UNIVERSITÉ DE GENÈVE

Archive ouverte UNIGE
https://archive-ouverte.unige.ch

Master 2022

Open Access

This version of the publication is provided by the author(s) and made available in accordance with the copyright holder(s).

Current Internationalization, Localization and Quality Assurance Practices in Open Source Software: A Case Study on Image Editors

Nedungadi, Anupama

How to cite

NEDUNGADI, Anupama. Current Internationalization, Localization and Quality Assurance Practices in Open Source Software: A Case Study on Image Editors. Master, 2022.

This publication URL: https://archive-ouverte.unige.ch/unige:164442

© This document is protected by copyright. Please refer to copyright holder(s) for terms of use.

Anupama Nedungadi

Current Internationalization, Localization and Quality Assurance Practices in Open Source Software: A Case Study on Image Editors

Directrice: Lucía Morado Vázquez

Jurée: Silvia Rodríguez Vázquez

Mémoire présenté à la Faculté de traduction et d'interprétation (Département du Traitement informatique multilingue (TIM)) pour l'obtention de la Maîtrise universitaire en Maîtrise universitaire en traduction et technologies (MATT), mention localisation et traduction automatique

Université de Genève

Année académique 2021-2022

Août 2022

Déclaration attestant le caractère original du travail effectué

J'affirme avoir pris connaissance des documents d'information et de prévention du plagiat émis par l'Université de Genève et la Faculté de traduction et d'interprétation (notamment la *Directive en matière de plagiat des étudiant-e-s*, le *Règlement d'études des Maîtrises universitaires en traduction et du Certificat complémentaire en traduction de la Faculté de traduction et d'interprétation* ainsi que *l'Aide-mémoire à l'intention des étudiants préparant un mémoire de Ma en traduction*).

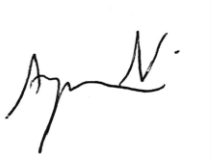
J'atteste que ce travail est le fruit d'un travail personnel et a été rédigé de manière autonome.

Je déclare que toutes les sources d'information utilisées sont citées de manière complète et précise, y compris les sources sur Internet.

Je suis consciente que le fait de ne pas citer une source ou de ne pas la citer correctement est constitutif de plagiat et que le plagiat est considéré comme une faute grave au sein de l'Université, passible de sanctions.

Au vu de ce qui précède, je déclare sur l'honneur que le présent travail est original.

Nedungadi Anupama



Brig-Glis, le 10.08.2022

Acknowledgements

First and foremost, I would like to extend my deepest gratitude to Lucía Morado Vázquez, my supervisor, for patiently guiding me through the whole process, for expertly answering all my questions, and being so supportive and understanding throughout all of my mental health struggles. She really was more of a mentor than a supervisor to me. I would also like to thank Silvia Rodríguez Vázquez for taking the time and effort to be the examiner of this thesis.

Next, I would like to express my gratitude to all the developers who filled out the questionnaire, some of whom even went out of their way and contacted me to give me more in-depth information and advice. This thesis would not have been possible without their help.

I am also grateful for Lesley at GitHub support and Manu Abraham at Atlassian support who were kind enough to answer all my questions and help me better understand their platforms.

A big thank you goes to my friends Anissa, Eva, Fanny, Hannah, Ivana, Maya and Sarah, who always offered hugs and words of encouragement whenever I needed them. They believed in me even when I didn't believe in myself.

Lastly, I would like to thank Arun, my boyfriend turned husband, my sister and my parents for being the best support system I could have asked for. Words cannot express how much you mean to me and I would not be here today if it weren't for you.

Abstract

This thesis primarily aims to identify the current internationalization, localization and quality assurance practices in multilingual open source software. A secondary research objective is to determine the reasons why developers might choose not to localize their monolingual software. The data was gathered through a questionnaire targeted at developers of image editing software specifically. There were 61 participants, out of which 32 were developers of localized software and 29 were developers of monolingual software. It was observed that current practices in open source internationalization and localization have not evolved much in recent years: open source is still highly volunteer-based and Gettext PO is still the prevalent file format. The best practices in the field are also not followed in terms of externalising hard-coded text and providing sufficient resources for translators. Furthermore, quality assurance and planning in particular are usually assigned low priority. Lastly, there seems to be a lack of awareness of localization in the open source community.

Keywords: open source, localization, internationalization, quality assurance, image editor

Table of contents

List of figures

List of tables

1 Introduction

What do Android, Audacity, GIMP, Linux, Mozilla Firefox and VLC Media Player have in common? While they are all very popular applications, they also all happen to be open source. In recent years, open source software has become more mainstream, and it is undeniable that the open source movement has given rise to software of incredibly high quality that compares to or is even superior to commercial applications. Many companies and businesses have recognized the advantages of open source software and have adopted them on a long-term basis.

Because this movement has been gaining in popularity, there have been numerous studies on this topic. However, both the intersection between open source and internationalization and the one between open source and localization have not been explored much. While there are research papers that discuss open source localization (Arjona Reina et al., 2013; Alshaikh et al., 2015), no studies seem to have been conducted in the last five years, which represents a gap in literature since the technology sector evolves very quickly.

There are some recent studies regarding quality assurance in open source (Bahamdain, 2015; Alami et al., 2018; Sirshar et al., 2019; Peslak & Hunsinger, 2020), however, to the best of the researcher's knowledge, none of them seem to specifically look at open source software that has been localized or internationalized.

Furthermore, research on open source software tends to focus on more popular and successful projects, whereas smaller projects get ignored, even though arguably the former do not represent the majority when it comes to all of the existing open source projects.

This thesis therefore aims to fill the gap that exists regarding internationalization, localization and quality assurance in multilingual software. In order to be able to analyse how these aspects are approached in both less popular and well-known programs, a category of software was chosen. This thesis will focus on image editing software in particular, mainly for three reasons: Firstly, there are many highly successful image editors (GIMP arguably being the most popular one), but it will not be difficult to find smaller projects, too. Secondly, image editors usually also have a graphical user interface with localizable text strings. Lastly, because image editors are not targeted towards a

specialised audience, this will make the results more generalisable to other types of software as well, compared to programs that are used in specialised fields such as engineering or medicine. However, there will not be a strong focus on image editing software in this thesis. The goal is to render the results more significant when comparing software of different sizes and ultimately try to generalise, which can be done by limiting the programs that are analysed to one category and therefore eliminating the variable relating to the type of software.

This thesis therefore aims to provide an overview of the current practices regarding internationalization, localization and quality assurance in open source software, specifically image editors. A secondary research objective is to identify the reasons why developers of monolingual software might not consider internationalization and/or localization.

The main structure of this thesis is as follows: Chapter 2 (Literature review) describes the internationalization and localization practices from an industry perspective and then goes into detail about the differences between open source and commercial localization. It also describes how quality is achieved within open source, albeit not from a multilingual point of view. The third chapter (Methodology) goes into detail about how the list of open source image editing software was compiled and how the data was collected by asking developers of the software to fill out an online questionnaire. In Chapter 4 (Results and discussion), the answers from the questionnaire will be presented and discussed, and the final chapter (Conclusion) will provide a summary of the results as well as present the limitations of the study and topics that could be examined further.

2 Literature review

The literature review is split into four main sections. In the first section (§ 2.1), the concept of internationalization will be discussed, alongside two important areas of internationalization and tools that can be used during the process. Section 2.2 goes into detail about localization, specifically the levels of localization and the different technologies that can render the process simpler and more effective. The third section (§ 2.3) will talk about open source software and its characteristics, followed by how open source localization differs from commercial localization. The fourth and final section of the literature review (§ 2.4) will discuss the common quality assurance practices within open source, what makes it difficult to achieve quality in open source and what the success factors are.

2.1 Internationalization

Internationalization, also referred to as i18n, is "the design of software code bases and resources that allow an application to be adapted to various locales without requiring changes to the code base" (GALA, 2020a). It therefore takes place before localization and will simplify and speed up the localization process.

The main idea behind internationalization is the separation of content and code. Through this step one should be left with software that "does not contain any language or culture-dependent information" (Abufardeh, 2009, p. 3). This includes the separation of hardcoded translatable text strings from the executable code, meaning that any text strings that appear to the end user and therefore would need to be translated (such as user interface elements, dialog boxes, tooltips and help files) are extracted into a separate resource file. If the translatable text is hardcoded, there is a high risk of damaging the source code in the process of localizing the software, as the translators would need access to the entire source file and most of the times, they do not have the same technological knowledge as developers to be aware of what can and must be translated and what should be left untouched.

However, there are many more aspects to consider when internationalizing a software. This next section will explore the following two areas of internationalization: the locale and the graphical user interface.

2.1.1 Locale

While people in different countries or regions around the world sometimes speak the same languages, their culture, perception of the world and how they use the language can differ significantly. Abufardeh (2009, p. 62) defines a locale as "the collection of features of the user's environment that is dependent upon language, country/region, and conventions". This means that Brazilian Portuguese and European Portuguese are two different locales, same goes for French spoken in Canada, French spoken in France and French spoken in Switzerland. Most of the times, they are displayed in the following format: 2 or 3-letter language code, dash, 2-letter region code (e.g. pt-BR, pt-PT).

In the following paragraphs, a few of the local differences shall be discussed.

Calendar and date/time formats

While most of the Western world uses the Gregorian calendar, in other parts of the world lunar calendars are the norm. An example of the latter is the Islamic calendar, also known as Hijiri. Therefore, if the software targets an audience of the Middle East, it should be able to display both the Gregorian and the Islamic calendar. (Abufardeh, 2009, pp. 62-63)

There are also different ways to display dates (Esselink, 2000, p. 33). In British English for example, the date format is dd/mm/yyyy, whereas the United States uses mm/dd/yyyy. The same goes for long dates. In US-English it is January 1, 2008, in UK-English it is 1 January 2008 and in Standard German it is 1. Januar 2008.

Time formats can also differ depending on the region. While some countries use the 24-hour format, other countries use the 12-hour system with "AM" and "PM" (Pym, 2004, p. 125).

Lastly, Abufardeh (2009, p. 64) mentions that rest days or the first day of the week can differ too depending on the locale. In US-American English, the week usually starts on a Sunday, in Standard German it starts on a Monday whereas in the Hijiri calendar, the week begins on a Saturday.

Number and currency formats

Esselink (2000, p. 33) notes that number formats can differ depending on the locale, such as the decimal separator that is used. There are also different currency symbols: while in

the UK the £ symbol precedes the amount, in most European countries the € symbol is placed after the number.

Measurement systems

There are two main measurement systems today. Most of the world uses the metric system, which measures length by meter, volume by litre and weight by kilograms. The United States (and a small number of other countries) however use the US customary system (USCS) which derives from the Imperial system. The measurement units in the USCS are inch, foot, yard and mile for length, cup, pint, quart and gallon for volume and pounds and ounces for weight. Another main difference between the United States and the rest of the world is the temperature scale: while the US uses Fahrenheit, most other countries in the world use Celsius. (Abufardeh, 2009, p. 65)

Addresses and phone numbers

Both Pym (2004, p. 126) and Esselink (2000, p. 33) mention that address formats can be very different depending on the country. While there are some elements that are common like first and last name, street, apartment number and city, there are also some differences, such as the length of the postcodes or zip codes and the order of the different elements (Abufardeh, 2009, pp. 65-66). It is also important that phone numbers reflect the regional conventions since the prefixes vary in different countries (Roturier, 2015, p. 174).

2.1.2 Graphical user interface (GUI)

In terms of internationalization and localization, one of the most important parts for the end user is for the graphical user interface (GUI) to be available in their native language (Abufardeh, 2009, p. 52). In the following, some of the most important aspects will be discussed.

User interface elements

According to Esselink (2000, p. 3), an important aspect of internationalization is to remove any hard coded text from the source code. The elements of the user interface that

should be checked and “neutralized” include windows, forms, dialog boxes, message boxes, menus, toolbars, text boxes, status bars, error messages, help balloons, and tool tips (Abufardeh, 2009, p. 53).

Another essential part of this step is making sure that the layout of these elements is flexible so that they can accommodate longer text strings and will not cut off any of the text in the target language if the translation is longer than the original. English for example is a very compact language, and according to Ramler and Hoschek (2017, p. 387), text strings can be 10 to 100 percent longer in the target language than they were in English. For some target languages like Japanese, Chinese or Thai, vertical expansion might also be necessary (Behrens, 2016, p. 93). Behrens (2016, p. 90) presents four possible solutions for the text expansion issue:

1. Static determination in the source code. The programmer manually leaves room for longer target text strings. Behrens recommends at least 200 percent of the source text.
2. Static determination in the resource file. The length of the container is adapted manually at the localization stage with the help of a localization tool.
3. The containers are dynamically adapted to fit the text with the help of a layout manager.
4. The containers are dynamically adapted to the hardware.

Apart from the text strings, sometimes there are also images, icons, and pictograms that need to be adapted to the target market (Abufardeh, 2009, p. 53). If they contain text, it needs to be translated. For this reason, images should ideally have a separate layer for the text, so that translators can easily replace the text in an image editing program. However, images can also have cultural meaning attached to them that is not represented textually. A graphic that works in one country can be less relatable to an audience in another country or even seem offensive (Dino, 2021).

Lastly, sounds can also have different meanings in different parts of the world. Therefore, general sounds should be used and those considered to be offensive should be avoided (Abufardeh, 2009, p. 180).

Text directionality

The software should be able to display languages that flow in different directions (Esselink, 2000, p. 337). While Western languages are written and read from left-to-right (LTR), Hebrew and Arabic are written and read from right-to-left (RTL). This affects not only the display of the text itself, but it means that the whole layout needs to be mirrored, as the eyes of someone who is used to read from right-to-left will gravitate towards the top right corner. This also includes direction-sensitive graphics and buttons like progress bars. Text directionality becomes especially complicated when dealing with bi-directionality, which is when some portions of an RTL-text are written LTR. In Arabic for example, numbers and embedded Latin language portions such as brand names are written and read LTR while the rest of the text is RTL (Murtaza & Shwan, 2009, p. 61).

In addition to the user interface, help files (online or offline) and documentation should be translated too (Abufardeh, 2009, p. 53). Therefore, any hard coded text must be removed and exported into resource files, and ideally all of the country specific elements mentioned above will be neutralized.

2.1.3 Internationalization tools

When designing a new software with the idea of localization in mind, there are many programming languages and development environments that facilitate internationalization (Murtaza & Shwan, 2009, p. 56). However, usually developers just choose based on whichever programming language and tools they are most familiar with (Murtaza & Shwan, 2009, p. 61).

More effort is required when existing software has to be internationalized. Fortunately, there are tools and websites that can simplify certain elements of the internationalization process.

Unicode code converter, UniView, Encoding converter and List character tools are all browser-based applications related to Unicode and the display as well as analysis of characters. There is also the W3C Internationalization Checker, which shows how well internationalized a web page is (W3C, 2016). Peng et al. (2009) developed a method based on lexical analysis to automatically convert the source code into supporting Unicode, which works best on C/C++ systems. Since the externalization of hard coded text

that needs to be translated is one of the most important tasks of internationalization, Wang et al. (2019) propose an automatic way of externalizing "need to translate" strings from the source code. Lastly, Murtaza and Shwan (2009, p. 67) mention the commercial tool Globalyzer, through which the source code of an existing software can be internationalized.

Finally, in the open source community, a lot of software programs use GNU gettext (Frimannsson & Hogan, 2005, p. 1), which is described as follows on their website:

> This package offers to programmers, translators, and even users, a well integrated set of tools and documentation. [...] These tools include a set of conventions about how programs should be written to support message catalogs, a directory and file naming organization for the message catalogs themselves, a runtime library supporting the retrieval of translated messages, and a few stand-alone programs to massage in various ways the sets of translatable strings, or already translated strings.
>
> (GNU Operating System, 2020)

GNU gettext also has its own file format called the Portable Object (PO) format, which will be further explored in section 2.3.1.

2.1.4 Conclusion

Internationalization is essential when it comes to developing multilingual software. In this step, translatable text strings are separated from the source code. Internationalization also ensures that the software supports different numerical, time, measurement and address formats and it accounts for text expansion occurring through translation. Any culture-specific elements such as pictures, colours, symbols or sounds are either "neutralized" or adapted to the target market. If the target language has a different flow of direction than the original, the code should be able to mirror the entire layout. Some of these processes can be automatized or sped up with the support of internationalization tools.

Ideally, internationalization should be part of the development stage of a product. According to Soh et al. (2016, p. 951), integrating internationalization tasks at the development stage proved to be more time-effective than performing them after software development. Ishida and Miller (2005) also stress the importance of considering internationalization early, so that the subsequent localization tasks can run faster,

smoother and will be less expensive. Furthermore, having a single internationalized source code makes the adaptation of software into multiple target markets easier, as the source code will no longer have to be modified for each target language and region.

If there are time or budget restraints and the software cannot be localized at a given moment, Murtaza and Shwan (2009, pp. 49-50) still recommend investing into internationalization to make future localizations possible.

2.2 Localization

GALA (2020a) defines localization (often abbreviated to L10n) as the "process of adapting a product or service to a specific locale". They further explain that the goal is for the end user to feel like the product has been created specifically for their language and culture.

Software localization goes beyond translating the user interface. As explained in the previous section about internationalization, each language has their particularities and oftentimes there are multiple variations within one language depending on the conventions and culture of a region or country.

During the internationalization stage, the text strings are extracted out of the source code and placed into separate resource files. These files are then sent to the localization teams for translation. Once the text as well as the graphical elements and the layout have been adapted for the target market, the files are sent back to the developers who will create a copy of the executable file and compile it with the localized resource file in order to create the target version (Dunne, 2015, p. 556).

2.2.1 Levels of localization

There are different levels of localization. As mentioned previously, the localization should not be limited to the graphical user interface (GUI), but for various reasons, sometimes a full localization is unfortunately not possible. Behrens (2016, p. 71) differentiates between three degrees of localization:

- First degree: only the GUI is localized (dialog boxes, menus, etc.).
- Second degree: in addition to the GUI, help files and documentation are also translated.

- Third degree: includes the localization of collateral material such as packaging, license contracts and the product websites.

A different approach to degrees of localization is presented by Abufardeh (2009, pp. 49-51). For a software product that has been developed for a US-English market, he presents six different levels of localization:

- U.S.-English product only: localization is not required, but people from other English-speaking countries might have some difficulties using the software due to different time formats, currencies, etc.
- English product handling European data: the software is not localized, but it is internationalized in the sense that it could handle single-byte character sets and the main different cultural conventions like date and time formats, currencies, etc.
- English product handling Far-Eastern data: the software is not localized, but it is internationalized in the sense that it could support languages like Japanese or Chinese.
- English product handling bi-directional data: the software is not localized, but it is internationalized in the sense that it could display both text that is read from left-to-right as well as text that is read from right-to-left, which is a requirement for languages like Arabic and Hebrew.
- Partial or full translation of the English user interface and documentation: this step ensures non-English speakers can use the software. Due to time or budget limitations certain elements remain in English.
- Full localization plus local market features: the product is perfectly tailored for the local market.

2.2.2 Localization tools

The localization teams work with a variety of tools. Their aim is to enable the translators to work faster without sacrificing quality, or better, while improving quality (Gaspari et al., 2015, p. 335). In the following sections, the most commonly used tools and technologies within the translation and localization domain will be explored further.

CAT tools

CAT tools (short for Computer-assisted translation) are programs designed to facilitate the translation process for human translators (GALA, 2020b).

A translation memory is a fundamental component of a CAT tool. Garcia (2015, pp. 71-73) describes it as "a database that contains past translations, aligned and ready for reuse in matching pairs of source and target units". The source text is split into segments, also called translation units. The translator then proceeds to translate each segment which gets stored into the translation memory database. Through this process, previously translated segments can be leveraged which considerably increases efficiency. Translation memories are one of the most common functionalities used by translators today. This was substantiated in a survey conducted under the framework of the OPTIMALE project ("Optimising Professional Translator Training in a Multilingual Europe"), where 38% of employers of translators considered the ability to use translation memory systems as "essential", while another 38% considered them as "important" (Garcés & Toudic, 2015, p. 192).

Another important feature that complements the translation memory is the alignment tool. Previously translated documents and their equivalents in the source language are fed into the system, where the alignment tool splits them into segments and subsequently imports them into the translation memory database (Garcia, 2015, p. 75). This way even documents which have been translated by other people can be taken advantage of. It also means that translators that are new to a project can produce text that is consistent in style and terminology to the work done previously.

Most CAT tools also include a terminology feature. This feature allows the translator to create one or multiple terminology databases and manually add terms, their equivalents as well as associated metadata to them. If the segment to be translated includes a term that has already been added to the database, it will be suggested to the translator, which will ensure a coherent translation. (Garcia, 2015, p. 73)

Depending on the CAT tool, different types of quality assurance modules are offered. Some of the most common ones are grammar and spell checkers. Most CAT tools also have a tag verification function, which ensures that the same tags and the same number of tags have been used in the target text. (Garcia, 2015, p. 76)

Nowadays, many CAT tools have an integrated machine translation feature, where the machine pre-translates the source text and the translator has to post-edit. If there are no matches found in the translation memory for a given segment, a machine translation will be suggested instead. The translator can then either reject this suggestion or accept and edit it if necessary. (Garcia, 2015, p. 81) Machine translation will be explained in more detail further down.

While most of the functions explained above are useful for both translation of documents as well as localization of software products, there exist localization tools that slightly differ from traditional CAT tools. Garcia (2015, p. 77) explains that while in the latter, a segment is usually equivalent to a sentence as the segments are separated by punctuation marks like periods, question marks, exclamation marks and colons, in localization, text strings are generally a lot shorter, which is why separating segments by punctuation does not make much sense. Some localization tools also allow translators to work in a WYSIWYG (What You See Is What You Get) environment, where there is a visual representation of the GUI of the software. This provides the translators with visual context and they can also make sure their translation fits the allocated space (Garcia, 2015, p. 77). Examples of such specialized localization tools include SDL Passolo, Weblate, Alchemy Catalyst, RC WinTrans, Multilizer, ResX Localization Studio and Visual Localize (Murtaza & Shwan, 2009, pp. 71-72; Arjona Reina et al., 2013, p. 163).

