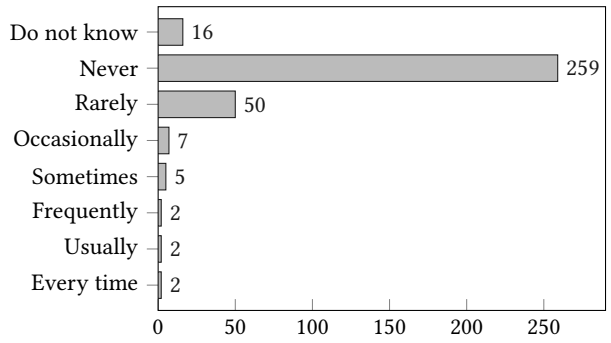

Figure 2: Frequency distribution of package managers usage for C++, during the last 3 months prior to the survey execution.

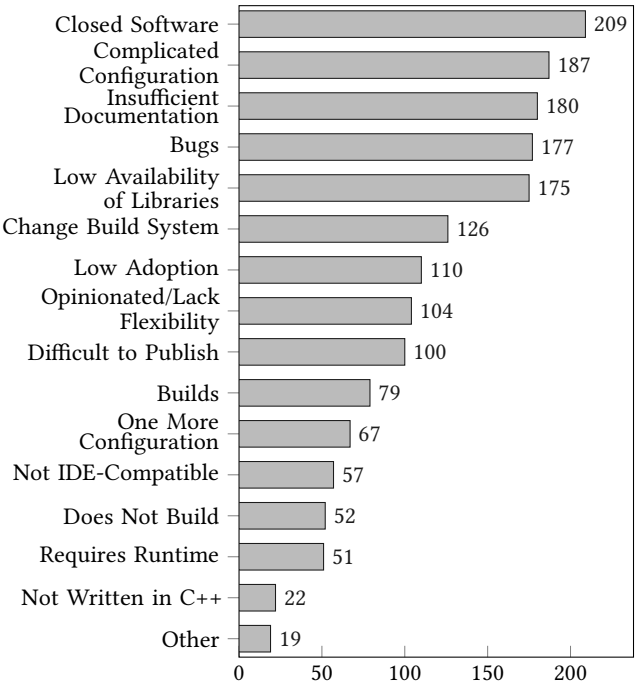

Figure 3: Negative features that may hurt the adoption of package managers.

are collaborators of open-source projects – for this reason a proprietary solution would not befit their purposes or would be against their philosophy. However, this is not a problem for any of the existing tools we identified as all of them are open-source projects.

The participants were also concerned with Complicated Configuration, as this was the second most selected negative feature.

Insufficient Documentation was the third most picked negative feature. This characteristic may be impacting the adoption of existing C++ package managers, since extensive documentation for them is not available. Six of the identified tools provide manuals in wiki-like format, while the others only offer brief instructions on readme files. Their presence in question & answer sites and programming forums is far smaller than what can be observed regarding package managers for other languages. Also, to the best of our knowledge, there is no published book or academic work regarding C++ package management tools.

Moreover, the occurrence of Bugs is relevant to participants. Bugs can affect the user experience and may range from minor glitches to blockers such as failure to handle dependency conflicts, compiling and linking breakages, etc. Every C++ package manager considered in this study is an open-source project, so bug reporting and push requests mechanisms are in place and provide a way for dealing with defects; nonetheless, some tools are currently unmaintained, making bugs an even worse hindrance.

Regarding repositories, almost all tools maintain some sort of official repository for packages. Nevertheless, they all allow the use of private repositories or to fetch dependencies from user-defined URLs. We perceive this feature as essential considering the strong presence C++ maintains on corporate projects. Nonetheless, hindrances to make libraries available and to maintain their versions might undermine tools adoption in any type environment, be it open-source projects where contributors usually rely on official repositories or corporate projects that need to justify the time and cost to maintain on-premises repositories. The Low Availability of Features was the 5th most selected negative feature.

Considering the least selected negative features, it seems that participants are not at large concerned in which language the tool must be written or if it requires a runtime environment (such as it is the case for Java or Python).

A bigger factor damaging the adoption of C++ package managers, however, may be the lack of awareness — when asking about the usage of and satisfaction with C++ package managers, we observed that these tools are unknown by 89.05% of participants.