Translation management systems

Translation management systems, sometimes referred to as globalization management systems, are very similar to CAT tools in that they offer many of the same features such as segmentation, file format conversion, tag verification and pre-translation (Shuttleworth, 2015, p. 683). However, there are additional, more complex functions that focus on project management, project monitoring and the automation of workflows. Some examples include deadline management and mechanisms for controlling translation assets, assignment of roles within the project, generation of different reports, integration of email functions, facilitation of collaboration and translation quality assessment tools (Shuttleworth, 2015, pp. 683-684).

Some of the well-known localization management platforms include Transifex, POEditor, Lingobit Localizer, Lokalise, LingoHub, Crowdin, Webtranslateit.com and Smartcat (Murtaza & Shwan, 2009, p. 72; Arjona Reina et al., 2013, pp. 162-163).

Machine translation

Wang et al. (2019, p. 3539) define machine translation as "an automated translation of text without human involvement". There are multiple ways machine translation output can be used, often dependent on the desired level of quality. While it can be used as is without any human involvement in some cases, for example for internal communication purposes, most of the time the machine translation output will be edited by a human in a process called post-editing.

Studies have shown that human translators are more efficient in terms of number of words processed per day if they are post-editing machine translation output compared to translating "from scratch" (Muntés-Mulero et al., 2012, p. 1). Many companies and organizations take advantage of this in order to keep the costs as low as possible. For instance, there are many websites that use the Google Translate widget, which albeit its lower quality output is very useful, especially if the number of target languages is high (Murtaza & Shwan, 2009, pp. 76-77). The issue with most machine translation engines is that they are intended for general purpose and not domain-specific texts (Wang et al., 2019, p. 3515). However, some engines can be trained with documents related to a specific field, through which the quality of the output can be significantly improved. Besides Google Translate, DeepL and DeepL Pro, Microsoft Translate, Yandex Translate, Apertium, Amazon Translate and Systran are among the most popular machine translation engines (LingoHub, 2020; Singh, 2021; Lim, 2022).

Between the 1950s, which is when research on machine translation began, and the present day, there have been numerous different approaches on this topic. Qun and Xiaojun (2015) go into detail about the direct approach, the transfer approach, the interlingua approach, the rule-based system and the example-based system, however, these methods are mostly relics of the past. The two main approaches that are used today are statistical machine translation and neural machine translation, although recently most of the engines have switched to the latter. Explaining the different approaches goes beyond the scope of this thesis, but Qun and Xiaojun (2015) provide a comprehensive

overview about the history of machine translation, Moorkens (2018) explains the main differences between statistical and neural machine translation and Melby (2019) explores the future of machine translation and the question whether neural machine translation is the final solution.

2.2.3 Conclusion

Localization is often mistaken to be synonymous with translation. During localization, conventions, practices and preferences of a certain region and culture should be taken into account, so the end user feels as if the product has been developed for their target market, which includes not only translating text but also adapting pictures, colours, symbols and sounds (Hill, 2020).

There are numerous technological aids aiming to help the localizers work faster and more efficiently, such as CAT tools and translation management systems. Machine translation is also becoming increasingly popular, where translators no longer translate from scratch but rather post-edit the machine translation output.

The main challenges with localization are cost, time and quality (Ryan et al., 2009, p. 17). This is why partial localization is very common in open source software. Usually, contributors choose to translate the user interface first, then tackle the documentation and help files, and lastly tend to other material such as the website.

One of the main benefits of internationalization and localization has been described by Arjona Reina et al. (2013):

> Internationalization and localization in libre software projects are two of their most interesting advantages for dissemination, competition with proprietary alternatives, and penetration in new markets.
>
> Arjona Reina et al. (2013, p. 165)

Murtaza and Shwan (2009, p. 48) have also pointed out that releasing an already developed software product in other languages in order to penetrate new markets is oftentimes more profitable than developing a new software for the market the company or organization is already established in.

2.3 Open source software

This section will explain the key characteristics of the open source movement as well as the main reasons why people choose to contribute and use open source software over proprietary software. In section 2.3.1 the main differences between open source localization and commercial localization will be discussed.

As explained by Opensource (2022), "[o]pen source software is software with source code that anyone can inspect, modify, and enhance." The software is released under an open source license, which users have to consider when using or distributing the software. Most licenses allow the user to use the software however and for whatever reason they wish, others (such as copyleft licenses) are more restrictive in their conditions.[1] Most of the time, open source software is also free of cost.

The open source community is very ideologically driven and most contributors work as volunteers without any monetary compensation. Stewart and Gosain (2006, pp. 294-295) identified 15 principles that are part of the open source ideology and categorized them into norms, beliefs and values, as listed in Table 1.

There are various reasons as to why people prefer to get involved in open source software compared to proprietary software. One of the reasons is control. Due to the nature of open source licenses, people can modify the source code however they want and adapt it to whatever they need it to be (Opensource, 2022).

Open source software is also considered by some to be more secure as anyone can spot bugs or errors and correct them immediately (Opensource, 2022). However, others argue that open source software is actually less secure than proprietary software as the community does not seem to accord much importance to this aspect during the development stage (Sirshar et al., 2019, p. 9).

Stability is another driving factor: because the source code is distributed publicly, the software will continue to be developed even if the initial contributors leave the project, which means that people can use open source software long-term. (Opensource, 2022)

[1] A comprehensive list of open source licenses can be found here: https://opensource.org/licenses/alphabetical (accessed: 3 August 2022).

Norms	Forking	Through forking, the project is split and the development of the forked project happens separately. One should not fork a project without good reason or re-labelling it.
	Distribution	One should not distribute code changes without first consulting the moderators or project managers.
	Named Credit	One should always openly give credit to contributors. Similarly, one should not remove the name of a contributor without their knowing.
Beliefs	Code Quality	Contributors believe that open source code quality is superior to that of closed software.
	Software Freedom	Contributors believe that results are better when the source code is available for everyone.
	Information Freedom	Contributors believe that information should be available to be copied, adapted and redistributed.
	Bug fixing	Contributors believe that the more people are involved, the quicker bugs in the code will be found and resolved.
	Practicality	Contributors believe that practical application is better than theoretical knowledge.
	Status Attainment	Contributors believe that status comes from recognition of other contributors in the community.
Values	Sharing	Sharing information and knowledge is highly valued.
	Technical	Technical knowledge (e.g. knowing multiple programming languages) is highly valued.
	Learning	Learning for the sake of learning and not being content with the minimum is highly valued.
	Cooperation	Voluntary cooperation is highly valued.
	Reputation	A good reputation is highly valued, which can be attained by participating in open source projects.

Table 1. Principles of the open source ideology (Stewart and Gosain, 2006, pp. 294-295)

Since it can be freely distributed, there is also a rapid evolution in open source. This in turn leads to more innovative software (Cánovas & Samson, 2011, p. 50). Another advantage of open source software is its transparency, which can be attributed to the fact

that the source code is available for everyone to see and also modify (Sirshar et al., 2019, p. 10). Van Wendel de Joode and de Bruijne (2006, p. 6) also assume it to be one of the most important factors when looking at the reliability of open source software.

Next, Otte et al. (2008, p. 1251) observed that 84.5% of respondents in their survey considered communication between developers and users as direct and efficient. This makes sense as most developers are themselves users of the software. It is also in line with the value of sharing information mentioned by Stewart and Gosain (2006, pp. 294-295). The communication between more experienced developers and contributors who are new to the project is also often dynamic, which leads to a very valuable knowledge transfer within the community (Otte et al., 2008, p. 1249).

Lastly, there is often a community formed around a particular software. Many people find the notion of exchanging ideas between like-minded people appealing (Opensource, 2022).

Naturally, open source software comes with its disadvantages as well, most of which relate to the process of quality assurance which will be discussed in section 2.4. Open source software is also rarely accompanied by clear and complete documentation, since contributors usually find the task of writing documentation boring and not rewarding (Alami et al., 2018, p. 66).

2.3.1 Commercial vs open source localization

While one can easily find literature on internationalization and localization as well as open source software, the combination of the two topics does not seem to have been studied as much, especially in recent years. Souphavanh and Karoonboonyanan (2005) talk about the benefits of localization from a humanitarian point of view and the advantages of choosing free and open source software (FOSS) over proprietary software for governments and policy and decision makers. They also present the history of localization in the Asia-Pacific region and talk about current localization efforts for languages including Chinese, Japanese, Korean, Hindi, Thai, Vietnamese, Malay and Khmer. Unfortunately, their work is from 2005 and might therefore be outdated.

Wolff (2011) created a manual on localization for people who would like to contribute to open source projects. He touches on open source software localization, how it is different

from regular translation and commercial localization and gives very practical advice for beginners on how to contribute translations to open source software projects.

Arjona Reina and Robles (2012) studied the localization practices in the Android project specifically. They found that Android does not have a dedicated infrastructure for localization and contributors have to hand in translations the same way they would additional code. They also observed that the people internationalizing and localizing the project seem to be the main contributors to the Android project in general and that there is no separate group that is responsible for these tasks.

Arjona Reina et. al (2013) provide an overview over localization practices in free, libre and open source software (FLOSS). They analysed a total of 41 projects that, similarly to this thesis, all differ in size and popularity. However, they did not choose a specific category of software, rather, their dataset consists of a variety of different software types such as operating systems and text editors. In contrast to this thesis, they did not contact the developers and collaborators to find out more about their stance on localization. They extracted the information about the software from project websites, documentation, mailing lists and related literature, rather than get the information directly from the people working on the project. Additionally, their focus was entirely on localization and they touched neither on internationalization nor on quality assurance. Their results showed that there are differences in the tools used by the localization teams: while some simply use plain text editors, others make use of more developed localization tools and platforms. They also found that while the methods of contributing translation vary greatly, most of the projects offer information on this process. This seemed to be the driving factor for having a well internationalized and localized software: projects that provide instructions about contributing translations are translated into more target languages.

Alshaikh et al. (2015) conducted a study where they analysed a randomized set made up of 2500 open source software projects hosted on SourceForge to see how many of them were localized, to assess the quality of the software localization and finally to find out how long the localization process takes. They observed that more popular projects (measured by weekly downloads) were more likely to be localized, that partial localization was very common and that most of the time, localization was performed months after the software was developed.

Literature suggests that the technical aspects of localization do not differ much when comparing open source and commercial localization apart from the lower cost and the licensing conditions (Souphavanh & Karoonboonyanan, 2005, p. 2). However, there are some other key differences, notably the file format and the volunteering translation environment, which will be explored in the next two sections.

File formats

One of the main differences between commercial and open source localization is the file format that is used. In open source localization, the Gettext PO file format is still the de facto standard. (Arjona Reina et al., 2013, p. 161). The most commonly used interchange file format in commercial localization, however, is XLIFF.

Frimannsson and Hogan (2005, p. 10) present three different variants of the PO file format: the PO template file (POT), the regular bilingual PO file and the PO Compendium file. A POT file can be compared to a table that contains both source and target language strings, but where the fields for the translation are left empty. A PO file is a copy of the POT file, the only difference being that it also contains the target strings. Lastly a PO Compendium file is a document that contains translations from multiple PO files.

The process of internationalization using Gettext is illustrated in Figure 1.

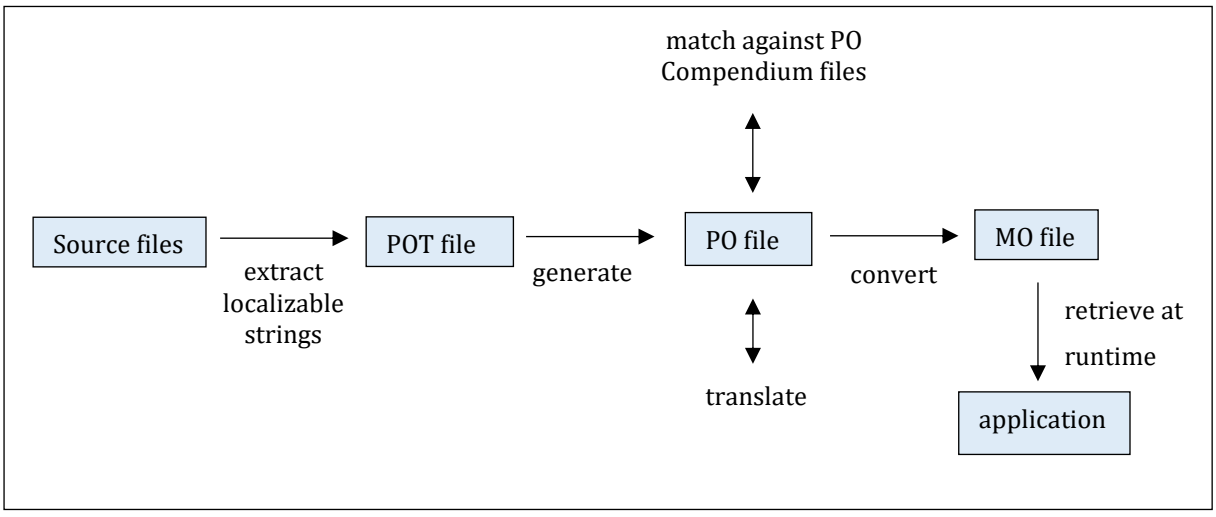


Figure 1. Process of internationalization using Gettext

Firstly, a POT file is generated by extracting the localizable strings from the source code. Then, a PO file is created for each target language. This PO file is sent to the translators who add the target strings by using programs that support the PO format. During this process, the PO file can get matched against PO Compendium files if available, to see

whether the localizable strings have been previously translated and whether the translation is acceptable in this case. After the translation is completed, the PO file is converted into a binary Machine Object (MO) resource file, which is retrieved by the application at runtime. (Frimannsson & Hogan, 2005, p. 10)

XLIFF is a file format that is based on the markup language XML. XLIFF stores translatable text and the corresponding translation, which makes it a bilingual document (Torres del Rey & Morado Vázquez, 2015, p. 588). It can also store a range of different metadata such as the source language (<srcLang>), the target language (<trgLang>) or the directionality of the content (<dir>) (OASIS, 2014). There have been efforts to bring the XLIFF standard into the open source world. Frimannsson and Hogan (2005, pp. 14-15) point out six main advantages of XLIFF over PO:

1. As mentioned above, XLIFF can store many different types of metadata on different levels, while this functionality is limited for PO files.
2. Some of the information that can be stored in the elements and attributes of an XLIFF file relates to workflow, which can simplify the localization process. An example of such an attribute would be <state>, which indicates the state of the translation (initial, translated, reviewed, final).
3. When using XLIFF in combination with TMX, sharing translation memories across different projects becomes easier. TMX is a translation memory format that was developed to ensure minimal loss of metadata during the exchange of translation memories between different tools.
4. Most localization tools used by the open source community are based on the Gettext toolkit, which in turn means that it strongly leans towards the PO file format. Making sure that open source localization tools also support the XLIFF format could lead to a decoupling of localization technologies, meaning that Gettext would not have to be the default technology in open source.
5. Since XLIFF is based on XML, this opens the door to many other tools and technologies that offer support for XML. One can for example use an XSLT stylesheet to present an XLIFF file in a table.
6. Lastly, XLIFF cannot only handle textual data but also binary data, meaning that for instance, executable files could technically be stored in the XLIFF files.

Finally, it should be said that not all projects, be it open source or commercial, use either PO files or XLIFF. There exist other languages and file formats such as YAML, JSON or Android XML, most of them being tool or project specific.

Volunteer translation environment

Another big difference between commercial and open source localization is the recruitment of translators and their profile. In a commercial setting, translators are either directly employed by the company or organization responsible for the software product or they work with them on a freelance basis. Either way, they are selected based on their language and technological skills and are compensated for their efforts.

In open source localization, the translators (as well as other contributors) are mostly volunteers.[2] The concept of volunteer translation is also referred to as crowd-sourcing, crowd-translation, collaborative translation, community translation or translation crowdsourcing (Arjona Reina et al., 2013, p. 155; Pym, 2011, p. 97; Jiménez-Crespo, 2018, p. 72). Jiménez-Crespo (2017, p. 25) defines "translation crowdsourcing" as "collaborative translation processes performed through dedicated web platforms that are initiated by companies or organizations and in which participants collaborate with motivations other than strictly monetary". Pym (2011, p. 97), however, recommends the use of the term "volunteer translation" because it puts the focus on the person and it emphasizes the main difference between a professional and a non-professional translator, which is the monetary compensation. Furthermore, the concept of volunteering is a big part of the ideology that surrounds open source software, as mentioned in the beginning of section 2.3.

Some key differences between a volunteer translation environment and a traditional commercial setting were identified by Sarigül and Ross (2020). They conducted a study

[2] While volunteer translation is mostly implemented by the open source community, there also exist some proprietary projects that take advantage of this model, the most popular of them being Facebook (Cánovas & Samson, 2011, p. 47).

in which they compared how a volunteer community and a professional localization company approach the process of video game localization.[3]

Firstly, they found that in both groups the role of the project manager is very important. However, while the project manager in the localization company delegates the tasks and assumes a more authoritative role, the project manager in the volunteer community acts more as a moderator, who guides new translators on how to contribute, provides them with resources such as terminology databases and translation memories, and never puts any pressure on the contributors to translate a certain amount within a certain time.

In both groups each member is seen as a valuable and equally important asset to the team, although in the localization company, members are usually expected to better justify their opinions on a specific topic. Furthermore, there is a lack of deadlines and hence pressure in the volunteer community.

The last difference between the two groups relates to the translation environment. According to Kelly et al. (2011, p. 90), who reviewed 104 volunteer translation environments in their study, there are three different environments: (i) wiki- or forum-based, (ii) database-driven and (iii) full-blown. The wiki- or forum-based environment is the simplest out of the three: it provides the translators with some basic instructions and a platform to ask questions. In the database-driven environment translators are given a simple interface or even a full-fledged dashboard for translating. Lasty, the full-blown environment is the most elaborate and also the costliest one as the name suggests. Here, the environment is designed similarly to a translation management system where translators are provided with translation memories, terminology databases and access to machine translation. Sarigül and Ross (2020, pp. 13, 16) noted that while the professional localization company chose the full-blown model, the volunteer community settled for a hybrid between the wiki- or forum-based and the database-driven environment.

Arjona Reina et al. (2013) found that the free, libre and open source software (FLOSS) community implements a few processes in order to exploit the full potential of the

[3] The volunteer community that is analysed in this paper is not open source; it is part of the video game distribution service Steam, which in turn is owned by the company Valve Corporation. Sarigül and Ross also focus on video game localization, which varies slightly from software and web localization.

volunteer translators. Firstly, developing helpful tools and providing instructions on how to contribute translations are very important.

They also noted that developing or using existing collaborative platforms is helpful. There is no one way to approach this: some projects use an external, web-based collaborative environment (such as Transifex or Pootle), others have set up independent servers within these environments, and some projects develop their own localization tools.

They further explain that giving credit to translators has proven to be effective in attracting new volunteer contributors and keeping them motivated. It was found that if a project gives credit to translators, it is more likely to be translated into a large number of languages.

Lastly, they argued that creating a translator community around the project has been beneficial in FLOSS localization. This can be done through creating language teams, mailing lists and forums, where contributors are free to exchange thoughts with each other.

Wolff (2011, p. 4) explains that thanks to the volunteering environment, contact between developers and localizers tends to be close, which means that the translators get quick feedback on their questions and suggestions. He also mentions that translators might have more freedom compared to their counterparts working on proprietary software. Since open source projects are reliant on volunteers, they are very welcoming and accommodating to translators. This explains why open source software is often translated into languages that would not be considered very profitable in the commercial sector.

2.3.2 Conclusion

Contributors for open source software are mostly volunteers. As such, they are mostly very ideologically driven and believe that open source software is superior to proprietary software regarding aspects such as control, security, stability, innovation and transparency. Many also enjoy the sense of community that is built around open source software.

The structure of these open source communities, the roles assumed by contributors and the general approach to localization vary from their counterparts for proprietary

software. For example, the most popular file format used in open source localization is the Gettext PO file, while this format is rarely adopted in closed-source software. Generally, the processes are also less rigid and there is little to no pressure to deliver translations within a certain time frame.

In order to increase the likelihood of the software being localized, it is necessary to keep the contributors interested and to also attract new volunteers. There are multiple ways to do this: openly giving credit, providing clear instructions on how contribute, and offering collaborative platforms are some of the most popular options.

Furthermore, contributors of open source software are often users themselves, which is an advantage when it comes to localization. People with the necessary skills can freely decide to contribute translations and make the software available in their native language, while this is usually not possible with closed-source software as the decision to localize is made by the company or organization and not the users.

2.4 Quality assurance

Quality assurance is a very important step in the project life cycle. In the closed-source software, it is marked by extensive planning, high-level testing and firm control throughout the project (Alami et al., 2018, p. 63). The quality assurance practices in the open source community however are quite different because of the distinct characteristics of open source software that were discussed in section 2.3.

There are numerous factors that make achieving quality in open source software difficult. Firstly, there is usually little planning and scheduling prior to the programming stage and the development methodology is not clearly defined (Aberdour, 2007, p. 59).

While closed-source software has well defined and measurable goals, this is not normally the case in open source (Aberdour, 2007, p. 59). Similarly, the requirements are defined by programmers and can often be unclear (Aberdour, 2007, p. 59; Bahamdain, 2015, p. 462). Aberdour (2007, p. 59) also argues that in open source, there is "no formal risk assessment process" and that "empirical evidence regarding quality is not collected".

The way open source communities are designed means that when issues arise, there is usually not a person who is solely responsible for the resolution; rather, the

responsibility is distributed to multiple developers (Bahamdain, 2015, p. 461). This can slow down the process and make quality assurance more difficult.

Another issue relates to the fact that open source communities rely mostly on volunteers. Since quality assurance tasks such as writing documentation or testing tend to get less visibility and credit, they are performed much less frequently. Furthermore, volunteers can pick and choose their tasks based on their skill and even more importantly their enjoyment. Because these tasks are generally considered less fun and enjoyable than coding tasks, they often take a back seat. This also becomes a problem regarding the testing process, as the attitude of open source developers is typically "features first and quality later", which could result in errors being found much later down the line (Alami et al., 2018, p. 66). The testing process will be discussed in more detail in section 2.4.1.

Despite all of the aforementioned challenges, there are numerous examples of successful and reliable open source software. The following section goes into detail about some of the main quality practices within the open source community in general as well as some other factors that have been linked to reliable software specifically.

2.4.1 Common practices

There are two processes that are commonly practiced within the open source community in regard to quality assurance: testing and peer review.

Testing

Testing the software is one of the most important tasks when it comes to quality assurance. There are various ways to categorize the different testing levels, however, many papers have referred to Esselink's (2000, pp. 147-154) categorization: internationalization testing, linguistic testing, cosmetic testing, functionality testing, and delivery testing.

- According to Esselink (2000, p. 147), *internationalization testing* (also called localization-readiness testing and localization ability testing by other authors) ensures the software fulfils all internationalization requirements and is ready for localization. As part of this process, pseudo-translation testing can be performed (Abufardeh, 2009, p. 118). This involves randomly replacing English characters

with Asian, Cyrillic or Arabic letters and symbols, through which can be verified whether other languages can be correctly displayed. Similarly, extra characters can be added to make sure that text expansion is accounted for. In addition to that, pseudo-mirroring testing can be performed for software that should be able to handle bi-directional text (Abufardeh, 2009, p. 118). Here, the source text strings are left untouched, while the layout is switched to read from right-to-left.

- Esselink (2000, p. 150) describes *linguistic testing* as the process of checking all language-related aspects of the software. This includes verifying whether the translations are accurate within their context, whether the translations are consistent throughout the different components of the software, whether there is any untranslated text and whether the text wraps, hyphenates and sorts correctly. According to Abufardeh (2009, p. 176), most errors in localization occur in this area. Ideally, this type of testing should be performed by the translator.
- *Cosmetic testing* focuses on the user interface and involves making sure that there are no visual errors (Abufardeh, 2009, p. 122). Esselink (2000, pp. 151-152) notes that during this step it should be checked whether all menus, options and commands from the original software also appear in the localized version, whether all items fit on the screen regardless of the screen resolution, and whether there are any truncations or overlays.
- Abufardeh (2009, p. 173) describes *functionality testing* as the step during which it is checked whether the localization has introduced any issues that could impact the functionality of the software. These range from searching, sorting and reading databases to conversion of data type. The better the internationalization from the start, the less likely there are to be any functional errors that are due to localization, however, Esselink (2000, p. 152) recommends functional testing even for well internationalized software.
- *Delivery testing* is the final level of testing mentioned by Esselink (2000, p. 154), and it refers to checking whether the expectations set by the project sponsor at the beginning are actually met. It is however not relevant for this thesis since in the open source community there is no concept of having to deliver a product by a certain date, instead, there is a continuous improvement.

Because testing can be a very time-consuming task, there have been efforts to try and automatize this process. Ramler and Hoschek (2017, pp. 388-389) describe an automation approach for localization testing that consists of four steps. First, all of the screens of the graphical user interface (GUI) are opened using a GUI test script that has been developed for this purpose. The script is executed for both the original software in the source language as well as the localized version. Then, each screen is captured in a screenshot and data relevant to testing such as element type, displayed text, visibility status, position and size is extracted. In a third step, the extracted data from the original software is compared to the data from the localized version to ensure that there are no missing translations, that all the keyboard shortcuts are present and consistent, that the data complies with basic style guidelines, that there are no illegal or obsolete terms and that the GUI elements all appear correctly within the screen. Finally, a test report is generated, where screenshots from both language versions are compared side by side accompanied by details about the GUI elements as well as potential errors. When this method was applied to a software translated into Korean, 59 errors were found in addition to the ones previously identified by human testers, most of them being minor cosmetic errors.

While there exist automatic testing tools that are open source, literature is inconclusive about whether automatic testing is used within the open source community. This thesis will therefore attempt to shed light to this matter.

Peer review

While peer review is used in numerous fields as a means to evaluate other people's work, in regard to software, there are two types of peer review processes: the review-then commit (RTC) and the commit-then-review (CTR) (Rigby et al., 2014, p. 542). In the former, a contributor submits the code they have written to the developer mailing list. The code is then reviewed and adapted if needed by peers before it gets incorporated into the main code. In the CTR technique, contributions can be committed directly into the code base and they are only reviewed afterwards. This right is usually reserved for experienced and trusted developers of the project and is mostly applied for smaller bug fixes. This technique is less common than RTC as contributions can go unreviewed, however, it is more time-effective.

While testing and peer reviewing are two very common quality assurance practices, there are other methods and processes that have been linked to reliable open source projects. The following section will explore them in more detail.

2.4.2 Success factors

Van Wendel de Joode and de Bruijne (2006) tried to explain how open source software can be reliable and what factors into it. They came up with three hypotheses:

1. "The bigger the percentage of developers in an open source community who actually use the software themselves, the more reliable the software." If the developers are users, they will have a strong motivation to solve issues and improve the quality of the software.
2. "The more transparent the flow of information in an open source community, the more reliable the software." Developers and contributors will be more careful and intentional with their submissions to the project because their mistakes could be seen by anyone.
3. "The more popular the open source software, the more reliable the software." Having external oversight from professional peers and other interest groups can be an additional motivation for contributors to keep the quality at a high level. More popular projects will have more external oversight which in turn means more reliability.

For an open source project to be reliable and therefore of high quality, it usually also needs a sustainable community. Aberdour (2007, pp. 59-60) explains that most sustainable communities follow the onion model. In this model, there are four different groups of contributors: the core developers, the contributing developers, the bug reporters, and the users. There should only be a few core developers so that the project remains manageable and does not become chaotic. Contributing developers are responsible for new features and the maintenance of existing ones, they conduct peer-reviewing and also take care of some bug fixes, although the latter mostly falls into the responsibility of the bug reporters. The point of this onion model is to have defined roles within a clear structure, where people can move towards the core by increasing their involvement.

Usually, a high quality open source project has access to a large pool of volunteers. This is beneficial in multiple ways. First of all, Eric S. Raymond coined the phrase "given enough eyeballs, all bugs are shallow", which is to say that the more people contribute to the project, the quicker bugs and errors are found. The large community is also relevant when it comes to the releasing method. "Release early, release often" is a popular approach in open source. The software is released to the public before it reaches certain milestones and an optimal level of quality, and it is continuously improved over time (Zhao & Elbaum, 2000, p. 54; Aberdour, 2007, p. 62). Moreover, since open source software is so highly reliant on volunteers (see section 2.3.1), it is important that there are enough people who use, test and better the software.

Lastly, many high quality open source projects implement code modularity. The program is separated into different modules that work independently from each other, which means that contributors can work on one or few modules without having to grasp the entire code functionality. Multiple people can also work on the same issue without affecting each other, which can lead to a faster resolution of the problem. (Aberdour, 2007, pp. 60-61)

2.4.3 Conclusion

As explained in the sections above, quality assurance in open source looks very different from quality assurance in a commercial setting. There are many factors that make achieving quality in open source difficult, such as the lack of planning, the lack of well-defined and measurable goals, the lack of single responsibility for problems and the lack of motivation for contributors to work on quality assurance tasks because they get less visibility and are less enjoyable. However, most open source projects still implement some quality assurance practices, the most common being testing and peer reviewing. Some even go further and implement code modularity and establish practices in order to form a sustainable community and expand the pool of volunteers.

Quality assurance is given high priority in proprietary software development. The earlier these tasks are taken into account, the better the result will be. Contributors for open source projects generally do not seem to be very concerned with quality assurance. However, there are numerous projects that prove that achieving quality in open source is not impossible.

This chapter presented the concepts of internationalization and localization and provided an overview over the well-established practices in these fields, it went into detail about the most important characteristics of the open source movement and the differences between commercial and open source localization, and lastly explained how quality assurance in proprietary software development differs from open source software development. The next chapter will address the methodology used to conduct this study.

3 Methodology

The goal of this thesis is to provide an overview of the current internationalization, localization and quality assurance practices in open source software by the example of image editors. More specifically, it seeks to understand what aspects of these three fields were considered most important and what tools and technologies were used in the process. It also aims to explain how translators are recruited for localization as well as their working conditions. Lastly, while the main focus of this thesis is on localized software, it also intends to shine a light on the question why many developers have not internationalized or localized their project.

To address these research objectives, a questionnaire was developed, the design and structure of which will be addressed in section 3.3. The questionnaire was chosen because it allows the collection of large amounts of data and the analysis is straightforward because of its given structure. Furthermore, it can be used to gather information about attitudes and behaviours, all while also ensuring the participants' anonymity. (Saldanha & O'Brien, 2014, p. 152)

The questionnaire was sent to developers and contributors of open source image editor projects. Most of these projects were compiled through four different source code hosting platforms, while a small number of very popular image editors was added to the list ad hoc. This first phase of data collection will be explained in more detail in section 3.1.

In order to be considered for this thesis, they had to fit certain pre-defined criteria. Firstly, they had to match the definition of image editors, which was specially established for this thesis and will be further elaborated on in section 3.2. Secondly, they had to be released under an open source license. Thirdly, they had to be current and still evolving, which is why any projects that were last updated more than 24 months ago (before May 2019) were not considered. Next, the image editors had to be standalone programs as opposed to software components. Lastly, it was necessary for the software to have a graphical user interface (GUI), as command line tools are rarely localized in the first place.

Because it was not always possible to filter the results on the platforms according to the pre-defined criteria, the list of projects compiled in the first phase had to be manually filtered through and any programs that were irrelevant for this research had to be removed. This second step will be discussed in section 3.2.

3.1 Collecting the data

During the first stage, a large list of open source image editing software was compiled. In this section it will first be discussed how the four source code hosting platforms, notably SourceForge, GitHub, GitLab and Bitbucket, were selected. It will then go into detail about the data collection process on each of the platforms. Lastly it will be explained that in addition to the software obtained directly from these platforms, six other popular image editing programs were added ad hoc to ensure that the final list would also include larger and more prominent image editor projects.

3.1.1 Platform selection

There are many platforms that host open source software, both exclusively and non-exclusively, but because of the limited scope of a master's thesis, a small number had to be selected.

The decision was made not to focus on platforms that are specifically for open source projects because there also exist many sites that host both proprietary and open source software. It was therefore deemed important to include the latter in order to get an accurate representation of open source software.

Because it proved to be difficult to find a list of platforms that accommodate open source software from an official institution, a simple Google search was performed. From the search *source code hosting sites* 16 relevant blog posts and websites were selected, which mentioned 32 different platforms in total. These platforms were then sorted by most mentions.

As can be seen from Table 2, GitHub, GitLab, SourceForge and Bitbucket were by far the most popular and recommended, which is why they were selected for the data collection.[4]

[4] The full table with all of the blog posts and platforms can be found in Appendix A.

Source code hosting site	Total mentions (out of 16)	Source code hosting site (continued)	Total mentions (continued)
GitHub	16	Trac	2
GitLab	15	GNU Savannah	2
SourceForge	15	Buddy	2
Bitbucket	14	Assembla	2
Launchpad	7	RubyForge	1
Apache Allura	7	Tarc	1
Beanstalk	7	Google Code[5]	1
Gitea	7	TaraVault	1
Cloud Source Repositories	6	GitPrep	1
Gogs	6	Kallithea	1
Gitbucket	6	TuleapL	1
Phabricator	6	Azure DevOps Server	1
Gitkraken	5	Jfrog Artifactory	1
AWS CodeCommit	5	Helix Core	1
CodeGiant	3	Sourcehut	1
RhodeCode	2	Codeberg	1

Table 2. Source code hosting sites

The data collection on all four platforms took place in May 2021. Because these platforms do not offer the same search functions or filtering and sorting options, this process was not as linear and straightforward as initially hoped. There was some going back and forth between the platforms to ensure the results from the different sites were as comparable as possible to each other. In the following sections each platform and the difficulties relating to the data collection will be discussed in more detail.

3.1.2 SourceForge

The first platform that was tested was SourceForge. It is a website that hosts open source software and also offers a business directory list that makes it easy to compare different

[5] closed down in 2016 (DiBona, 2015)

software and services through filters and categories (SourceForge, 2022a). They claim to host over 502,000 projects and have millions of registered users.

There are multiple ways to browse through open source software on SourceForge. There is a search box available for queries, but there are also many filter options grouped into seven big categories, namely: (i) *OS* (operating system), (ii) *Category*, (iii) *License*, (iv) *Translations*, (v) *Programming Language*, (vi) *Status* and (vii) *Freshness*. Some of the filters are further divided into subcategories.

The first step was to explore the different subcategories within *Category* in order to find a list of image editing software. After some testing, it was gathered that the filter *Multimedia* > *Graphics* > *Editor* within the section *Category* would be most representative of image editing software. This filter matched more than 1000 projects.

Since the objective of this research is to determine the recent practices in internationalization and localization, the filter *Freshness: Recently updated* was also applied. The sorting option revealed that *Recently updated* on SourceForge meant that the project had been updated within the last two years, which was later taken into account when collecting data on the other platforms.

As this thesis does not focus on distinct operating systems or specifically on either desktop or mobile applications, the category *OS* on SourceForge remained unused. It also did not make sense to limit the list of projects to certain open source licenses or programming languages, which is why applying a filter within the categories *License* and *Programming Language* on SourceForge was not necessary.

An early version of the data collection from SourceForge only included software that was available in English and at least one other language. This was done through the filter category *Translations,* which filters the projects by natural languages. English was chosen as the common denominator because firstly, it was assumed that most programs would be available in English considering its status as a lingua franca, especially in the tech world, and secondly, because trying every combination of the 60 plus languages available on SourceForge would have been too time-consuming. Even with English as a point of departure, this task proved to be very tedious as for every language combination (49 in total) a new search had to be conducted and the projects had to be compiled into a list. Unfortunately, when consulting GitHub and GitLab, it was found that there exists no

reliable way to filter by natural language on these two platforms and that it was rarely disclosed in what languages the software was available. The approach was therefore changed to include monolingual software and the filter *Translations* was removed on SourceForge. More information on why this decision was taken can be found in section 3.1.3 about the data collection on GitHub.

The filter category *Status* was also taken into consideration, notably its subcategories *Production/Stable* and *Mature*. Unfortunately, with these filters the number of projects was almost reduced by half and even with this small list there was no guarantee that all of the projects listed would actually match the criteria for image editors established in the introduction of chapter 3. Furthermore, not all projects on SourceForge were categorized in terms of status, which meant that there could exist more software on this platform that could be considered stable.

It is also worth noting that it was attempted to obtain a list of image editing software through the search box by combining different search queries in combination with the filter *Freshness > Recently updated*. Image editors can also be described with other keywords such as *graphic editor* or *image manipulation software*. Unfortunately, stemming does not apply on SourceForge, which means that each keyword has to match the word in the name or description of the project exactly. For example, a project described as a *graphics editor* would not appear in the results of the search query *graphic editor*. Using an asterisk as a wild card character did not work either. This meant that unless many different and slightly nuanced search queries were conducted, there was a risk that many projects would go unnoticed, which is why the search box was discarded as a method to find projects on SourceForge. Admittedly, this risk prevails even with the method of only applying filters, since it is possible that the project was not correctly categorized. However, when comparing the number of results for each method, it became clear that using filter categories instead of the search box was the best option.

Taking all these particular aspects into account, 114 projects were found on SourceForge in May 2021 by applying the following filters: *Category > Multimedia > Graphics > Editor* and *Freshness > Recently updated*.

3.1.3 GitHub

GitHub is the largest development platform in the world with more than 83 million users and four million organizations (GitHub, 2022). It does not exclusively host open source software, but it is very popular among open source communities.

While there is a search box on GitHub, there are also collections and topics that group similar projects together. Unfortunately, there were not any relevant ones for image editors, which meant that the list of projects would have to be compiled through a search query.

As a starting point, the very general *image editor* query without any filters or operators was performed. This matched thousands of results. While browsing through the list, it became apparent that most of the repositories neither mentioned what language is used for the graphical user interface nor whether the software had been localized into other languages. Many different combinations were tried in order to only get repositories with localized software, for example by adding a qualifier that would only match repositories mentioning *translation* in the readme file or by combining the search query *image editor* with the name of a language (e.g. English, Deutsch, German, etc.). However, it was quickly realised that this would be an impossible undertaking as either the search query would be too restrictive and would not find enough results or it would be too large and would match too many irrelevant repositories. At this point it was decided that monolingual software would also be included in the data set and be analysed as part of this thesis, as it could be interesting to see why some developers chose not to internationalize and localize their programs and whether it really was due to a lack of awareness as explained by Murtaza and Shwan (2012, p. 102). Furthermore, it was also possible that developers of monolingual software had indeed internationalized their program or had at least given it some thought but had not had the opportunity, time or budget to actually localize their software. After the decision to include monolingual software in the data collection was made, the search on SourceForge hence also had to be adapted, as mentioned in section 3.1.2.

Image editors can also be described with other keywords as mentioned in section 3.1.2, and it became apparent that the terminology on GitHub was not consistent, since there was a lot of image editing software described as *graphic editor, graphics editor, graphical editor, photo editor* and *image manipulation*. The main challenge therefore was to narrow

down the list of results by also making sure that this list covered as many image editors as possible, no matter their exact description nor name.

As mentioned before, the search query *image editor* matched thousands of results. It was however found that most of the repositories were irrelevant for this thesis as they were not image editors and simply mentioned the two keywords somewhere within the name or description (e.g. *graphical text editor*). To avoid this problem, quotation marks were used. Now, only repositories that matched the keywords in the same order as they were in the search query were found. Surprisingly, it did not only match repositories with the exact string of characters within quotation marks, as sometimes the description or name of the repository would be slightly different from the keywords. The search *"graphic editor"* would for example also match repositories mentioning *graphical editor* or *graphics editor*.

As previously mentioned, the most common descriptions for image editing software other than *image editor* were *graphic editor, graphics editor, graphical editor, photo editor* and *image manipulation*. Because stemming applies on GitHub, the first three were combined into the search query *"graphic editor".* The queries *"graphic manipulation"* and *"photo manipulation"* were also added for coherence and completeness. The different keywords within quotation marks were then combined into one single search query by using OR operators: *"image editor" OR "graphic editor" OR "photo editor" OR "image manipulation" OR "graphic manipulation" OR "photo manipulation".*

Next, the list of repositories had to be further narrowed down by only including those that were recently updated. Since the search query on SourceForge was conducted on 2 May 2021 and this meant that only projects updated after 1 May 2019 were taken into consideration, the search qualifier *pushed:>=2019-05-01* was added to the base search query on GitHub.

Because the number of repositories was too large considering the scope of this thesis, another search qualifier had to be applied. After some deliberation, the option to filter by number of stars was selected. Admittedly, this is a completely arbitrary way of narrowing down the list and unfortunately, this meant that very small and unpopular projects with zero stars were excluded. It was attempted to minimize this discriminatory effect by keeping the minimum number of stars very low at 10. By adding the search qualifier *stars:>=*10, the list became more manageable in size.

The final search query on GitHub conducted in May 2021 matched 185 repositories and was as follows: *"image editor" OR "graphic editor" OR "photo editor" OR "image manipulation" OR "graphic manipulation" OR "photo manipulation" pushed:>=2019-05-01 stars:>=10*

It should be mentioned here that GitHub support was contacted in the early stages of the data collection in order to help with some of the issues explained above. They explained that unfortunately searching on GitHub was limited and that the best option would be to slice the long query into individual ones and then combine the results for these slices, as a query on GitHub cannot include more than five AND, OR or NOT operators. However, since the base query mentioned a few paragraphs above contained just five OR operators, the decision was made to keep the longer search query and do without slicing on GitHub.

3.1.4 GitLab

GitLab is a platform that follows the open core model, meaning that its core is open source and additional features are released under a proprietary license. They claim to have more than one million active license users and that their platform is used by more than 100,000 organizations (GitLab, 2022).

On GitLab there is a global search service, which searches for projects, issues, merge requests, milestones and users at the same time; plus it is possible to filter the results by group or projects. However, since the objective was to get a list of projects matching certain criteria, the Explore projects page was more relevant.

Similar to GitHub, the first step on GitLab consisted of conducting the general search query *image editor* in order to get an overview of the types of repositories hosted on this platform. Unfortunately, GitLab does not provide the number of results for a particular query, but a manual count showed that there were more than 300 repositories for the search query *image editor* alone. It turned out that one of the most prominent image editors hosted on GitLab was Inkscape. Due to its popularity, hundreds of users had forked this repository, which explained the seemingly endless list of projects for the query *image editor.* For this reason, forks were excluded from the final data collection, as they could have otherwise massively distorted the results. In future studies however, it could be interesting to study these forks as well, since they might have evolved into different directions with different localization approaches.

The next step involved familiarization with the GitLab search function and testing of different logical operators. It was found that similarly to SourceForge, stemming does not apply on GitLab, which meant that the query *"graphical editor"* with quotation marks would only match repositories with this exact character sequence. It was also observed that there were numerous repositories that seemed to be image editing software, but because they used hyphens and underscores or there was not any space between the two words (e.g. graphical-editor, graphical_editor, GraphicalEditor), they did not appear in the list of results for the search query *"graphical editor"*.