4.3 RQ3: Currently what are the preferred methods to handle dependencies in C++ projects?

In the survey, we asked the participants about their preferred method to add dependencies to C++ projects; their answers are depicted in Figure 4.

By far the most selected method to add dependencies is through system package managers. This can be explained by the fact that system package managers are widely adopted by Linux/BSD users and macOS users, which correspond to 58.13% and 17.89% of respondents, respectively.

While system package managers are available for numerous Linux distributions and other Unixes, this is not the case for Windows, which is used by 21.23% of respondents. This helps to understand why the use of Header-only libraries was the second most selected option, as this method is usually not affected by the availability or the lack thereof of a system-level package manager.

Perhaps because many open-source projects are nowadays using Git as their Version Control System, some participants selected the Git Submodules option. This method liberates developers from system-level package managers and has the advantage of keeping the repository as lean as possible, containing only project files and no library sources in the repository. Unfortunately, this facility

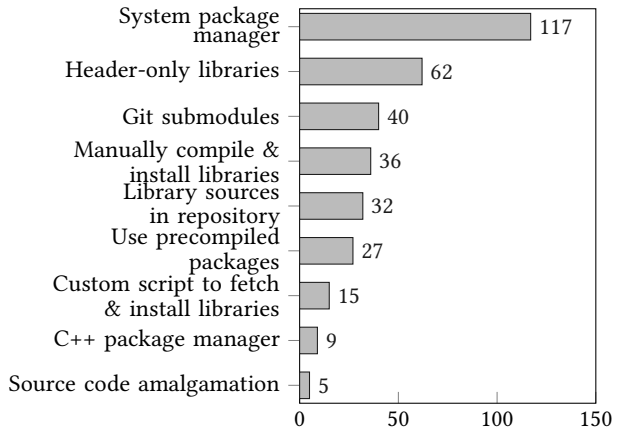

Figure 4: Preferred methods to add dependencies to C++ projects.

comes at the price of integrating the building of those Submodules to the project's own build configuration.

4.4 RQ4: What is expected of a C++ package manager?

Similar to RQ2, we have asked what are the five most important features that could positively influence the participants while using a package manager, aiming to find out what features the participants expect from package managers. The answers are summarized in Figure 5.

Just like Closed Software was the major concern in RQ2, participants' most expected aspect of a package manager is Open-Source License. Again, this is most likely because of the respondents' collaboration with open-source projects.

Very close to the former feature, Good Availability of Libraries is also highly expected from package managers. Even though most of the tools allow running private repositories or adding dependencies from arbitrary locations, participants desire hassle-free use of this kind of tool — in other words, they do not want to spend time packaging each dependency that is not available in the official repository. Currently, this is a problem for tools like Meson and Build2, since at the time of writing their official repositories offer no more than 20 packages. This is a bigger issue yet for tools such as Pacm, which has no official repository.

A bit distant from the previously discussed features, we can find the capability of Handling Dependency Conflicts. Such conflicts arise, for instance, when different projects require different versions of the same library. Specifically for C++ projects, complexities related to the build and dynamic linking processes may also play a role in the Handling of Dependency Conflicts because both processes are affected by system-wide libraries and their versions may vary from system to system.

As also discussed in RQ2, the participants value Good Documentation for package managers, confirming that this feature is crucial for a successful package manager. Even though the term Good Documentation is too broad, we perceive it as a collection of manuals and guides, books, tutorials, sensible example projects and