In order to keep the results as coherent as possible with GitHub, at least the following six search queries would have to be performed: *"image editor", "graphic editor", "photo editor", "image manipulation", "graphic manipulation"* and *"photo manipulation".* However, while it was possible on GitHub to put these keywords between quotation marks and hereby remove most irrelevant repositories from the list, this was not feasible on GitLab as too many seemingly relevant repositories would have been excluded because of slight variations in naming (hyphens, underscores, spaces, etc.). Therefore, it was decided that quotation marks would not be used, even if that meant that the data collection would include hundreds of repositories that would have to be manually removed at a later stage (see section 3.2).

Because the combination of the base search queries through OR operators did not work, the slicing method was implemented instead, meaning each of the six keyword combinations was conducted as a separate search. They were then all combined manually in a spreadsheet.

The documentation page of GitLab Advanced Search (GitLab, n.d.) provided different search filters, but unfortunately none of them were relevant for this thesis. Reducing the size of the list of repositories arbitrarily by only considering those with a certain number of stars like it was done on GitHub was also not feasible on GitLab, because most of the projects had zero stars.

Another issue was that while it was visible when the project was last updated, there was no possibility to automatically exclude those updated before a certain date (i.e. 1 May 2019, in line with SourceForge and GitHub) by applying a filter. Therefore, when copying the repositories into a spreadsheet, the date of the last update was recorded as

well. The projects that were last updated before 1 May 2019 were then manually removed from the list.

The different search queries with the respective number of results can be found in Table 3 below. This table does not take into consideration the 244 Inkscape forks that were found for the search query *image editor*. Other forks that were less obvious however were removed during the next stage, which will be described in section 3.2.

Search query	**Number of repositories**	**Number of repositories updated after 1 May 2019**
image editor (without Inkscape forks)	134	81
graphic editor	64	37
photo editor	63	41
image manipulation	80	40
graphic manipulation	7	3
photo manipulation	1	0
Total	**349**	**202**

Table 3. Search queries and results on GitLab

3.1.5 Bitbucket

Bitbucket is a platform for source code hosting that is owned by the company Atlassian. It emphasizes the ease of collaboration for teams and offers both free and paid plans with unlimited private repositories for everyone (Bitbucket, 2022).

On Bitbucket, the repository search was not very intuitive. When using the search feature, the results come from the list of own repositories and not from other public repositories. To find new public repositories, the options are to either use Google search or the Explore Page (Bitbucket, n.d.), which has been created for this purpose (Atlassian Community, 2022).

On the Explore page there is a search box and the option to filter by programming language. While a Bitbucket Support page (Bitbucket Support, 2022) provides options to filter the results, unfortunately, none were relevant for the purpose of this thesis.

The same six search queries that were already used on GitHub and GitLab were chosen for better comparability between platforms. According to the official Bitbucket documentation (Bitbucket Support, 2022), quotation marks could be used to find multiple words appearing in a specific order. However, the results of the queries *image editor* and *"image editor"* were exactly the same, which implied that quotation marks did not influence the search query results. Similarly, the same documentation page explains that the combination of keywords with OR operators was possible, but zero repositories were found while testing this method. Therefore, six simple search queries without any search operators or filters were performed and later combined in a spreadsheet like it was done on GitLab. Each repository was then opened manually to check the date of the last update, because there was no way to filter out repositories that were updated before a certain date or to at least sort the results by last updated on Bitbucket. Similar to GitLab, the date was added to the spreadsheet and the repositories whose last update predated 1 May 2019 were manually removed from the list.

Table 4 shows the number of repositories that matched each of the six search queries on Bitbucket.

Search query	**Number of repositories**	**Number of repositories updated after 1 May 2019**
image editor	86	41
graphic editor	59	24
photo editor	30	19
image manipulation	102	33
graphic manipulation	18	9
photo manipulation	3	0
Total	**298**	**126**

Table 4. Search queries and results on Bitbucket

3.1.6 Ad hoc programs

When deciding to limit this research to open source image editing software, it was clear that the software GIMP should be included in the data collection as it is indisputably the most popular in this domain. However, apart from mirrored repositories and forks, GIMP did not appear on any of the four studied platforms. This led to the realization that there could exist popular open source image editing programs that would unintentionally be

excluded from this research because they are hosted on less popular platforms or because they are named or described differently from the keywords used in the search queries. Since one of the objectives of this thesis is to compare open source programs that are very different in terms of size and popularity and to find out whether the internationalization, localization and quality assurance practices change when more resources become available, it had to be ensured that more popular programs would be included in the data collection.

For this purpose, an approach similar to the one used to find source code hosting platforms (see section 3.1.1) was taken. Two Google searches were performed in order to find websites and blog posts with lists of popular open source image editing software:

1. *(image OR graphic OR photo) editor "open source"*
2. *(image OR graphic OR photo) manipulation "open source"*

The first Google search found nine relevant websites and one additional website was found with the second Google search. In total there were 39 programs, however, those that were mentioned only once were not taken into consideration in order to avoid any bias from individual websites. The remaining programs can be seen in Table 5.[6]

[6] The complete table with all of the ad hoc software including on which websites they were mentioned can be found in Appendix B.

Software	Platform	Mentions	Already included in data collection
Gimp	GNOME (GitLab)	10	No
Darktable	GitHub	5	Yes
RawTherapee	GitHub	5	No
Inkscape	GitLab	5	Yes
Digikam	KDE Repository (GitLab)	5	No
Krita	KDE Repository (GitLab)	5	No
Paint.net	Self-hosted	4	No
Photivo	Self-hosted	3	No
Shotwell	GNOME (GitLab)	3	No
LightZone	GitHub	3	Yes
Fotoxx	Kornelix.net	2	Yes
Skencil	Wald.intevation.org	2	No
FontForge	GitHub	2	No
PhotoScape X	Self-hosted	2	No

Table 5. List of ad hoc software

Interestingly, through this method two more relevant GitHub projects were identified (RawTherapee and FontForge), that were not found with the method described in section 3.1.3. Furthermore, four programs were already present in the data collection. The remaining ten programs were then added to those collected from SourceForge, GitHub, GitLab and Bitbucket. Table 6 provides an overview of the number of projects found on each platform.

Platform	Number of projects
SourceForge	114
GitHub	185
GitLab	202
Bitbucket	126
Ad hoc	10
Total	**637**

Table 6. Number of projects on each platform

3.2 Creating the final list of projects

In the initial phase described in section 3.1, a list of 637 image editing projects was compiled. During this phase it became clear that not all repositories within this list matched the predefined criteria established for this thesis, which is why in a second phase, all of the projects were checked manually to see which ones were relevant for this thesis.

One of the criteria was that the programs had to be current and still evolving. This was already addressed in the first stage of data collection as any projects that were last updated before May 2019 had already been removed from the list.

The next criterion was that the project had to be an "image editing software". Because this term is very broad and can encompass different types of programs, for the purpose of this thesis a definition with four distinct elements was produced which made it clear what counts as an image editing software and what does not. It was decided that any software

- of which the main objective it is
- to modify the surface area (not the metadata)
- of a static graphical element (not videos or animations)
- and which offers the possibility to export into standard image formats (TIFF, JPEG, GIF, PNG, etc.)

would be considered an image editor. This is an ad hoc definition and proved to be very effective in helping to decide whether a project should be excluded or not.

The second criterion was that the project had to be open source, and therefore any closed-source software had to be removed from the list. GitHub, GitLab and Bitbucket all host both open source and proprietary projects. Because there was no way to exclude the latter at the stage of data collection, every project on these platforms was opened to check whether there was any information about an open source license. It is assumed that if someone were to use any of these platforms to host their open source software, they would provide the necessary information for it to be reused, which includes the license, otherwise it would defeat the purpose of the open source concept. This was later confirmed through an email exchange with Bitbucket support.

Next, it was important that the image editors were standalone programs. Within the data collection there were many projects that were libraries, add-ins, plug-ins, wrappers, SDKs or modules for image editing software. It lies out of the scope of this thesis to explain each one of these examples in detail, however, it shall be noted that all of these are only components of a more complex software application. The end user would not be able to download these projects and use them without having to add them to an existing program and compiling it with the executable file, which is why they were removed from the list.

The last criterion relates to the graphical user interface (GUI). As explained in section 2.1.2, a translated GUI is one of the most important points of satisfaction to the end user. Some repositories were therefore excluded because they were command line tools and hence no localization had taken place.

Apart from not matching these predefined criteria, there were some other reasons for exclusion. One of them relates only to Bitbucket: in August 2020, all mercurial repositories had been disabled as Bitbucket decided to focus on Git (Chan, 2020). This meant that the mercurial repositories in the data collection were no longer accessible, which is why they were removed from the data set. Furthermore, repositories that were empty and had no files available to download were excluded as well, no matter the platform. A table providing an overview of all the reasons for exclusion explained above alongside a few examples of repositories that were excluded can be found in Appendix C.

After going through the list of 637 projects, the remaining repositories that were relevant for this thesis were combined into one single spreadsheet in order to find any cross-platform forks or doubles. Table 7 illustrates the remaining number of projects per platform after the phase of exclusion and after removing any forks and doubles from the data set.

Source code hosting site	**Number of relevant repositories**
SourceForge	50
GitHub	52
GitLab	30
Bitbucket	6
Ad hoc	6
Total	**144**

Table 7. Data set after removal of irrelevant repositories, forks and doubles

3.2.1 Exclusion of Bitbucket

At this stage it was decided that Bitbucket will not be taken into consideration as a platform for open source image editing software in the context of this thesis.

Firstly, Table 7 shows that out of the initial 126 repositories, only six were actually relevant. Additionally, during the stages of familiarization and data collection on Bitbucket, it became apparent that this platform is not really intended for open source projects. Open source developers need to apply to an Open Source Cloud Subscription Request and their project also needs to meet certain criteria (Atlassian, 2022). During a call with a member of Bitbucket support it was confirmed that Bitbucket does not actually target open source developers but rather is designed for private projects, since a user can host an unlimited number of private repositories on Bitbucket for free. They also pointed out that the only way to contact a developer on Bitbucket is to post a comment on a commit in the repository. According to them, it would be highly unlikely that anyone would respond if the software had not been updated since 2020.

For these reasons, the six projects hosted on Bitbucket were removed from the final list of projects.

3.2.2 Final list of projects

During the stage of manually filtering through the data collection that consisted of 637 projects, any programs that did not match the predefined criteria explained in section 3.2 as well as empty repositories and mercurial repositories were removed. Then, a cross-platform examination was performed to ensure that projects did not appear multiple times within the final list because they were available on more than one of the selected platforms. In a last step, for reasons explained in section 3.2.1, the six remaining projects on Bitbucket were removed from the list.

The final data collection therefore consists of **138 repositories**: 50 from SourceForge, 52 from GitHub, 30 from GitLab, and 6 added ad hoc. The full list can be found in Appendix D.

3.3 Questionnaire

Because the goal of this research is to collect and evaluate quantitative data, the questionnaire was chosen as the preferred data collection method (Saldanha & O'Brien,

2014, p. 152). It was decided that the information about the internationalization, localization and quality assurance practices would solely be obtained through the questionnaire and not through the official pages and repositories of the projects. This was based on the assumption that the majority of projects in the data collection consists of smaller image editors that were likely developed by a single programmer who might not have the time or interest to provide much information on their developing process.

The questionnaire was created on Limesurvey. One of the main advantages of this platform is that the University of Geneva has it installed on its servers which in turn means that the data would be secure and confidential. Additionally, on Limesurvey it was possible to add conditions to questions. This was especially important because the questionnaire targets developers of both monolingual and multilingual software and depending on whether the software is localized or not, the questions would be different.

In the following section, the questionnaire design will be discussed.

3.3.1 Questionnaire design

The questionnaire was designed with the recommendations by Saldanha and O'Brien (2015, pp. 153-163) in mind. Some of the questions are inspired by the questionnaires used in the master's theses by Berthouzoz (2019) and Sanchez Espinoza (2015) as well as by the interview questions of the master's thesis by Murtaza and Shwan (2012). The full questionnaire can be found in Appendix E.

The questions were kept as short as possible. They were mostly multiple-choice so that participants can quickly complete the questionnaire without feeling discouraged or losing interest. However, for most of the questions, an "Other" answering option was added to allow the respondents to provide their own answer. Since not all of the participants might be familiar with the fields of internationalization and localization, explanations and definitions were provided along the way whenever a term was used or it was deemed necessary otherwise.

The questionnaire consists of seven sections:

1. *General questions about the software*
2. *Internationalization*
3. *Localization*

4. *Translators*
5. *Quality assurance and testing*
6. *Internationalization and localization*
7. *Demographic questions*.

Depending on the previously given answers, some questions or even whole sections would be added or hidden, and, in some cases, the default answer options would slightly change. Table 8 shows the different scenarios and the corresponding structure of the questionnaire.

	Software is internationalized and localized	**Software is internationalized but not localized**	**Software is localized but not internationalized**	**Software is neither internationalized nor localized**
General questions about the software	✓	✓	✓	✓
Internationalization	✓	✓		
Localization	✓		✓	
Translators	✓		✓	
Quality assurance and testing	✓	✓	✓	✓
Internationalization and Localization		✓		✓
Demographic questions	✓	✓	✓	✓

Table 8. Structure of questionnaire

Before getting to the questions that are relevant for the research, the participants have to confirm that they are older than 18 for ethical reasons and that their software is open source. If any of these two questions are answered negatively, the participants will be taken to the end of the questionnaire with a note explaining why they unfortunately cannot partake in it.

In the following, the seven main sections of the questionnaire as well as their objective will be explained in more detail.

General questions about the software

The first information that is collected is the name of the software. In order to increase the chances of getting a response, sometimes multiple developers and contributors were contacted for the same software. In the event that the questionnaire was filled out more than once for the same project, thanks to this question any distortion can be avoided by only taking into consideration one set of answers. While the name of the software is not as personal as the name of the participants, in some cases, especially with smaller projects, revealing the name of the software alongside the other results could potentially compromise the participants' anonymity. Therefore, each software will be given a unique code after the participants' answers have been downloaded from Limesurvey, and the version with the names of the projects will be destroyed.

The goal of the next few questions is to collect information that will allow the programs to be grouped into different categories. Questions about the number of collaborators and downloads per week will help determine the scale and popularity of the software, and a question about when the software was last updated can be used to categorize the programs as "older" or "newer". Other information collected includes the operating system or platform the program runs on and the programming language used to develop the software (Sanchez Espinoza, 2015, p. 133). Furthermore, participants are asked about which open source license they chose.

Lastly, at the end of this section the participants are asked whether their software is available in another language, which will determine whether they will be shown the sections *Localization* and *Translators* (see Table 8).

Internationalization

The first question in this segment is "Is your software internationalized?" This will determine whether the rest of the section will be hidden to the participant or not. If the software has been at least partially internationalized, the participants will then be asked about the internationalization process for their software, for example at what stage of the development they started considering internationalization (Berthouzoz, 2019, p. 128) and whether in retrospect, they would reconsider it (Murtaza & Shwan, 2012, p. 114). More technical questions in regard to what tools they used and what aspects they deemed most important are also part of this segment.

Localization

The aim of this section is to get information about the localization process. It can be roughly divided into four subsections:

1. Level of localization: How much and what parts exactly of the software have been localized?
2. Timeline: At what stage was localization considered and when were the localized versions released in relation to the original? (Berthouzoz, 2019, pp. 128-129)
3. Languages: What languages were chosen and why?
4. Tools: Was machine translation integrated into the localization process, if yes, in what way? What other tools were used for localization?

The questions in this segment will help determine how the localization practices in open source differ from those described in section 2.2.

Translators

This section also deals with the topic of localization, however, it focuses specifically on the localizers or translators. As explained in section 2.3.1, literature suggests that volunteering is very common in the open source community. However, following Berthouzoz (2019, pp.128-129), it should not be assumed that this is the only model in open source localization, which is why a question about the recruitment of translators was included. It was also important to find out what the environment and working conditions are for these translators, meaning what resources they are provided, whether there are any specific instructions on localization, what file versions they work with (up to date or not) and how often they interact with the developer.

Quality assurance and testing

The questions in this segment aim to determine whether any quality assurance has taken place in the development cycles of these programs, and if so, what aspects were considered most important.

Some of the questions in this segment vary slightly depending on whether the software is available in another language or not. Developers of multilingual software can choose between three different types of testing ("linguistic testing", "cosmetic testing" and

"functional testing") when asked about the aspects they consider important for a satisfying degree of quality, whereas developers of monolingual software will only see "testing" in a general sense as one of the possible answers.

Similarly, the follow-up question about who was responsible for the testing offers different default answers depending on whether the software has been localized or not. While developers of multilingual software can among other things choose between "translators", "external testers (native speakers of the localized version)" and "external testers (non-native speakers of the localized version)", developers of monolingual software will only see "External testers" as one of the default answers.

Internationalization and Localization

As can be seen in Table 8, this part of the questionnaire is targeted towards developers of monolingual software. The questions in this segment aim to find out why the software has not been localized or even internationalized and whether they would consider these aspects for future projects (Murtaza & Shwan, 2012, p. 114). If the latter question is answered positively, a few hypothetical questions are asked to find out how developers who might not yet be very familiar with internationalization and localization practices would approach this subject.

Demographic questions

In this last section, information about the participants themselves is collected. Most of them relate to open source internationalization and localization, e.g. their mother tongue, how long they have been involved in open source projects, or how they would rate their knowledge of internationalization and localization (Sanchez Espinoza, 2015, pp. 130-131). Two more personal questions about their age and role in the project are included (Sanchez Espinoza, 2015, p. 130; Murtaza & Shwan, 2012, p. 112), however, these should not compromise the participants' anonymity. Lastly, before finishing the questionnaire, the participants can enter their email address if they wish to be informed about the results. Since this is personal information, after downloading the participants' answers from Limesurvey, the email addresses will be deleted from the file and stored separately.

3.3.2 Sending out the questionnaire

The invitations to the questionnaire were mostly sent out by email. Some of the 138 projects in the final data collection provided email addresses of the developers and collaborators on their webpage or on their repository. Developers and contributors of repositories on SourceForge were contacted through the messaging feature that the platform offers if an email address was not available. Other ways of contacting included Facebook, LinkedIn and the contact form on the official website. For the repositories on GitHub and GitLab where no contact information could be found, an issue was created under the repository.

When sending out the invitations, the messages were mostly identical, however, the name of the software was mentioned and therefore adapted as well as the fact whether they were the developer or a contributor to this program. The automatic invitation email feature on Limesurvey was not used because it was assumed that people were less likely to reply to the email and take part in the questionnaire if they realised that it is automatically generated.

The participants were first contacted on 31 May and 1 June 2021. The questionnaire stayed open until 16 June. Nine days later, after checking for which programs there were not yet any answers, a reminder email was sent out. Both the invitation and reminder message can be found in Appendix F.

3.4 Summary

This chapter explained how the data was collected, how the final list of projects was compiled and how the questionnaire was designed. To summarize, at first, four source code hosting platforms that host open source software were chosen: SourceForge, GitHub, GitLab and Bitbucket. By conducting different search queries and applying filters, 627 projects were found. An additional ten projects were added to the list ad hoc to ensure that big and popular image editors that are hosted on other platforms would still be included in the data collection.

Next, the 637 projects were compared to the following pre-set criteria: (i) recently updated, (ii) image editing software, (iii) open source, (iv) standalone program and (v) graphical user interface. If one of the criteria was not met, the software was removed

from the data set. This ultimately led to all projects on Bitbucket being excluded from the list. The final list of open source image editors consists of 138 repositories.

At the end of this chapter, the questionnaire and its design were discussed. It consists of seven sections: (i) *General questions about the software*, (ii) *Internationalization*, (iii) *Localization*, (iv) *Translators*, (v) *Quality assurance and testing*, (vi) *Internationalization and localization*, and (vii) *Demographic questions*. The questionnaire was then sent out to the developers of the programs on 31 May and 1 June 2021.

The results of the questionnaire will be presented and discussed in the next chapter.

4 Results and discussion

While the previous chapter explained how the data was collected and how the questionnaire was designed, now the information gathered through the questionnaire will be presented and discussed. This chapter is divided into four sections. In the first section (§ 4.1), a general overview of the data will be provided. The next section (§ 4.2) is dedicated to internationalization. In section 4.3 results regarding localized software will be discussed, specifically on the localization and quality assurance processes. The fourth and final section (§ 4.4) revolves around monolingual software and the reasons behind not being able to or wanting to localize the program. Quality assurance practices in regard to monolingual software will be elaborated on too.

4.1 Data overview

In this section the general composition of the data collected from the questionnaire will be discussed. After establishing how many participants there were and how many of them claimed that their software was internationalized and/or localized, section 4.1.1 will provide an overview over the information collected about the participants, while section 4.1.2 will touch upon general information about the programs. Some of the results from these two sections will be used later in sections 4.2, 4.3 and 4.4 to draw further conclusions.

Between 31 May and 16 June 2021, there were 96 people who participated in the questionnaire, all of whom claimed to be over 18 years old. However, only 61 respondents' answers were considered for this thesis for the following reasons:

- 28 participants did not finish and submit the questionnaire.
- One participant claimed that their software is not open source.
- Two participants seemed to have only filled in the mandatory questions in the beginning and clicked through the rest of the questionnaire without choosing any answers.
- For four projects, two people each submitted the questionnaire. This became clear because the participants were asked to provide the name of the software. In order to reduce any bias in the results, only one answer per software could be included in the analysis, but unfortunately, in all four cases some questions were answered

differently for the same software. The conflicting answers were analysed and the decision on which set of answers would be excluded was made firstly by comparing the information given by the participants with the information found on the official websites and repositories of the software and secondly by checking the role of the participant in the project (the answers of a project maintainer were considered to be more reliable than those of a translator).

As explained in section 3.2.2, the final data set consists of 138 projects. Since there are 61 unique project answers, the response rate is therefore 44.2%. The developers were contacted either via email, on Facebook, by opening an issue on GitHub or GitLab, via the messaging feature on SourceForge, on LinkedIn or through the contact form on the official website. Table 9 shows the response rates for the respective contact methods.

Contact method	**Number of answers**	**Number of invitations sent**	**Response rate**
Email	40	81	44%
Ticket on GitHub	5	8	62.5%
Ticket on GitLab	0	8	0%
Messaging feature on SourceForge	12	19	63.2%
Facebook	1	3	33.3%
LinkedIn	2	7	28.6%
Contact form	1	2	50%

Table 9. Response rates for each contact method

Furthermore, it was observed that out of the 61 unique projects answers, 31 (50.8%) are hosted on SourceForge, 20 (32.8%) are hosted on GitHub, eight (13.1%) are hosted on GitLab and two (3.3%) were added to the data collection ad hoc. The response rates for each platform are presented in Table 10.

Source code hosting site	**Number of answers**	**Number of projects in the data set**	**Response rate**
SourceForge	31	50	62%
GitHub	20	52	38.5%
GitLab	8	30	26.7%
Ad hoc	2	6	33.3%

Table 10. Response rates for each source code hosting site

Out of the 61 complete and relevant answers, 32 respondents (52.5%) indicated that their software was localized into at least one other language, while according to the other 29 participants (47.5%), their software was monolingual. Five out of the 32 localized programs (15.6%) were said to be partially internationalized, while 27 participants (84.4%) reported they had fully internationalized their program. Thus, it becomes clear that all localized projects seem to have been at least partially internationalized.

Regarding the monolingual software, 24 out of 29 participants (82.8%) said that their project was not internationalized, four participants (13.8%) claimed that it was partially internationalized, and one participant (4%) reported that their software was fully internationalized. It can be observed that a large majority of software that has not been localized has neither been internationalized, implying that these processes are closely linked.

4.1.1 Information about the participants

This section will discuss the information collected from the final section of the questionnaire titled "Demographic questions", namely the participants' age, mother tongue, number of years of experience with open source software, knowledge of internationalization and localization and role in the project.

Age

As can be seen in Table 11. eight people (13.1%) were between 18-24, 11 participants (18%) were between 25-34, 20 participants (32.8%) were between 35-44, 17 participants (27.9%) were between 45-54, two participants (3.3%) were between 55-64, and three participants (4.9%) were over 65 years old. It can be observed that more than half of the participants (60.7%) were between the ages of 35 and 54.

Age	Number of participants	Percentage
18-24	8	13.1%
25-34	11	18%
35-44	20	32.8%
45-54	17	27.9%
55-64	2	3.3%
65 and over	3	4.9%

Table 11. Age of participants

Mother tongues

The mother tongues indicated by the respondents can be seen in Figure 2.

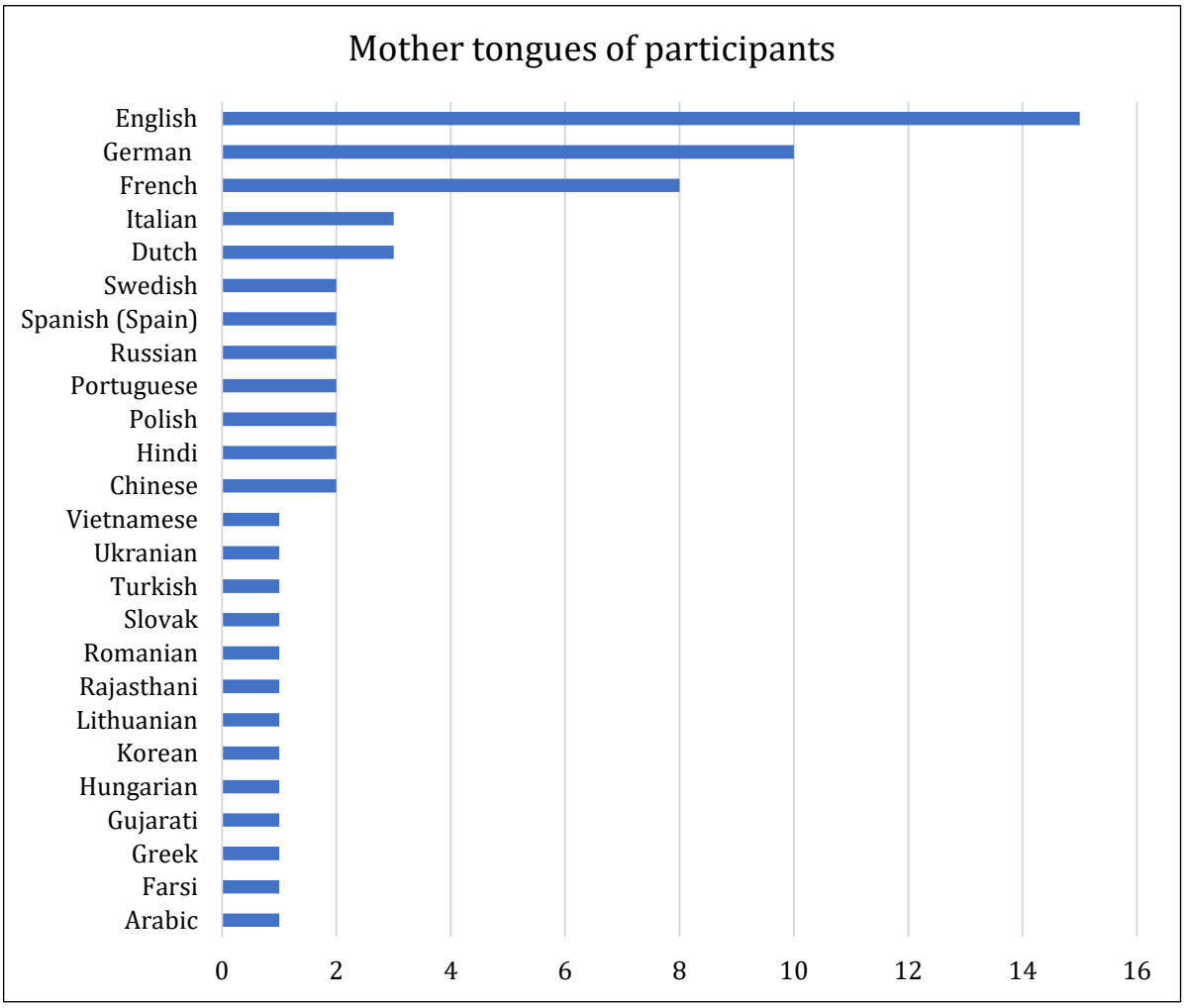


Figure 2. Mother tongues of participants

While there were 25 different languages mentioned in total, English, German and French were by far the most common ones in the data set with 15 (24.6%), ten (16.4%) and eight answers (13.1%) respectively. It should also be mentioned that two participants (3.3%)

wrote down two native languages and that another respondent (1.6%) even claimed to have three mother tongues.

Experience with open source software

The answers to the question "How long have you been involved with open source projects" are presented in Table 12. It can be observed that while only one person (1.6%) each said that they have less than a year experience and that they are not involved in open source projects, 34 participants (55.7%) claimed to have more than ten years of experience within open source.

Experience with open source software	**Number of participants**	**Percentage**
I'm very new to open source projects	0	0%
Less than a year	1	1.6%
1-5 years	10	16.4%
6-10 years	15	24.6%
More than 10 years	34	55.7%
I am not involved in open source projects	1	1.6%

Table 12. Participants' experience with open source software

There is no direct correlation between the number of years of experience and the age of the participant. However, it is interesting to observe that out of the 34 participants that have more than ten years of experience, 28 participants (82.4%) claimed that their age is between 25 and 54 years.

Knowledge of internationalization and localization

As can be seen in Figure 3, for both internationalization and localization, the majority of respondents (36.1% for internationalization and 39.3% for localization) chose the middle option "3" (option 1 representing "I have never heard of it" and option 5 standing for "I am an expert"). When asked about their knowledge about internationalization, one person (1.6%) claimed to have never heard of it, 13 people (21.3%) chose option "2", and 19 participants (31.1%) chose option "4". Regarding localization, three people (4.9%) had never heard of it before, 15 participants (24.6%) rated their knowledge of localization at a level 2 and 14 respondents (23%) chose option "4". Only five people (8.2%) said that

they were an expert in both internationalization and localization. This could also be due to the wording, which led the participants to underestimate their knowledge and experience.

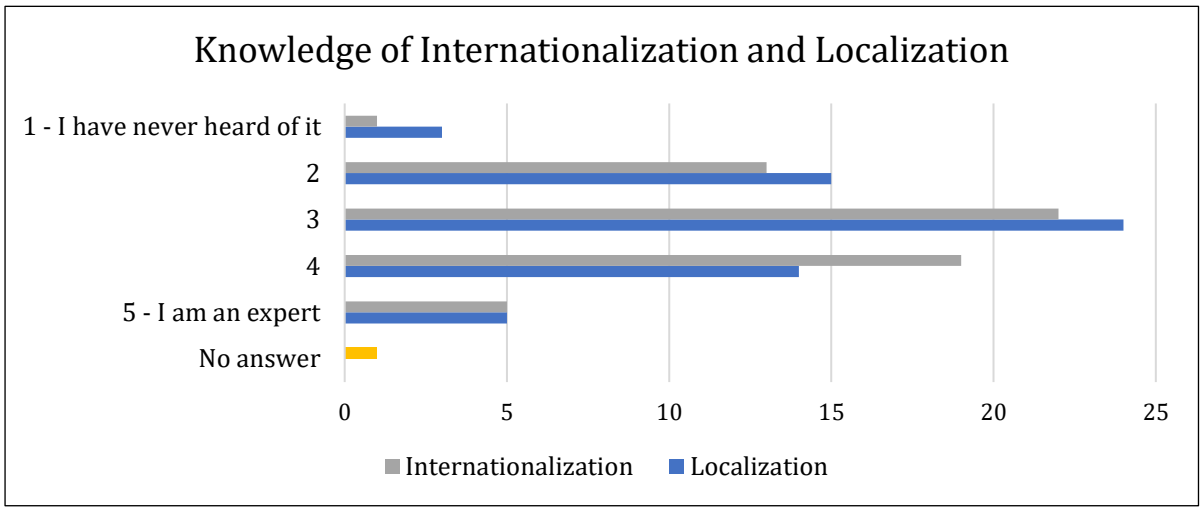


Figure 3. Participants' knowledge of internationalization and localization

Roles of the participants

Regarding the role of the participants in the project, because they could not choose from pre-defined answers and instead had to provide their own, they all varied greatly. A lot of respondents take on multiple roles within the project, such as developer, translator and maintainer. Many participants assume a leading role: there are 29 people (47.5%) whose role was either main or lead developer, head, maintainer, board member, project leader, project creator or author, release manager, or architect. It can also be observed that another 15 participants (24.6%) are the only ones working on their respective projects, which becomes clear because they either said that they were responsible for everything, that they were the sole developer, or they are the owner of the software. Table 13 shows how many different roles and combinations of roles were provided as answers by the participants.

Roles of participants	Number of participants
Sole developer	12
Developer	10
Maintainer	7
Main Developer	5
Head	4
Owner	2
Founder	2
Programmer, designer	1
Everything	1
Former project maintainer, now just a developer and translator	1
Development, bug fixing, release management, board member	1
Project manager and main developer	1
Project Lead, Software Architect, Developer, Tester, Translator	1
Designer, developer and main translator	1
Developer, Release Manager	1
Lead Developer/Product Owner	1
Designer and coder	1
Founder, developer, maintainer and main user	1
Maintainer/owner	1
Project creator + former leader + developer	1
Project Creator/Core Developer	1
Project author, main developer	1
Architect	1
Creator / product manager / designer / developer	1
Mentor	1
Main developer and designer	1

Table 13. Roles of participants in the project

4.1.2 General questions about the software

In the following, questions asked in the section "General questions about the software" of the questionnaire will be further elaborated on. First, the open source licenses selected by the participants will be discussed, followed by the time of the last update and the

operating systems or platforms the software supports. This section then goes into detail about the programming languages used and ends with detailed information about the number of weekly downloads on average and the number of collaborators, which will help determine the sizes of the projects.

Open source licenses

While the participants could choose from a list of more than 100 open source licenses, they only selected 12 in total, which are presented in Figure 4. It can be observed that the GNU General Public License version 3, the GNU General Public License version 2 and the MIT license were by far the most popular with 17 (27.9%), 15 (24.6%) and 14 (23%) mentions respectively.

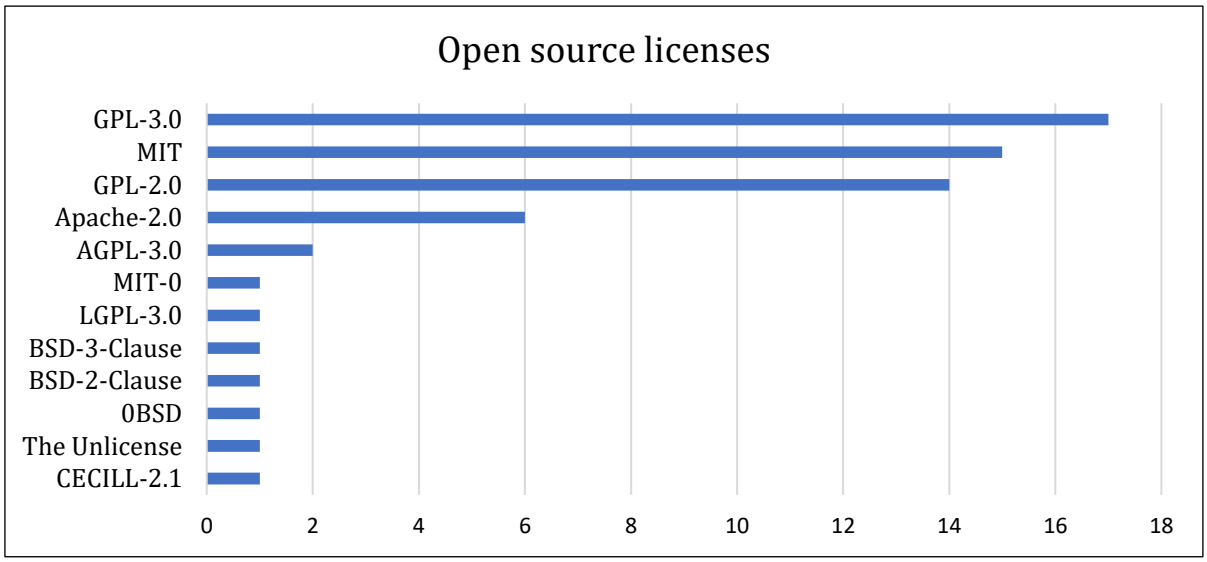


Figure 4. Open source licenses

Both version 3 and version 2 of the GNU General Public License are copyleft licenses. This means that while the user is free to use, change and distribute the original software as well as share their changes, they are obliged to release their work under a license that grants their users these same rights. A software derived from a piece of work under a copyleft license can therefore not be copyrighted. (Smith, 2022)

There are actually no fundamental differences between the two versions. Version 3 attempts to close a few loopholes through which big companies have previously been able to exploit free software. It also guarantees better license compatibility with the Apache license. (Stallman, 2022)

The MIT License in contrast is more permissive than the aforementioned copyleft licenses. The only restrictions are that the "copyright notice and [the] permission notice shall be included in all copies or substantial portions of the Software" (Open Source Initiative, n.d.).

Last update

As can be seen from Figure 5, 27 out of the 61 participants (44.3%) said that their software was updated within the last six months[7]. 15 participants (24.6%) said that their software was updated that month[7] and according to another 12 participants (19.7%) their software had last been updated in 2020. A small minority of four (6.6%) and three participants (4.9%) respectively claimed that their software had last been updated in 2019 or before 2019. This shows that according to the respondents' answers, 88.6% of the programs had been developed and/or maintained within the last 18 months from when the questionnaire was sent out.

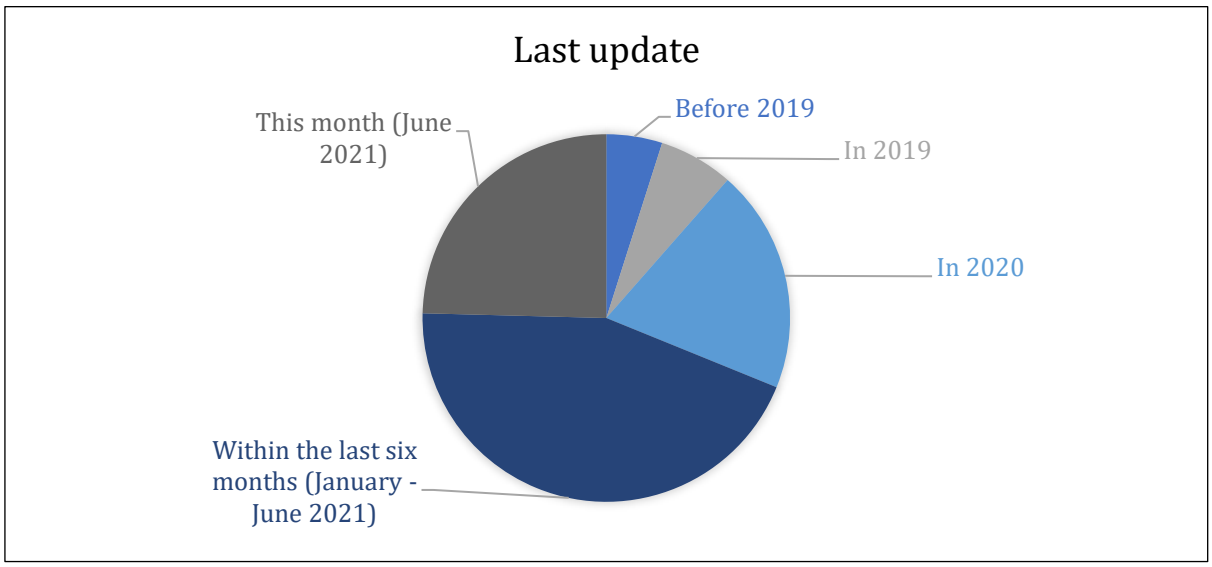


Figure 5. Time of last update

[7] Because the questionnaire was sent out on 31 May and 1 June 2021 and remained open until 16 June 2021, the answering options "Within the last six months" and "This month" roughly designate the timeframe from January to June 2021 and June 2021 respectively.

Platforms and operating systems

From Figure 6 it can be seen that 46 out of the 61 participants (75.4%) indicated that their software runs on Windows, closely followed by Linux which 41 participants (67.2%) chose as an answer. Another popular operating system with 33 mentions (54.1%) is MacOS. Android, iOS, ChromeOS and iPadOS were chosen by 13 (21.3%), ten (16.4%), nine (14.8%) and eight participants (13.1%) respectively. It is probably no coincidence that Linux and Android, which both follow the open source model, were popular among open source developers.

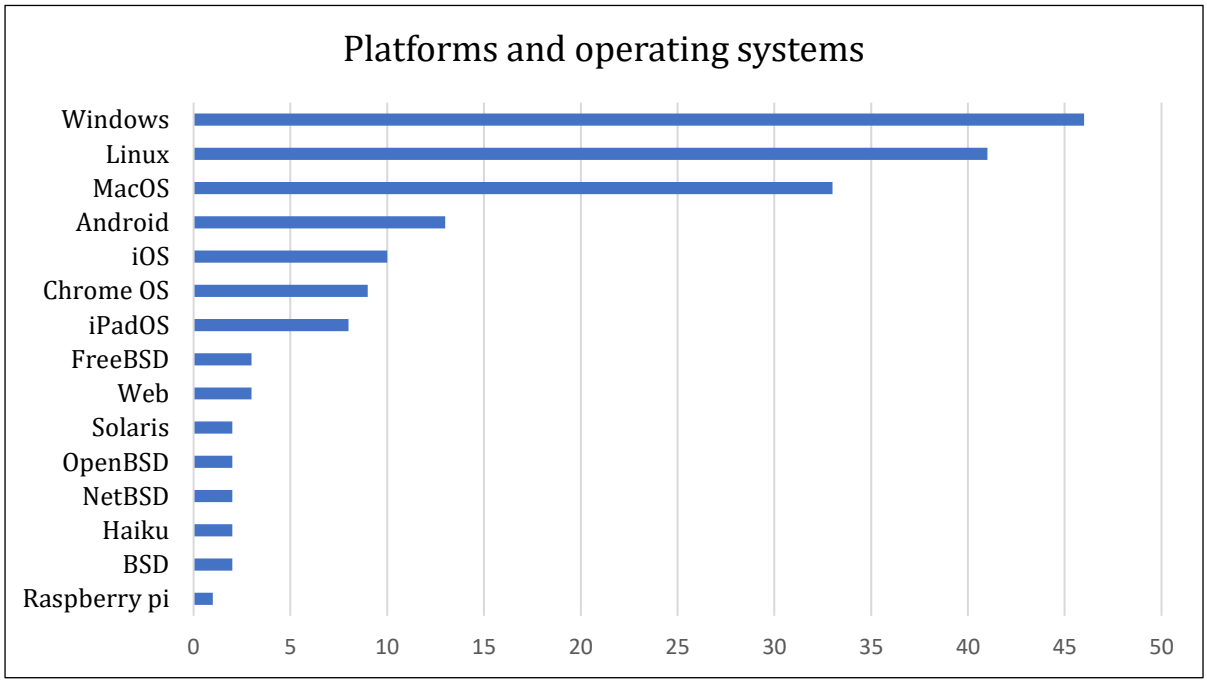


Figure 6. Platforms and operating systems

Apart from choosing from the seven pre-set answers (that were coincidentally the most popular among respondents in total), the participants were also able to include additional operating systems and platforms under the option "other". A few respondents indicated that their software runs on the descendants of the discontinued operating system Berkeley Software Distribution (BSD), namely FreeBSD with three mentions (4.9%), OpenBSD with two mentions (3.3%) and NetBSD with two mentions (3.3%). Another two participants reported that their software supports BSD, however it is unlikely that they meant the discontinued operating system itself, but rather used it as a blanket term for

all its descendants. BSD as well as FreeBSD, OpenBSD and NetBSD are free and open source operating systems[8].