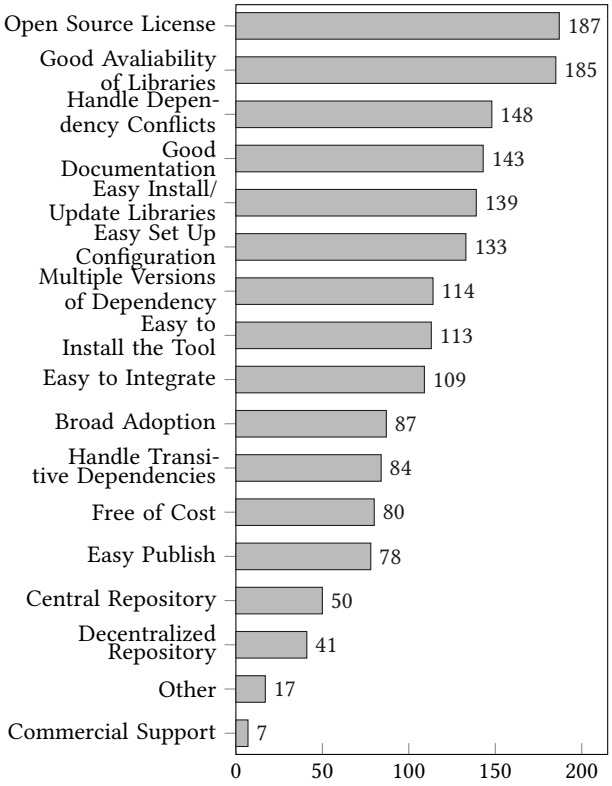

Figure 5: Positive features expected in package managers.

presence on forums and question & answers sites. Of course, this cannot be expected from projects carried out by few individuals, mostly volunteers — perhaps a strong community or a backing company is required in order to address this concern.

Regarding the bottom section of this features list (Figure 5), Commercial support was by far the least selected feature — perhaps because of the open-source background of the participants. It is also interesting to notice that both types of repositories (centralized or decentralized) were similarly neglected.

Lastly, the survey questionnaire also included a question regarding the use of Integrated Development Environments (IDEs). We can conclude from the results presented in Figure 6 that participants are unlikely to give up their preferred IDE, so we understand that IDE support is an important feature for C++ package managers to attract users.

4.5 RQ5: What is the community's opinion towards C++ package managers?

One of the survey's questions asked the participants about their stance regarding language-specific package managers. Their answers are represented in Figure 7.

Besides this nominal closed question, we asked the participants to explain their stance. The following quotes account well each positioning (Pro and Against):

"C++ is a great language, but think about the life of a C++ developer compared to a developer of another language: If he is a new

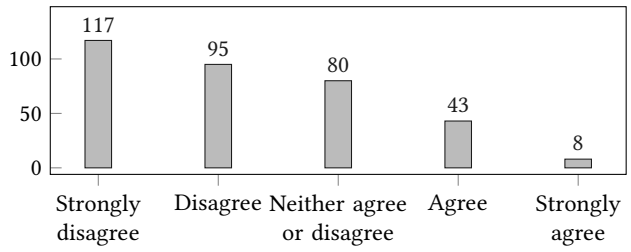

Figure 6: How willing are the participants to give up their preferred IDE.

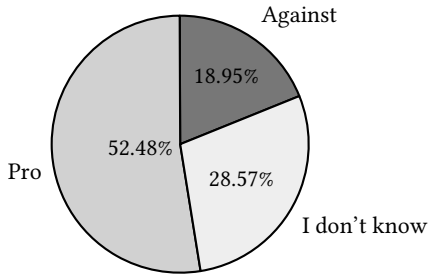

Figure 7: Stance regarding language-specific package managers.

user, the barrier of entry is massive. Not only must he learn a new language, but also the intricacies of linking and compiling dependent libraries into his project. Because of the former, even for an experienced developer, the time to set up a new project is greatly exasperated by having to write a ton of build files and link dependencies manually. Some libraries don't list their dependencies or don't have a How-to on importing it to your project. Then you are stuck with trying to reverse-engineer their dependencies and how to use it in your project. There are way better things to spend your time on than that! (...) This is why I support language-specific package managers." – Participant #230, Pro.

"Language-specific package managers may be useful on Windows, but on Linux, I prefer to have software packaged on the system level. On Fedora, most of the libraries are provided as RPMs from official repositories. Typically, if I need anything outside of that, it is due to different version requirements, and I end up setting up a separate build environment anyway." – Participant #223, Against.

We have also asked whether the participants would use C++ package managers if they were as mature as the ones for other languages. Figure 8 tells us that 51.31% of participants would use a C++ package manager, not surprising since about 52.48% of them are pro language-specific package managers. This reveals that C++ package managers have a significant user base waiting for a tool that can elegantly manage dependencies for C++ projects.