Three out of the 61 participants (4.9%) gave the answer "Web", indicating that their program is a browser-based application and therefore runs independently of the operating system. Furthermore, Solaris (now called Oracle Solaris[9]) and the open source operating system Haiku were mentioned by two participants (3.3%) each. Lastly, one person (1.6%) wrote Raspberry Pi, which in itself is not an operating system but rather a small computer developed to teach people the basics of computing and programming (Raspberry Pi Foundation, n.d.). However, the participant might have meant "Raspberry Pi OS", which is the official supported operating system of the Raspberry Pi computers.

As can be observed from Figure 6, according to the participants' answers, open source image editors mostly support desktop based operating systems such as Windows, Linux, MacOS and Chrome OS. The support for mobile operating systems like Android, iOS and iPadOS seems to be less common. In other words, only 16 out of 61 participants (26.2%) indicated that their software offers support for at least one mobile operating system, while 54 participants (88.5%) claimed their software is available for at least one desktop operating system.

Another interesting observation is that 26 programs (42.6%) are said to only run on one operating system according to the participants' answers. In comparison, only 15 participants (24.6%) claim to support between four and eight operating systems. The detailed distribution can be seen in Table 14.

[8] For more information on the descendants of BSD, please consult the official websites of the projects: https://www.freebsd.org/ (accessed: 3 August 2022), https://www.openbsd.org/ (accessed: 3 August 2022) and https://www.netbsd.org/ (accessed: 3 August 2022)

[9] For more information on Oracle Solaris, please consult their official website: https://www.oracle.com/ (accessed: 3 August 2022)

	Number of participants	Percentage
Available on 1 operating system	26	42.6%
Available on 2 operating systems	4	6.6%
Available on 3 operating systems	16	26.2%
Available on 4 operating systems	3	4.9%
Available on 5 operating systems	2	3.4%
Available on 6 operating systems	0	0%
Available on 7 operating systems	9	14.8%
Available on 8 operating systems	1	1.6%

Table 14. Number of operating systems

Programming languages

The most popular programming language of the data set is Java, with 14 out of 61 participants (23%) claiming they used it. It is followed by C++ with 11 mentions (18%), Python with ten mentions (16.4%), C with seven mentions (11.5%) and JavaScript with six mentions (9.8%). The remaining programming languages selected by the respondents can be found in Figure 7.

Interestingly, 47 participants (77%) claim to have only used one programming language for their software. Ten participants (16.4%) reportedly used two programming languages, while two people (3.3%) said that they used three languages. Lastly, one participant (1.6%) each indicated that they used four and six programming languages respectively (Table 15).

	Number of participants	Percentage
One programming language	47	77%
Two programming languages	10	16.4%
Three programming languages	2	3.3%
Four programming languages	1	1.6%
Six programming languages	1	1.6%

Table 15. Number of programming languages

When looking at the 15 programs that supposedly were developed using more than one programming language, it can be observed that C++ was used in six programs (40%).

Python and Java were also combined with other programming languages in five (31.3%) and three (20%) projects respectively.

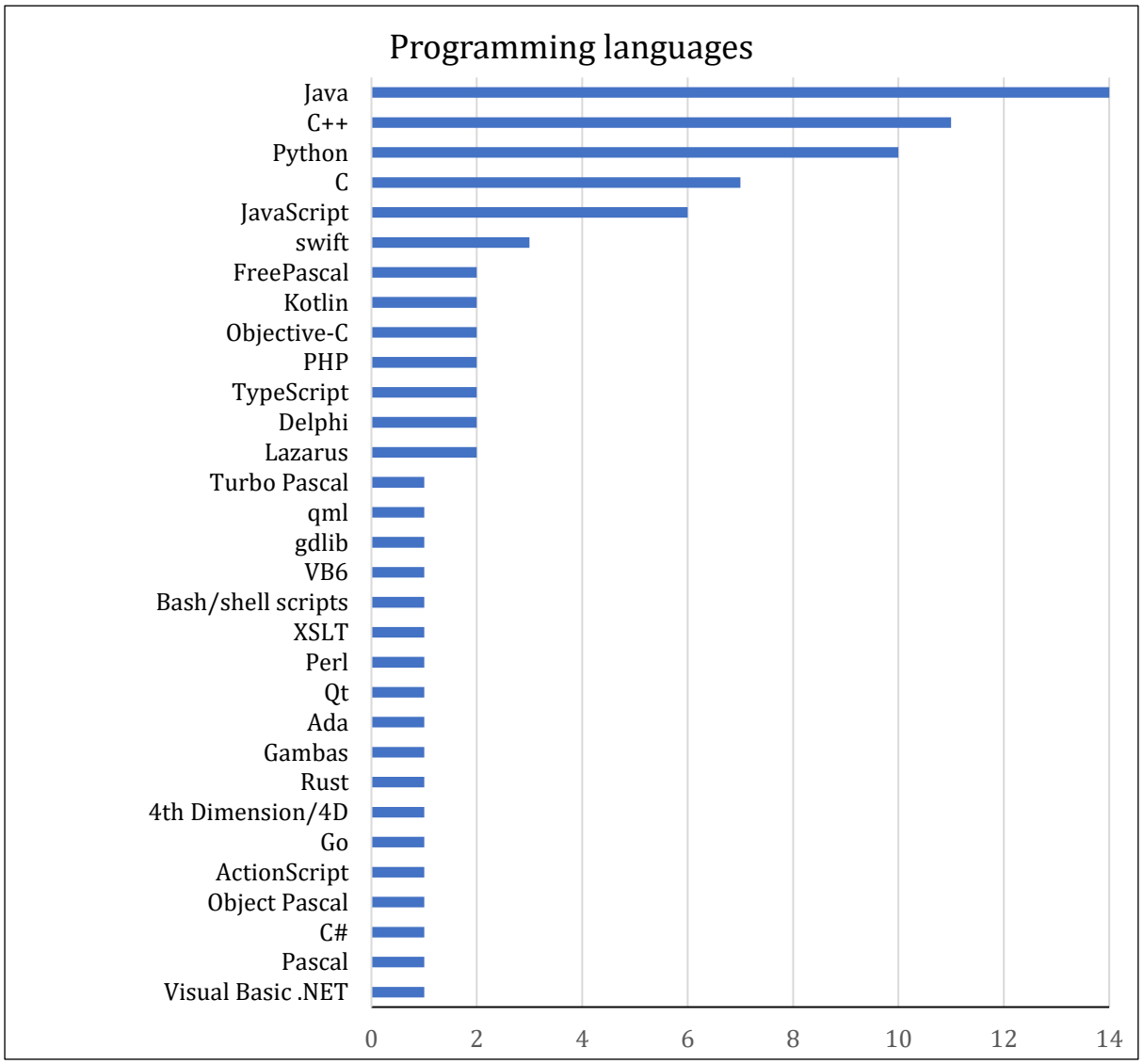


Figure 7. Programming languages

Upon further research it was realised that some of the so-called programming languages that were mentioned by the participants in the comment section are not technically considered programming languages. Qt for example is a framework written in C++ and serves to extend the features of the C++ language (Qt Wiki, 2022). Similarly, one participant said they used “gdlib”, referring to the GD Graphics Library, which is, as the name suggests, a library and not a programming language. Next, Lazarus is a cross-platform integrated development environment (IDE), which enables developers to work more efficiently as many aspects of programming such as editing source code and debugging are combined into one platform (Codecadamy, 2022). Turbo Pascal is a

compiler and IDE combined for the Pascal language (SourceForge, 2022b), and lastly, FreePascal is technically a compiler for the languages Pascal and Object Pascal (Free Pascal team, 2019).

The fact that some of the answers provided by the respondents did not really denominate a programming language was disregarded when creating Figure 7. Manually removing participants' answers and arbitrarily changing the results was not considered necessary as these answers were still related to programming languages and provided valuable insight over the developing process.

Downloads per week on average

When asked how many times their software was downloaded per week on average, as can be seen in Table 16, 19 people (31.1%) responded with less than ten times. Nine participants (14.8%) said that they had between 11-50 downloads per week on average, five people (8.2%) each claimed it was between 51-100 and 101-500 downloads respectively, and one participant (1.6%) each responded with 501-1000 and 1001-10,000 weekly downloads. According to five participants (8.2%), their software is downloaded more than 10,000 times per week on average. 16 respondents (26.2%) unfortunately did not know the answer to this question.

Weekly downloads on average	Number of participants	Percentage
Less than 10	19	31.1%
11 to 50	9	14.8%
51 to 100	5	8.2%
101-500	5	8.2%
501-1000	1	1.6%
1001-10,000	1	1.6%
More than 10,000	5	8.2%
I don't know	16	26.2%

Table 16. Downloads per week on average

When only taking into consideration the 45 programs for which the participants knew how many weekly downloads there were, 73.3% are downloaded less than 100 times per week on average. Only 13.3% had more than 1000 weekly downloads.

Collaborators

Regarding the number of collaborators on the projects, Table 17 shows that 50 respondents (82%) said that their team consists of less than ten people. Eight participants (13.1%) chose the answer 11-30, and two participants (3.3%) claimed that there are more than 50 collaborators. One participant (1.6%) did not provide any answer.

Number of collaborators	Number of participants	Percentage
Less than 10	50	82%
11 to 30	8	13.1%
31 to 50	0	0%
More than 50	2	3.3%
No answer	1	1.6%

Table 17. Number of collaborators

There seems to be a correlation between the number of collaborators and the number of downloads per week on average (presented in the previous section). When disregarding the 16 participants who did not know the number of weekly downloads, Figure 8 shows that participants who indicated that they collaborate with less than ten people are more likely to also claim that their software has less than ten downloads per week on average. For software that has more than 50 collaborators, the correlation is weaker, but still present. However, because in total, the percentage of respondents who did not know the number of weekly downloads is considerably high (26.2%), such deductions have to be made with caution.

The reason behind this possible correlation could be that software with a small number of weekly downloads is not as popular among users, which in turn means that there is little to no appeal for people to contribute to them.

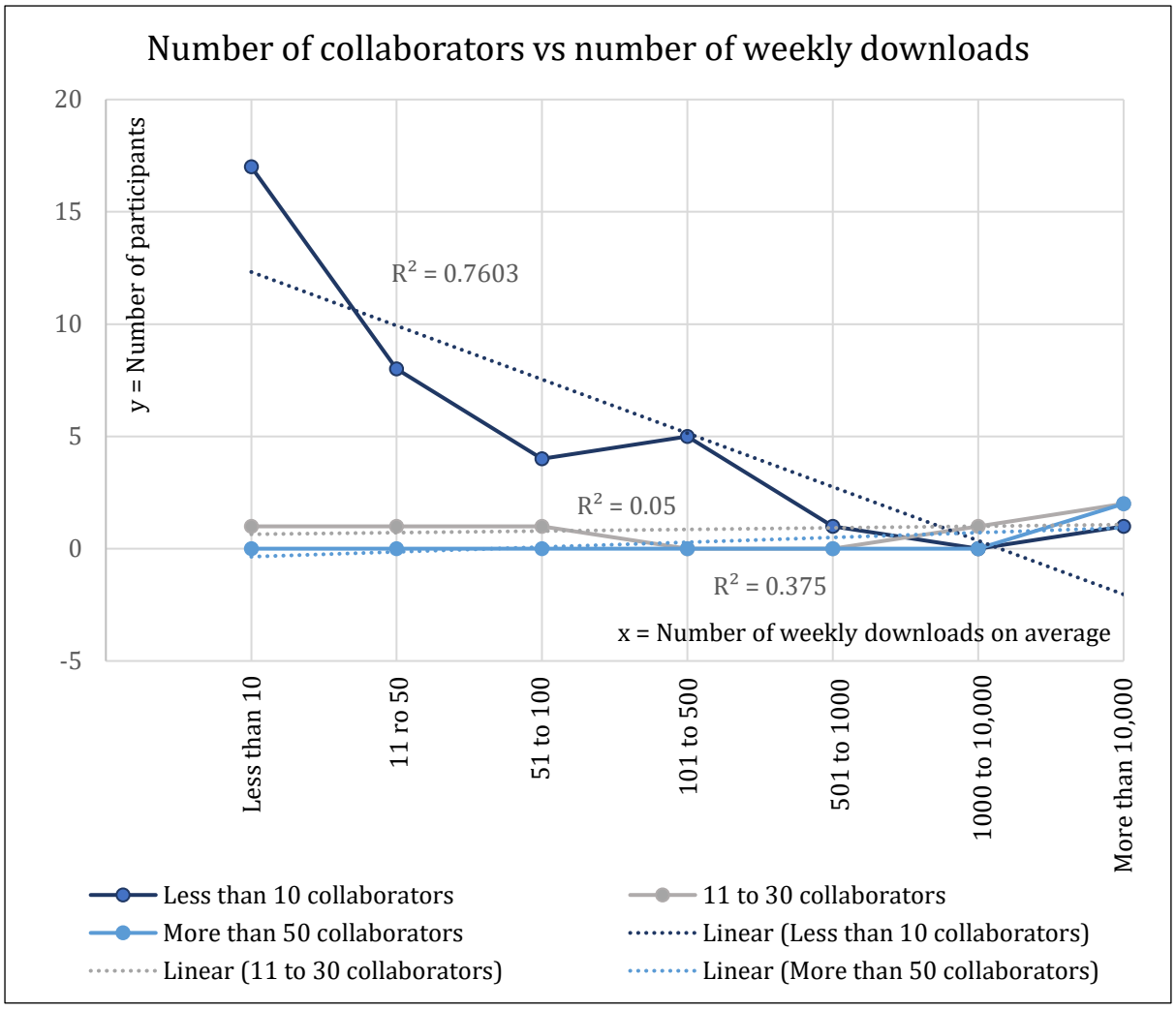


Figure 8. Number of weekly downloads vs number of collaborators

For the remainder of chapter 4, a program with more than 50 collaborators and/or more than 1000 weekly downloads on average will be considered a "bigger" software, whereas a project with less than ten collaborators and/or less than 100 weekly downloads will count as a "smaller" software. Evidently, this is an arbitrary definition of project size, however, it will be helpful when comparing the different results in terms of size and popularity of the program.

4.2 Internationalization

This section will discuss the results regarding internationalization. It is divided into the following subsections: level of internationalization (§ 4.2.1), stage of consideration of internationalization (§ 4.2.2), important aspects of internationalization (§ 4.2.3) and lastly internationalization tools (§ 4.2.4).

4.2.1 Level of internationalization

More than half of the participants (37 out of 61, 60.7%) said that their software is internationalized. Out of these 37, 28 respondents (75.7%) claimed that their program is fully internationalized, while the other nine participants (24.3%) answered that it is only partially internationalized. Regarding the level of internationalization and the size of the project, there does not seem to be a correlation.

When asked whether their software contains any hardcoded text, 19 out of 37 people (51.4%) responded positively, 13 (35.1%) responded negatively and five (13.5%) did not give any answer at all. It is interesting to note that out of the 19 participants that said their software contains hardcoded text, 15 participants (78.9%) also claimed earlier on in the questionnaire that their software was fully internationalized. This could imply that they were not aware of the best practice in this field discussed in section 2.1.2, namely that any hard coded text should be removed from the source code. Meanwhile, the other four participants (21.1%) that said that their software contained hardcoded text also reported that they had only partially internationalized the program, which leaves room for the removal of hardcoded text at a later stage.

4.2.2 Stage of consideration of internationalization

Literature suggests that the earlier internationalization is considered in the development process, the easier and more cost-effective its implementation will be (see section 2.1.4). Figure 9 shows at what stage the 37 participants first thought about internationalization.

While only seven participants (18.9%) considered it before development, 14 participants (37.8%) thought about internationalization in the early stages of development. Six people (16.2%) answered with “Near the end of development”, while ten participants (27%) chose the answer “After completion or release in its first language”. It therefore becomes clear that the percentage of people who considered internationalization early on (56.7%) is higher than the percentage of people who took internationalization into account much

later (39.2%). However, it can be observed that the difference between the two is not striking.

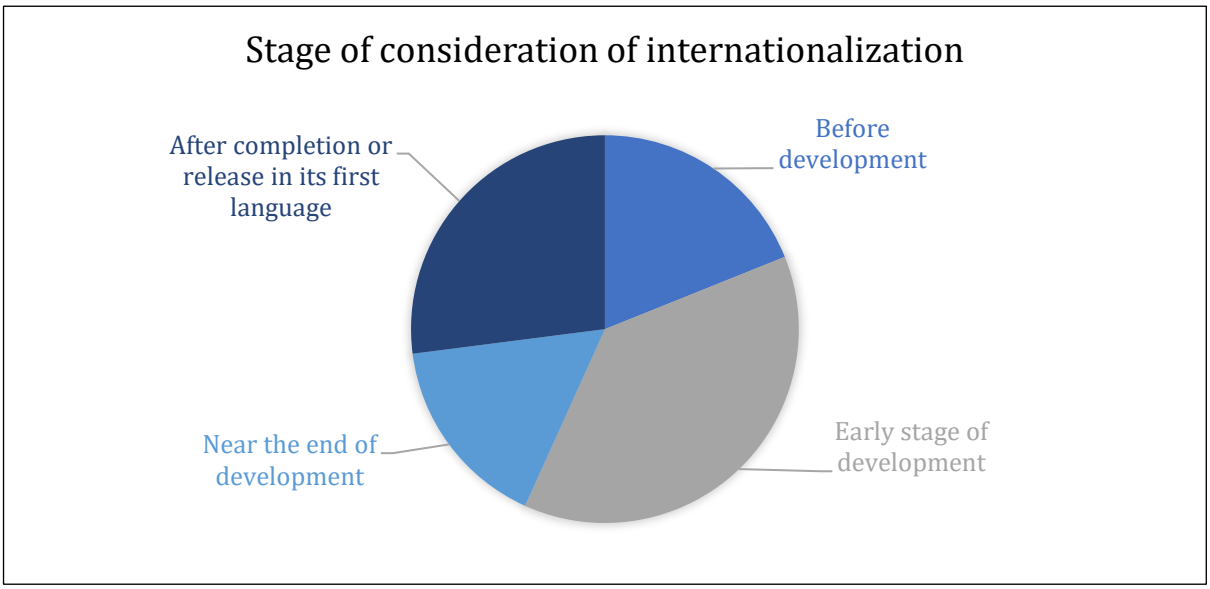


Figure 9. Stage of consideration of internationalization

4.2.3 Important aspects of internationalization

Figure 10 shows what aspects the 37 respondents considered important when internationalizing their software. Besides choosing from a few pre-set answers, they were also able to provide their own in a comment box. It can be seen that the most commonly chosen answers were accommodation for text string expansion with 25 mentions (67.6%), character encoding with 16 mentions (43.2%), number formats with 11 mentions (29.7%) and page layout with nine mentions (24.3%). Only five respondents (13.5%) considered date and time formats to be an important aspect of internationalization, which could be due to the nature of image editing software and the fact that date and time functionalities are not necessarily important for this type of program. The reason why text directionality and progress bar directionality were not a priority for more than four participants (10.8%) and one participant (2.7%) respectively could be because they did not (plan to) localize their software into a language with a different text directionality compared to the original version. Similarly, only one person (2.7%) considered the calendar system as an important aspect of internationalization, probably because for the other participants this issue did not arise due to the chosen target market.

Two respondents (5.4%) also mentioned that they did not have to prioritize any of these as the framework they used already took care of all the aspects mentioned. Another two participants (5.4%) mentioned ease of translation as an aspect they considered important, which is arguably very broad and vague. One participant (2.7%) explained that they wanted to avoid the issue of consolidation of text strings, which is when a text string in English means to two different things and both meanings are saved in the same file, but these have to be translated into two different strings in another language (Behrens, 2016, p. 98). Lastly, one person (2.7%) mentioned automatic tests as an important aspect of internationalization, even though this step usually occurs at the stage of quality assurance.