Lastly, we have asked participants if they would like to split the source modules into packages and assemble them via a C++ package manager. This is an interesting approach observed in projects written in other languages, e.g., Java and JavaScript. This approach takes advantage of package management not only to incorporate

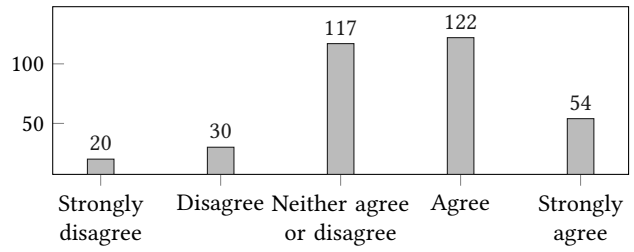

Figure 8: How willing are the participants to use a C++ package manager if they were as mature as are the ones for other languages.

third-party code but also to affect the project's design, promoting modularization. The answers are summarized in Figure 9. We can deduce by a slight margin that most participants would like to use this approach, but the portion that is not willing cannot be disregarded. Perhaps people might be persuaded if this pattern becomes commonplace as happened for other languages.

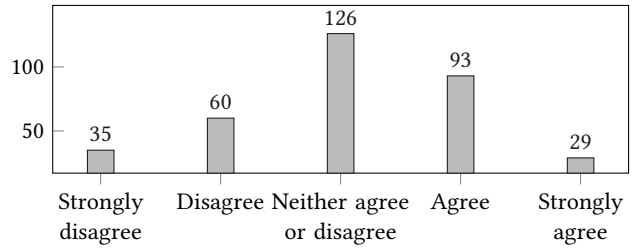

Figure 9: How willing are the participants to split source code into modules and integrate them via a C++ package manager

5 ANALYSIS AND DISCUSSION

This section presents some considerations on the survey results. A discussion of validity threats is presented in Section 6.

Based on the results for RQ1, we could observe a great discrepancy between the use of package managers (19.83% of the respondents) in C++ and their use in other languages (86.43%). This is even more evident when analyzing the frequency of use, since 73.91% of the former use C++ package managers rarely.

Contrary to our expectations, the language on which the C++ package management tool is developed was not considered a relevant factor for its adoption (only 22 votes for it being a negative factor, out of 1715). The most voted negative features are: low availability of libraries (175 votes), bugs (177 votes), insufficient documentation (180 votes), complicated configuration (187), and closed software (209 votes).

Analyzing the characteristics that could be seen as positive, the least voted one (with seven votes out of 1715) was commercial support. This lack of interest on commercial support is also corroborated by the anecdotal evidence of Biicode, which was supported by a company but then became unmaintained as the company went

out of business[5] after failing to sell services and thus to get enough paying customers. Nevertheless, this result may be biased since our survey was performed with developers of open-source projects only.

Whether the package repository is centralized or decentralized does not seem to be a relevant factor, with a similar amount of votes for each option (41 respondents considered decentralized repository a positive feature, whereas 50 respondents considered a central repository a positive feature). The broad adoption of the tool was also not considered particularly relevant, with only 87 votes. Thus, even though the popularity of a package manager tool may increase the number of people that are aware of the tool, it does not seem to be a relevant factor in making them actually adopt the tool.

The characteristics that are most considered positive are: ease to install and update libraries (139 votes), good documentation (143 votes), handling of dependency conflicts (148), good availability of libraries (185 votes), and open-source license (187 votes).

Contrasting the results for negative and positive characteristics, it can be observed that they mostly corroborate each other, as shown in Figure 10. The most positive one (187 votes) is open-source licensing, which is the opposite of the most negative one (closed software, 209 votes). Good availability of libraries, considered positive with 185 votes, is the opposite of low availability of libraries, which had 175 votes as a negative feature. Good documentation was considered positive with 143 votes, while insufficient documentation was considered negative with 180 votes. Lastly, complicated configuration (187 votes for negative characteristic) can be contrasted with easy to setup configuration and easy to install/update libraries, with 133 and 139 positive votes, respectively.

Besides these features, a factor that may be hurting the adoption of C++ package managers is the lack of awareness. It was observed that the developers of C++ open-source projects are not aware of the existing tools: Biicode, Build2, Conan, CPM, Hunter, Mason, Meson, Pacm, and Yotta. In fact, 89.05% of respondents do not know any of these tools.

Considering the current methods for handling dependencies, the favorite ones are: system package manager (117 votes, 34.11%), header-only libraries (62 votes, 18.08%), and git submodules (40 votes, 11.66%). This result confirms the participants' preference for system package managers, which is the method adopted in 69.05% of the surveyed projects. On the other hand, only five projects (14.29%) make use of git submodules to handle dependencies, and none of them use third-party header-only libraries. Moreover, during this study, none of the analyzed projects were using any of the presented C++ package managers. This is further evidence of the low adoption of C++ package management tools.

In the next section, the discussion continues with considerations regarding validity threats.

6 THREATS TO VALIDITY

In order to prevent inadequate explication of constructs, the questionnaire was checked for consistency; alternative answers were precisely specified; and synonyms and examples were provided. Additionally, slangs and popular expressions were removed. The questionnaire was validated by native and non-native English speakers. Additionally, a pilot study was conducted in order to verify potential confusing points. The consistency between answers indicates that the respondents were able to understand the intent of the questions.

In order to reduce the odds of bias from researchers' expectancies or hypothesis guessing, the following measures were taken: balanced scales were provided; the text of the questions was carefully written as to not influence the answers; the questionnaire was revised by experts with no stakes in the study.

In order to mitigate the threat of non-response by specific groups, recommendations from Sivo et al. [15] were adopted: follow-up reminders were dispatched, incentives were provided, authority was evidenced, confidentiality was explicitly declared, and anonymity was made possible.

Aiming to mitigate researchers' conclusions bias, we opted to present a full breakdown of the answers here discussed, instead of just reporting descriptive statistics.

When analyzing the results of this survey, it is pertinent to consider the characteristics of the sample. Most of the projects they contributed to are very small: the outliers are MongoDB (89), Kodi (115), Minetest (150), OpenCV (176), Ceph (226), LibreOffice (265), and Firefox (834). It is also important to consider the response rate for each project — a higher number of responses from Firefox (21.28%) may have influenced the results of this survey. Additional analysis is required in order to measure and handle such influence. Therefore, generalization of the findings here reported should be made with caution.

Moreover, the survey was restricted to open-source projects, even though C++ is still largely used in closed-source commercial projects. As all considered tools in this study are open-source, we do not see this as a major threat since in many cases an open-source technology first gains traction within open-source communities before its adoption by the closed-source industry.

7 CONCLUSIONS AND FUTURE WORK

In this empirical study, we have presented a survey on C++ dependency management, aiming to elucidate the factors that are hurting the adoption of C++ package managers. The survey had responses from 343 contributors of 42 OSS projects, spread across ten categories, and of varying sizes.

Through preliminary observations, we had noticed that C++ package managers are hardly adopted in practice. This impression found corroborating evidence in this study: none of the surveyed projects adopt a C++ package management tool, and 89.05% of respondents do not know any of these existing tools. This lack of awareness might be an important driver of this low adoption since respondents seem to have a positive stance regarding this kind of tool, as discussed in section 4.5. Being open source tools, without formal backing from companies, these tools' future are endangered if they remain unknown to the broad audience, since it is hard for them to evolve without feedback and contributions from the developer community.

According to our survey, system package managers are currently the most used method for handling C++ dependencies. While it has advantages (e.g., ease of use and large availability of packages),

[5] http://blog.biicode.com/biicode-just-the-company-post-mortem

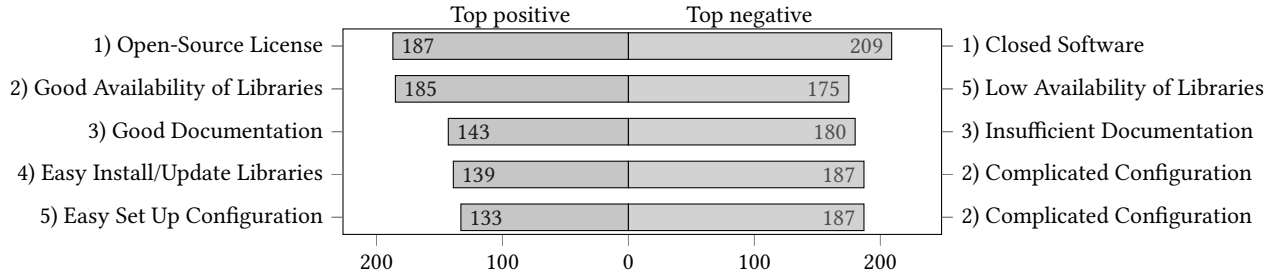

Figure 10: Top positive and top negative features presented side by side.

there are several different operating systems with their own repositories and policies, thus it is not an optimal cross-platform solution, so inflicting more effort and cost to maintain different sets of configurations, requiring workarounds for cases where features are not supported in some platforms and missing packages. In particular, some software development methodologies [19] recommend avoiding reliance on implicit system-wide packages, endorsing explicit dependency declaration and dependency isolation tools during execution.