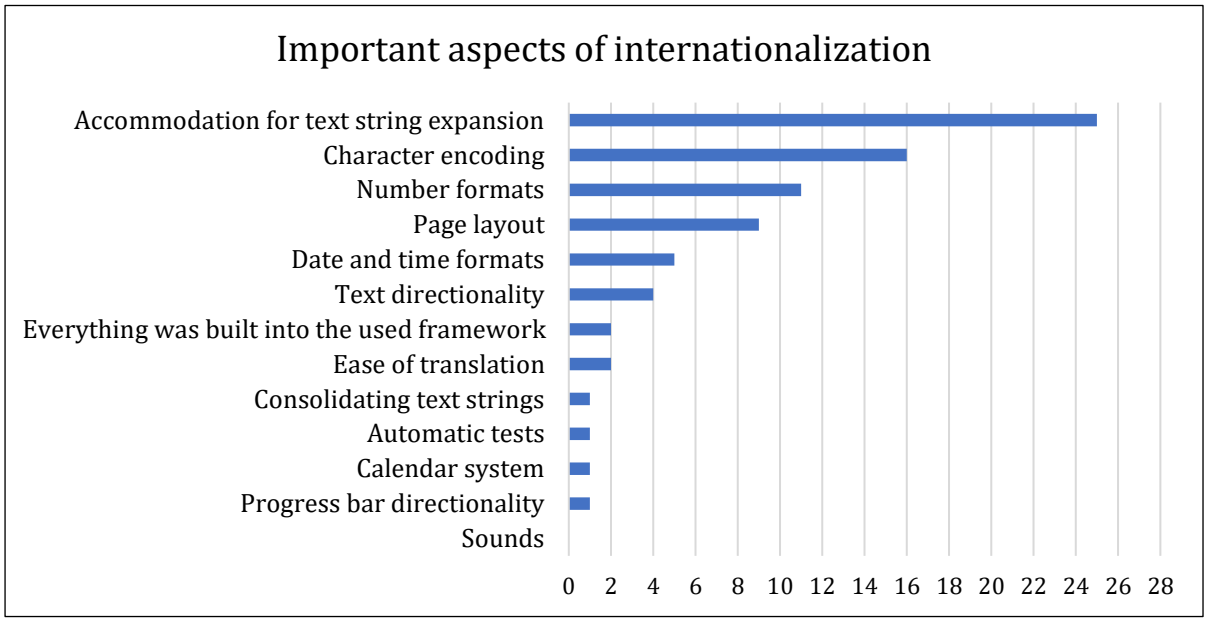


Figure 10. Important aspects of internationalization

4.2.4 Internationalization tools

As was the case for the question discussed in the previous section, when it came to internationalization tools, the participants were given a few pre-set answers but could also provide their own. Because the question "What tool(s) did you use in the internationalization process?" was very broad, there were a wide range of different answers, which are presented in Figure 11. It can be observed that 13 participants (35.1%) said to have used Gettext during the internationalization stage, which makes it the most popular internationalization tool in this data set. This is in line with Frimannsson and Hogan's (2005, p. 1) claim that Gettext is used in the majority of open source software.

The website Poeditor.com was mentioned by three people (8.1%), however, it should be said that it is actually a localization management platform (POEditor, 2022). Google Translate was brought up by two respondents (5.4%), even though it is technically not a tool for internationalization but rather a machine translation engine and is therefore usually used during the localization stage. The tools Encoding converter and Unicode code converter that were mentioned in section 2.1.3 of the literature review were selected by two people (5.4%) each. All the other answers provided by the participants were each only mentioned by one person (2.7%) and can be seen in Figure 11.

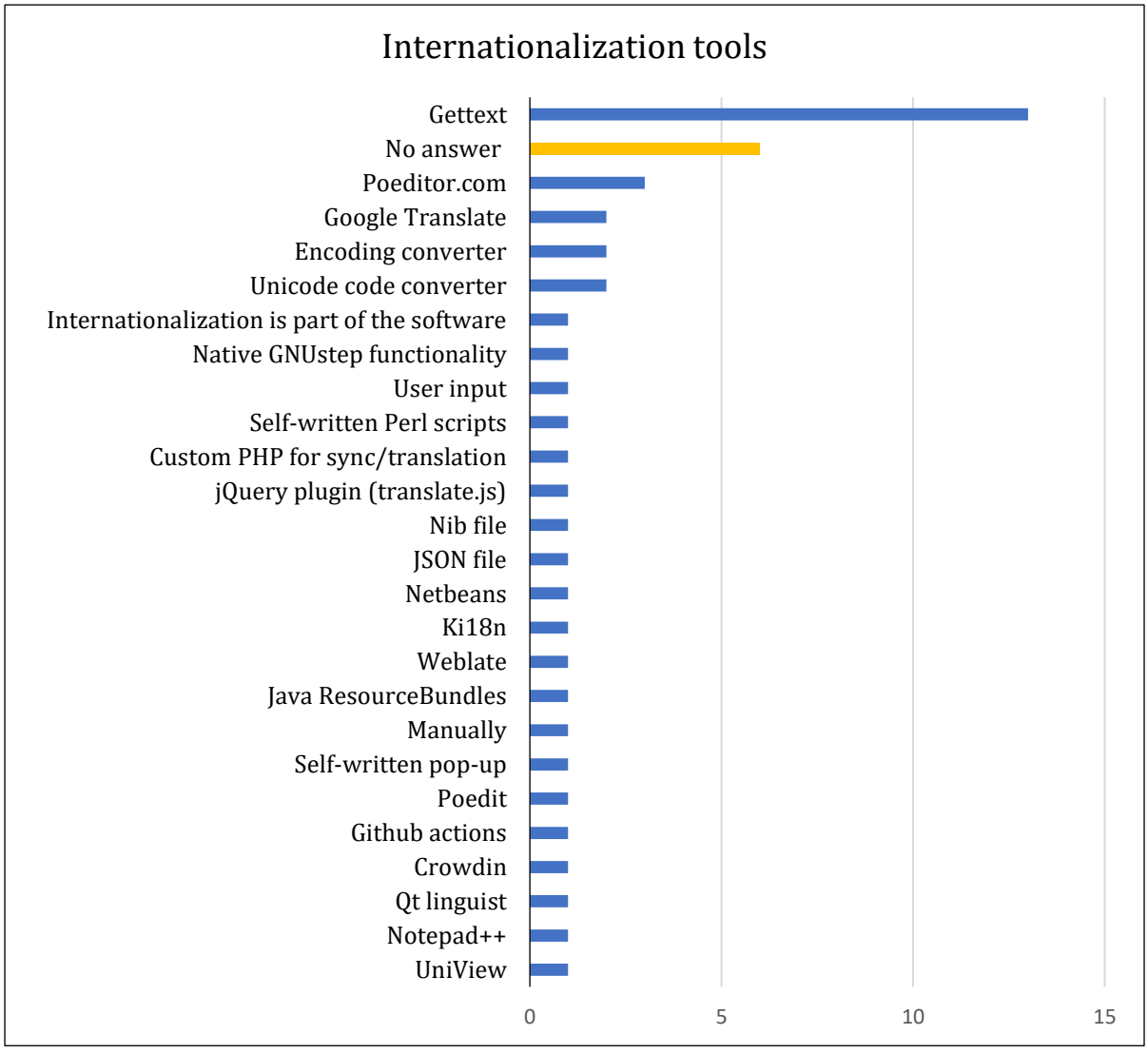


Figure 11. Internationalization tools

It could be argued that at least seven people (18.9%) wrote their own code to potentially simplify the internationalization process, namely those that provided the answers "Self-written Perl scripts", "Custom PHP for sync/translation", "Nib file", "JSON file", "Java

ResourceBundles", "Manually", and "Self-written pop-up". Lastly, six people (16.2%) neither selected any of the pre-set answers, nor did they provide any of their own.

Before turning to the localization results, it should be said that while 36 participants (97.3%) would consider internationalization again if they were to redo the project, one participant (2.7%) said they would not. Their reasoning was that "it felt like an easy after thought, [but] not a whole lot of contributors [were] interested".

4.3 Localized software

This section will present information that specifically relates to software that has been localized into at least one target language. While subsection 4.3.1 will talk about the process of localization and the people involved, subsection 4.3.2 goes into detail about what the developers and collaborators of localized software have said about quality assurance.

4.3.1 Localization

In the following, the results collected from the sections "Localization" and "Translators" of the questionnaire will be discussed. The first three subsections will be about the level of localization, the stage at which localization was first considered and when the localized software was released in relation to the original version. Then, it will be explained which target languages were chosen by the participants and why. The next section will talk about the tools used during the localization process, which will be followed by a section about machine translation. At the end of the localization section the translators, details about their profile, the file formats they use, the resources they are provided and the frequency at which they are interacted with will be discussed.

Level of localization

When asked whether their software was available in more than one language, 32 out of 61 participants (52.5%) answered positively. However, only 15 out of these 32 respondents (46.9%) said that the software is fully localized. As can be expected, all five participants who indicated that their software has more than 10,000 weekly downloads on average also reported that their software was fully localized.

When the other 17 participants were asked which parts they have already localized, everyone (100%) indicated that the user interface has been translated. Two people (11.8%) also checked the answer "Help files & documentation", while one of these people (5.9%) additionally claimed that the website has been localized. It therefore remains unclear what part of the software is only available in one language, if the user interface, the help files and the documentation as well as the website have been translated. Furthermore, the two programs of which the help files and documentation were also localized according to the participants' answers are not bigger projects as one would expect. Rather, one respondent indicated that their software had 51-100 weekly downloads and 10-30 collaborators while the other respondent did not know the number of weekly downloads but claimed to work with less than ten other collaborators.

Finally, eight out of 17 respondents (47.1%) seem to be satisfied with the degree of localization, while the other nine participants (52.9%) are continuing to localize parts of their software.

Stage of consideration of localization

As can be seen in Figure 12, six out of 32 participants (18.8%) claimed that they first thought about localization before the development of the software, while 12 participants (37.5%) said it was during the early stage of development. In comparison, only five people (15.6%) chose the answer "Near the end of development" and nine participants (28.1%) responded that it was only considered after the software was released in its first language. The percentage of participants who thought about localization in the beginning stages of development (56.3%) is therefore slightly higher than the percentage of people considering it later on (43.7%).

Interestingly, when comparing the results of this questions to those from section 4.2.2, the correlation coefficient is 0.9972, meaning the percentages are nearly identical, which can be seen in Figure 13. This implies that developers of multilingual software believe internationalization and localization to be closely linked and that they most likely consider both at the same time.

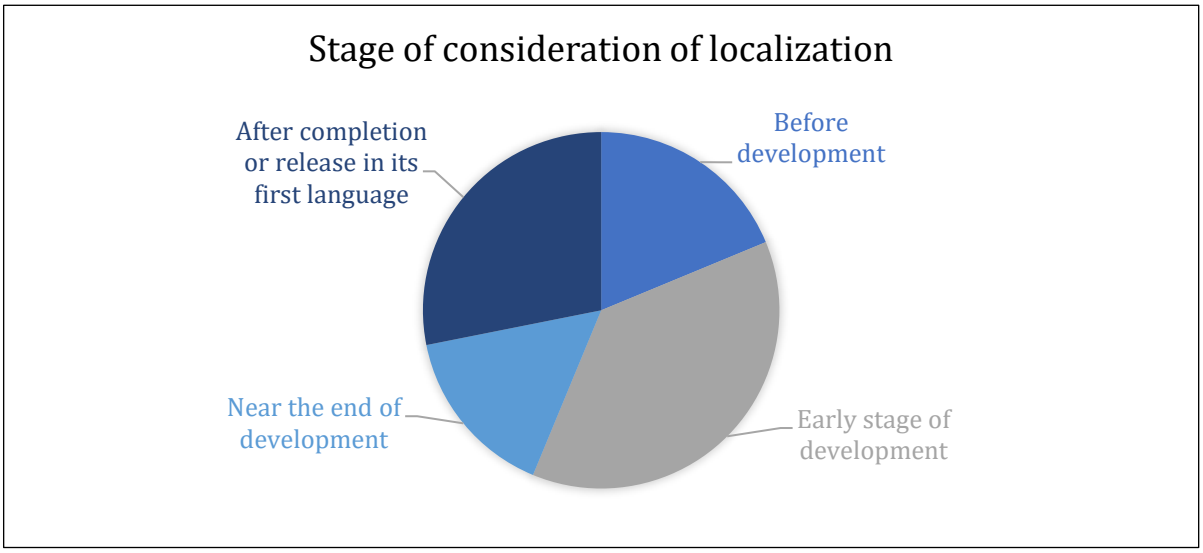


Figure 12. Stage of consideration of localization

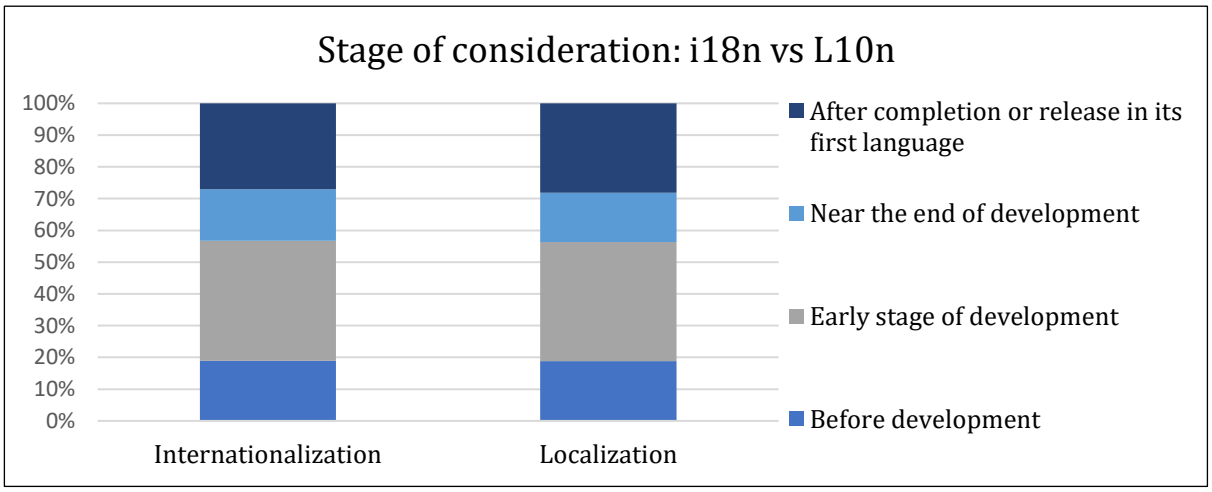


Figure 13. Stage of consideration: internationalization vs localization

Release of localized version

When asked when the localized version was released in relation to the original, 16 people (50%) claimed that it was released after the original version, while 15 people (46.9%) said that that it was released simultaneously with the original software. One person (3.1%) responded that their software had actually not yet been released. This shows that it is pretty much an equal split when it comes to the question whether localization was an afterthought or whether it was planned and taken into consideration early on.

Target languages

The 32 people whose software is available in at least one other language provided 147 different answers when asked what languages their software has been localized into. The top 25 target languages are presented in Table 18, while the list with all 147 languages can be found in Appendix G. The eleven most common were French and German with 21 mentions each (65.6%), closely followed by Spanish with 18 (56.3%), Chinese (simplified) and Japanese both with 16 (50%), Italian with 15 (46.9%), Dutch with 14 (43.8%), Russian with 13 (40.6%), and English, Portuguese and Swedish with 12 mentions (37.5%) each. Unsurprisingly, French, Italian, German and Spanish all figure in the list of the ten most common target languages. These languages are also known under the abbreviation of "FIGS", which has represented the biggest markets for localization for a long time (Schäler, 2010, p. 209). As can be seen in Table 18, 19 out of the top 25 (76%) most common target languages are languages spoken in Europe, out of which 11 (57.9%) in Western Europe. Chinese and Japanese seem to be important markets as well since they both figure in third place.

While in a commercial setting the target languages are chosen mostly based on which markets are most lucrative, this is not the case for open source software as most of them are offered to the user free of charge. The participants could select multiple reasons as to why they chose these particular languages and could also provide their own answers.
16 out of 32 participants (50%) responded that native speakers of the respective language volunteered and provided the translations and that it therefore was not an active choice. This could also explain the immense variety of languages in Appendix G, as traditionally, most of them would neither be considered popular nor lucrative. 14 participants (43.8%) indicated they had a personal connection to the language(s), while eight respondents (32%) wanted to reach a bigger audience and therefore chose more wide-spread languages that are spoken by a large group of people. One participant (3.1%) responded that they speak the target language, which implies that they localized the software themselves. Another reasoning provided by a participant (3.1%) was that the language they chose was a starting point for other supported languages, while someone else (3.1%) claimed that the target language was chosen based on a user's request.

Language	Number of participants
French	21
German	21
Spanish	18
Chinese China (simplified)	16
Japanese	16
Italian	15
Dutch	14
Russian	13
English	12
Portuguese	12
Swedish	12
Czech	11
Turkish	11
Catalan, Valencian	10
Polish	10
Portuguese Brazil	10
Greek modern	9
Hungarian	9
Serbian Serbia	9
Arabic	8
English UK	8
Finnish	8
Galician	8
Hebrew	8
Twi	**8**

Table 18. Target languages (top 25)

An interesting observation that can be made is that only two programs (6.3%) have been localized into a single target language. Ten participants (31.3%) indicated that their software is available in two to five other languages, while four participants (12.5%) each reported that there are 6-10 and 11-20 target languages respectively. Two people (6.3%) did not mention any language. Lastly, ten people (31.3%) reported that their software has been localized into more than 20 languages. (Table 19)

	Number of participants	Percentage
One target language	2	6.3%
2-5 target languages	10	31.3%
6-10 target languages	4	12.5%
11-20 target languages	4	12.5%
More than 20 target languages	10	31.3%
No answer	2	6.3%

Table 19. Number of target languages

When analysing the programs with 11-20 and more than 20 target languages, there does not seem to be a clear correlation between bigger projects having more localized versions. As can be seen from Figure 14, the R-squared values for smaller, medium-sized and bigger projects are all below 0.4. It can be observed however, that all five projects that have more than 10,000 weekly downloads on average have many target languages, one participant (20%) reporting 11-20 languages, while the other four respondents (80%) claim to have localized their software into more than 20 languages.

Finally, when asked whether they are planning to add more languages in the future, 21 participants (65.6%) answered positively, nine participants (28.1%) did not consider it necessary and two people (6.3%) did not answer at all.

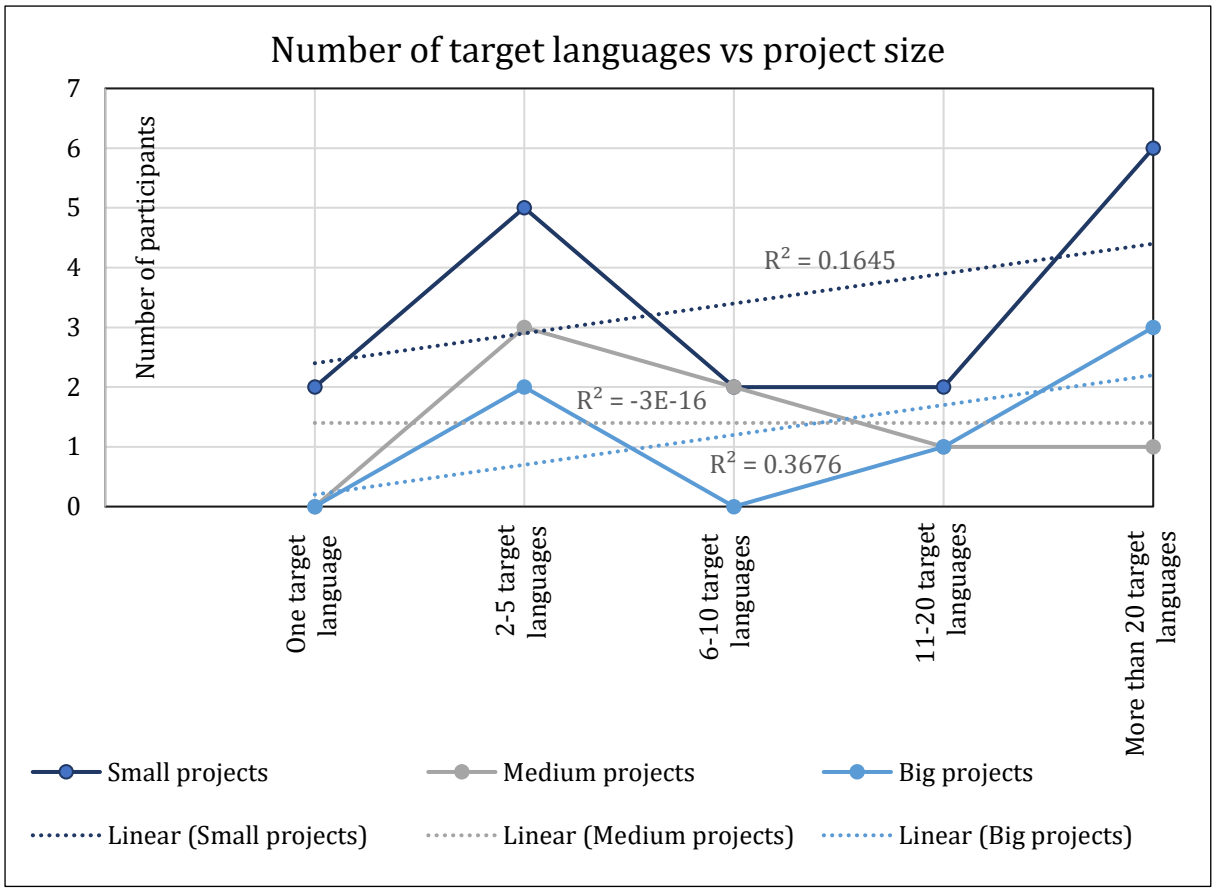


Figure 14. Number of target languages vs projects size

Localization tools

The answers to the question what tools were used during the localization phase are presented in Figure 15. Five out of 32 participants (15.6%) mentioned that they used the localization management platform POEditor. Another three respondents (9.4%) indicated that they used Launchpad.net, which is a software collaboration platform (Launchpad, 2022). Two participants (6.3%) said they used Poedit, which is a translation editor that was built to support the Gettext PO format (Poedit, 2022). Three participants (9.4%) explained that they did not use any specific localization tools and instead left that up to the translators. The remaining answers that are summarised in Figure 15 have been mentioned by one participant (3.1%) each. Moreover, nine people (28.1%) did not provide any answer at all.

Interestingly, the answers given to this question as well as those provided for the question about tools used during the internationalization phase are strikingly similar. For example, POEditor, Poedit, translate.js, Java resource files, Netbeans, “manually”, “a text editor”, Weblate and Crowdin were mentioned both times, which could either imply that

the internationalization and localization phase are not actually detachable and are rather interwoven into each other, or it could mean that some of the participants are not very aware of the differences between these two processes and consider them to be interchangeable.