In future work, we plan to perform statistical analysis on the gathered data in order to identify significant correlations. For instance, is there a correlation between respondents' stance towards language-specific package managers and their age, experience, or their usage of other languages? Additionally, we expect to perform follow-up interviews with selected respondents and professional developers in order to further explore the adoption factors.

Some aspects of C++ dependency management were not considered in this study, such as dynamic vs. static linking, packaging, and security. This was intentional as our focus is on understanding the low adoption of C++ package managers. If usage of these tools gains traction, however, the aforementioned issues will become even more relevant.

The C++ ecosystem is a thriving and heterogeneous one. The accidental complexity of managing dependencies may be undermining innovation and reducing maintainability, especially for cross-platform projects. We look forward to more homogeneity in dependency management for C++ projects across the many possible combinations of operating systems, compilers, and build systems.

ACKNOWLEDGMENTS

The authors would like to thank all survey participants for their collaboration and feedback.

REFERENCES

[1] Tiobe index for ranking the popularity of programming languages, august 2016, 2016.
[2] P. Abate, R. DiCosmo, R. Treinen, and S. Zacchiroli. MPM: a modular package manager. *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, pages 179–188, 2011.
[3] L. Courtès. Functional package management with Guix. *European Lisp Symposium*, 2013.
[4] G. Dos Reis, M. Hall, and G. Nishanov. A module system for C++ (revision 2). 2014.
[5] A. Fink. *The survey kit (Vols. 1–9)*. Thousand Oaks, CA: Sage, 1995.
[6] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. The Spack package manager: Bringing order to HPC software chaos. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.
[7] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, and M. Irlbeck. On the extent and nature of software reuse in open source Java projects. *Top productivity through software reuse*, pages 207–222, 2011.
[8] G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in open source projects: an internet-based survey of contributors to the Linux kernel. *Research policy*, 32(7):1159–1177, 2003.
[9] M. F. Krafft, K.-J. Stol, and B. Fitzgerald. How do free/open source developers pick their tools? a delphi study of the Debian project. *International Conference on Software Engineering (SEIP Track)*, pages 232–241, 2016.
[10] H. Muhammad, F. Mascarenhas, and R. Ierusalimschy. LuaRocks - a declarative and extensible package management system for Lua. *Brazilian Symposium on Programming Languages*, pages 16–30, 2013.
[11] A. Neitsch, K. Wong, and M. W. Godfrey. Build system issues in multilanguage software. *28th IEEE International Conference on Software Maintenance (ICSM)*, pages 140–149, 2012.
[12] S. L. Pfleeger and B. A. Kitchenham. Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26(6):16–18, 2001.
[13] S. Raemaekers, A. van Deursen, and J. Visser. An analysis of dependence on third-party libraries in open source and proprietary systems. *Sixth International Workshop on Software Quality and Maintainability, SQM*, 12:64–67, 2012.
[14] C. Schmitz et al. LimeSurvey: An open source survey tool, 2012.
[15] S. A. Sivo, C. Saunders, Q. Chang, and J. J. Jiang. How low should you go? low response rates and the validity of inference in is questionnaire research. *Journal of the Association for Information Systems*, 7(6):17, 2006.
[16] M. Sojer and J. Henkel. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems*, 11(12):868–901, 2010.
[17] M. Tiller and D. Winkler. impact - a Modelica package manager. *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, (096):543–548, 2014.
[18] G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin. Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, 36(2):649–676, 2012.
[19] A. Wiggins. The twelve-factor app, 2017.
[20] D. Wu, L. Chen, Y. Zhou, and B. Xu. How do developers use C++ libraries? an empirical study. *Proceedings of the Twenty-Seventh International Conference on Software Engineering and Knowledge Engineering*, pages 260–265, 2015.