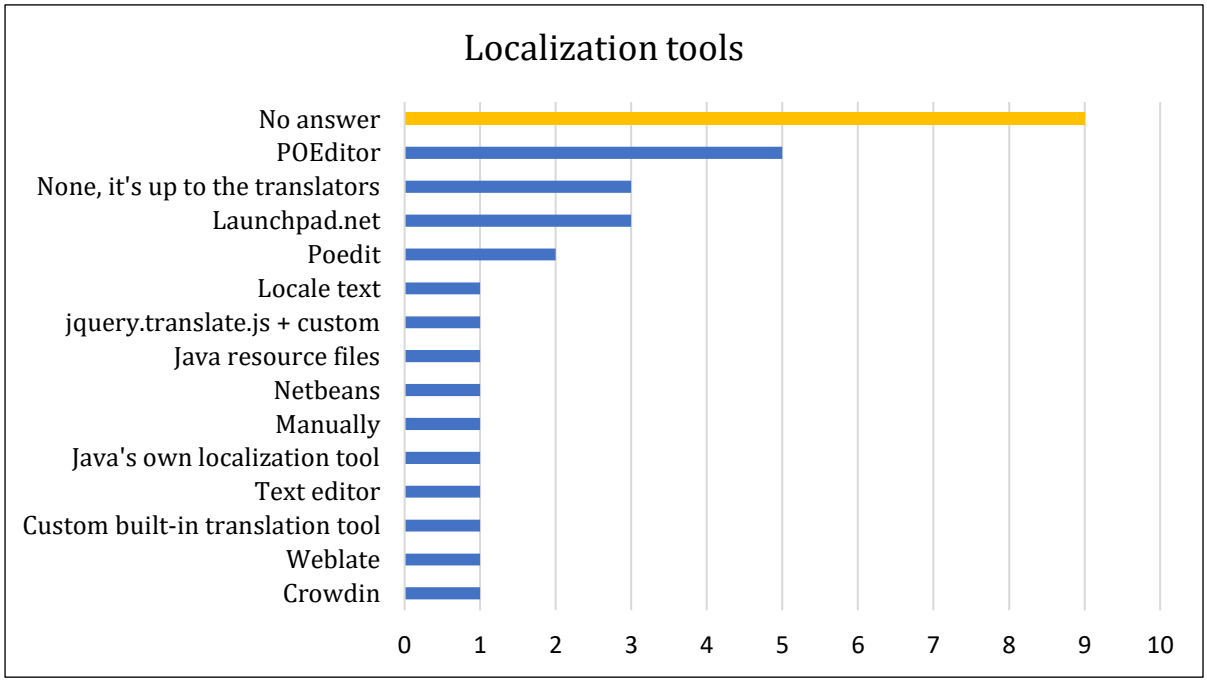


Figure 15. Localization tools

It is also worth noting that 12 out of the 15 localization tools and platforms that were provided as pre-set answers were not chosen by a single participant. This is most likely because those tools require a paid subscription, while the three pre-set answers that were indeed chosen by some participants (namely POEditor, Crowdin and Weblate) also offer a free albeit more restricted version. Lastly, it was also observed that every participant either chose one single answer, provided one localization tool themselves in the comment box or did not answer the question at all. Nobody seemed to have used multiple tools during the localization process.

Machine translation

Out of the 32 participants that had claimed that their software was available in another language, only four (12.5%) responded to have used machine translation in the process. All four participants said they used the machine translation engine Google Translate, while Yandex Translate and DeepL were additionally mentioned by one participant (3.1%) each. Only one person (25%) responded positively to the question whether they

trained the machine translation system with documents related to their software. Lastly, three respondents (75%) said that they post-edited the machine translation output before using it in their software, while the other participant (25%) claimed that only some languages were reviewed and others were directly used as such without any review. It should be noted that the latter goes against the best practice in machine translation and that the quality of the machine translation output might not be satisfying if it is not post-edited by a human.

Translators

The responses to the question "Who localized the software" can be seen in Figure 16. There were 21 participants (65.6%) that said that the translations came from volunteers, with nine respondents (42.9%) claiming that they recruited the volunteers via crowdsourcing methods, ten respondents (47.6%) reporting that users had voluntarily contributed translations without the developers having to seek them out, and one participant (4.8%) explaining that they both sought out volunteer translators and had users sending the translations for some languages without being asked. Seven people (21.9%) had friends or acquaintances help them with the localization, while one person (3.1%) used the support of Google Translate. Lastly, 19 out of 32 participants (59.4%) claimed to have localized the software (or at least parts of it) themselves, out of which only six people (31.6%) localized the software themselves without any assistance, while the other 13 respondents (68.4%) checked at least one of the other answers mentioned above.

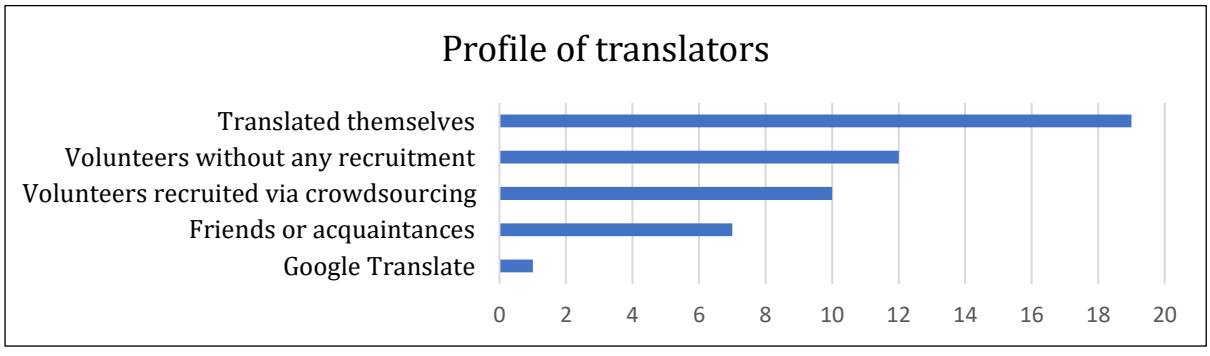


Figure 16. Profile of translators

Interestingly, when looking at the six bigger projects in the data set that have more than 1000 weekly downloads on average, according to the respondents, for all of them the translations were provided by volunteers, three participants (50%) claiming they sought

out the volunteers via crowdsourcing methods and the other three (50%) explaining that the users voluntarily contributed the translations without being asked. One person (16.7%) each additionally reported to have translated the software themselves or to have assigned the task to friends or acquaintances.

Unsurprisingly, 25 out of 32 participants (78.1%) said that they also accept partial translation. This was to be expected since open source software is highly reliant on volunteers (as discussed in section 2.3.1) and therefore cannot afford to impose too many restrictions on the translators. This is also in line with the findings of Alshaikh et al. (2015, p. 3) that software developers often perform partial localization. One person (3.1%) did not provide any answer to this question, which means that only six participants (18.8%) claimed to not accept partial translations. Unexpectedly, it was not the bigger projects with more than 1000 weekly downloads that refuse partial translations. Five out of the six projects (83.3%) that do not accept partial translations have less ten collaborators according to the participants; similarly, four respondents (66.7%) claimed to have less than 50 weekly downloads, while one person (16.7%) reported between 101 and 500 downloads and the remaining participant (16.7%) did not know the number.

When asked what formats the translators worked with, two formats were found to be clearly more popular than the others. As can be taken from Figure 17, 14 participants (43.8%) used PO and MO files, which confirms what Arjona Reina et al. (2013) have said about the Gettext PO file format still being the de facto standard. Another ten participants (31.3%) use Plaintext. When adding together the number of participants who use XML without specifying the specific vocabulary and those who use a specific XML vocabulary, there are four in total (12.5%): two for Android XML (6.3%) and one (3.1%) each for custom XML and YAML. Four participants (12.5%) selected JSON as one of the file formats, while one person (3.1%) each chose nib files and properties files. Lastly, none of the 32 developers of multilingual software selected XLIFF, even though literature suggests that it is superior to the PO format (Frimannsson & Hogan, 2005, pp. 14-15).

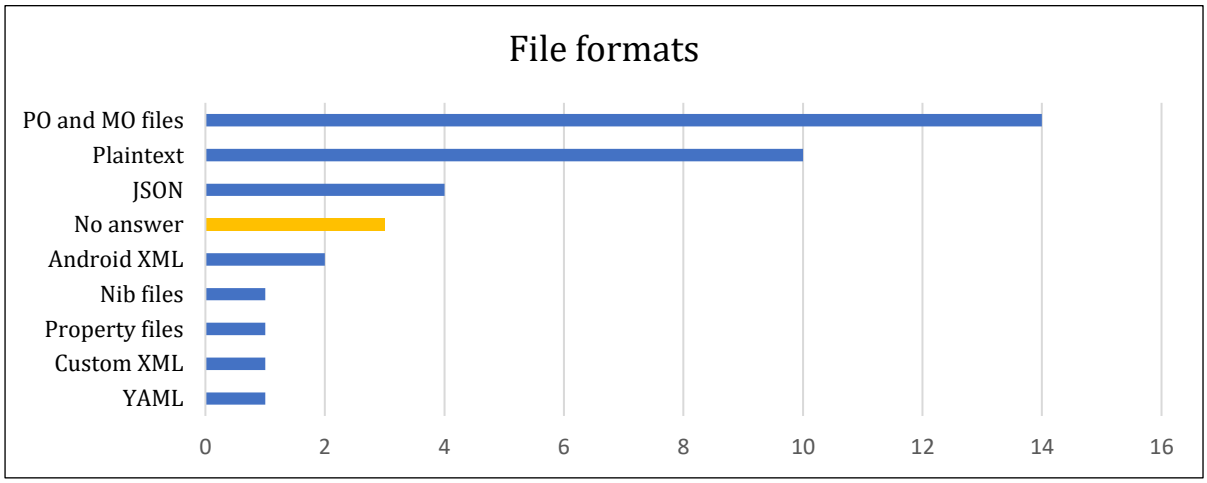


Figure 17. File formats

Interestingly, the difference between the number of people who provide the translators with up to date files versus files of the released version is not that big. While according to 18 participants (56.3%), translators work with the newest files, 14 participants (43.8%) claimed that it is the already finished and released files that are given to the translators to localize.

The answers given to the question regarding the resources that are provided for the translators are presented in Figure 18.

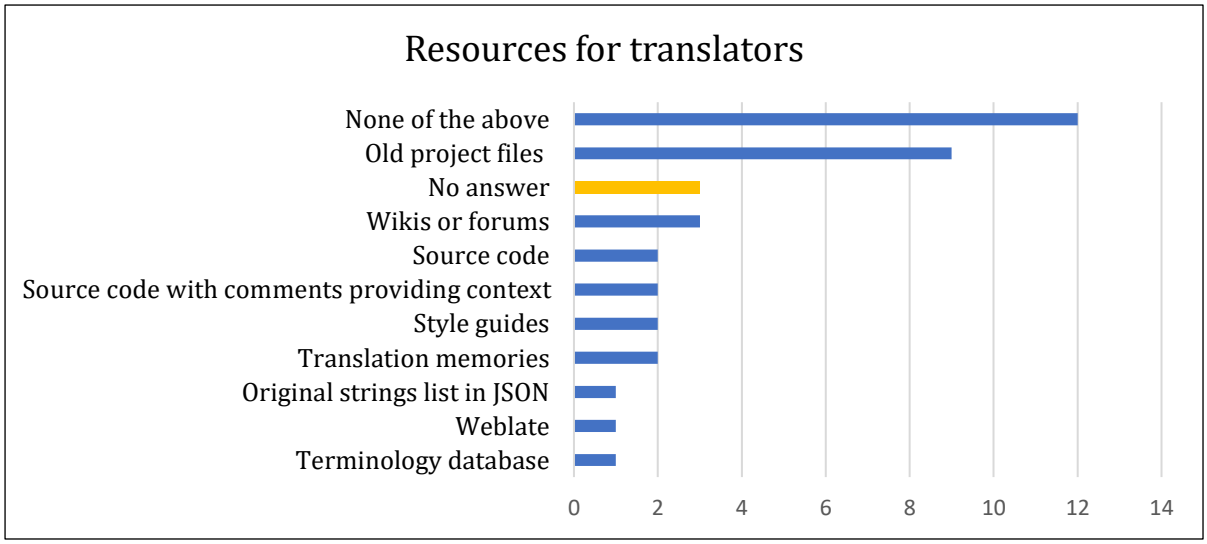


Figure 18. Resources for translators

Nine people (28.1%) made old project files available, while three participants (9.4%) gave access to wikis or forums for the translators. The source code was named as a resource for translators by four respondents (12.5%), two of which specified that they included comments as guidance when necessary. Two people (6.3%) each claimed that they provide translation memories and style guides. One participant (3.1%) each made

available terminology databases and the original text strings list in JSON. Lastly, 12 participants (37.5%) did not choose any of the pre-set answers, namely style guides, old project files, terminology databases, wikis or forums, and also did not provide any answers of their own.

When the participants were asked in a separate question whether the translators have access to the source code, 26 out of 32 (81.3%) answered positively, while only five said no. As explained in section 2.1, this is problematic and could lead to damage in the source code, since translators usually do not have the same technical knowledge as developers. However, some participants might have interpreted this question in the sense that translators have access because the software is open source, not because they had to add the translations directly to the source code (instead of for example using separate resource files). These results therefore need to be interpreted with caution.

In the present data set, 50% of participants claimed they did not provide instructions or documentation for translators on how to submit their contributions, while the other 50% did. In section 2.3.1 it was mentioned that providing instructions on how to contribute translations can be a driving factor for software to be localized into multiple languages (Arjona Reina et al., 2013, p. 163). As can be seen in Figure 19, this has interestingly been confirmed for programs with a large number of target languages, as three out of four projects (75%) with 11-20 target languages and seven out of ten projects (70%) with more than 20 target languages seem to provide instructions on how to contribute translations according to the participants.

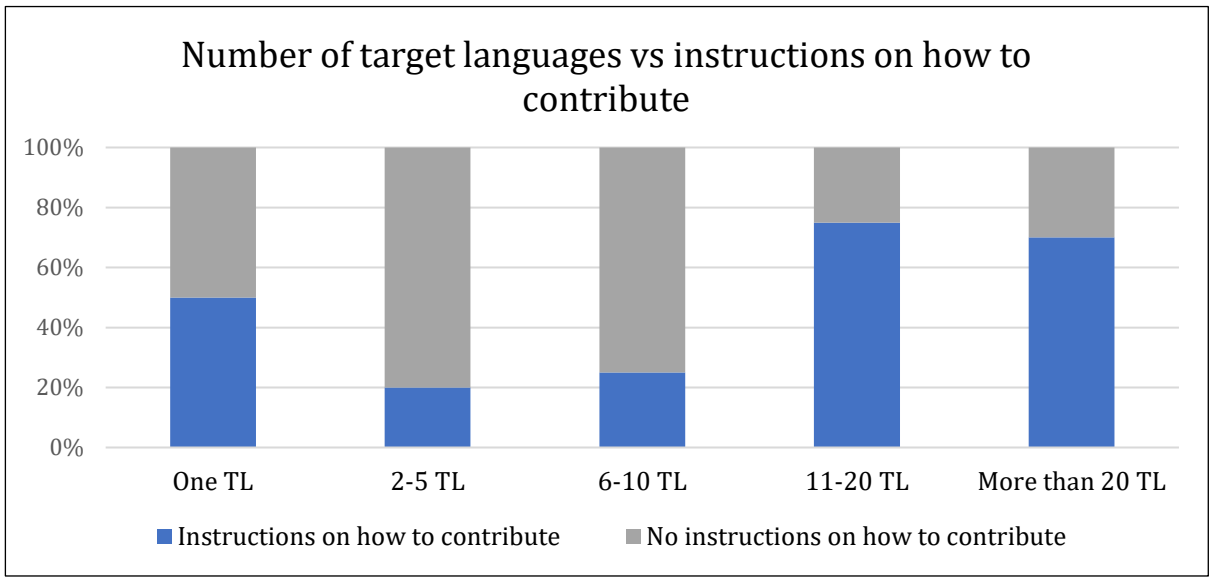


Figure 19. Number of target languages vs instructions on how to contribute

Lastly, Figure 20 shows the frequency of interaction with translators. More than half of the respondents (17 out of 32, 53.1%) said the interaction was sporadic. Similarly, three participants (9.4%) responded that the translators could approach the developers when needed. Five people (15.6%) never interacted with translators, while two respondents (6.3%) interacted with them as much as once a week. Lastly, three participants (9.4%) were the translators themselves and therefore this question did not apply to them, and two people (6.3%) did not provide any answer.

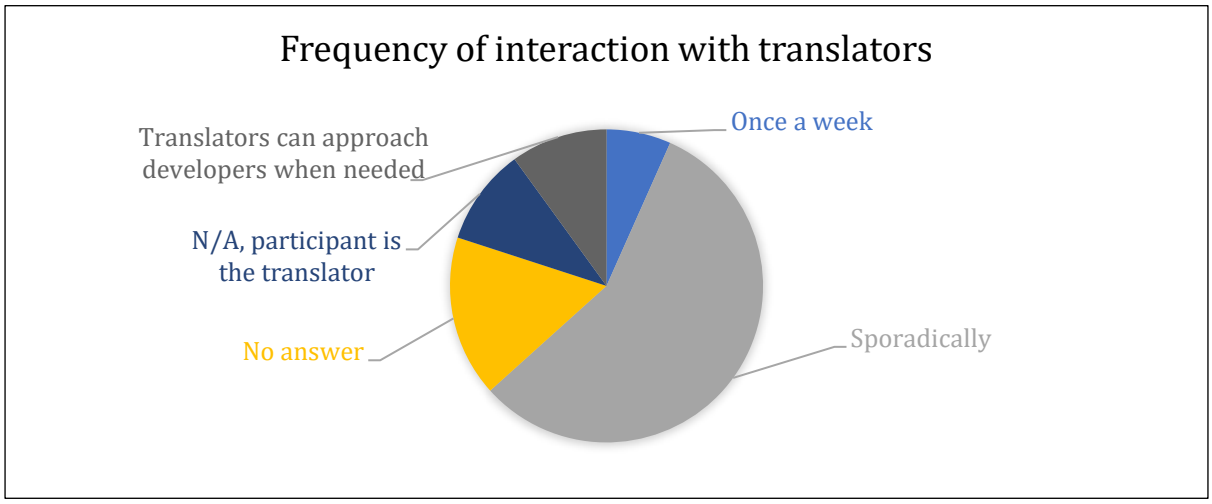


Figure 20. Frequency of interaction with translators

4.3.2 Quality assurance

The following section will look at the quality assurance practices in regard to localized software, while section 4.4.4 will discuss the same in regard to monolingual software. There are three subsections in total: in the first one it will be explained how much time participants dedicated to planning and scheduling before the programming began, the second one goes into detail about what aspects the participants considered important and the third and final subsection touches on the process of testing.

Before continuing, it should be noted that out of the 32 participants that claimed that their software was localized, 19 people (59.4%) said to have carried out some quality assurance tasks, while the other 13 respondents (40.6%) did not.

Planning and scheduling

From Figure 21 it can be observed that 14 out of 32 participants (43.8%) did not dedicate any time to planning and scheduling before the programming stage. Nine participants (28.1%) indicated that planning and scheduling lasted less than a week, which means that 71.9% of contributors and developers of localized software did little to no planning before programming. This confirms Aberdour's (2007, p. 59) claim that in open source, there is little project planning or scheduling. Next, six participants (18.8%) indicated that they set aside 2-4 weeks for planning and two people (6.3%) responded with more than a month. Lastly, one participant (3.1%) did not provide any answer.

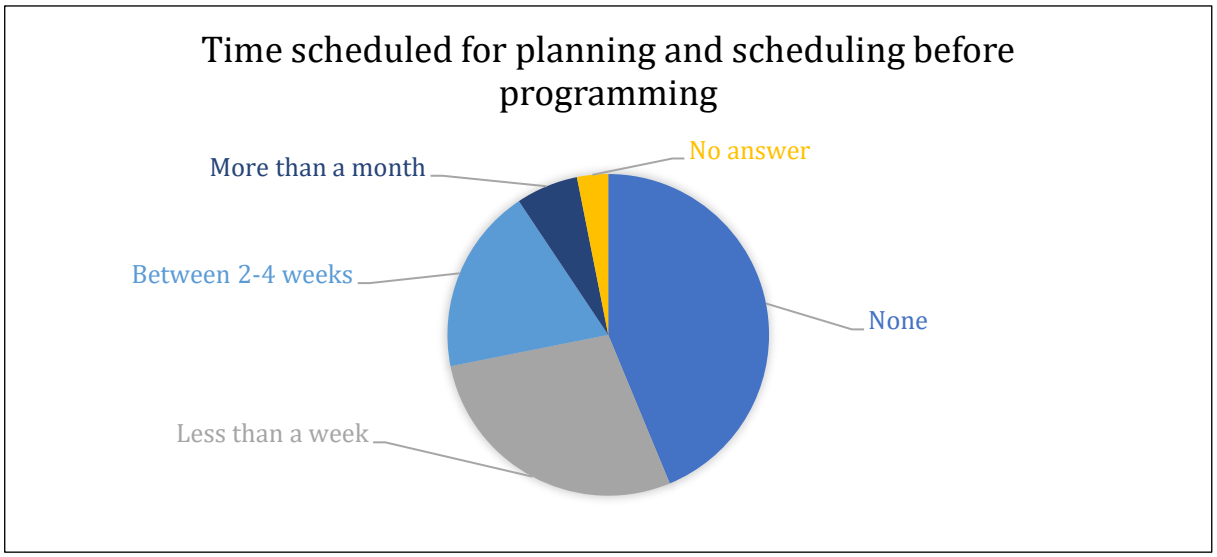


Figure 21. Time scheduled for planning and scheduling before programming

Important aspects of quality assurance

When asked what aspects of quality assurance the participants considered important in order to maintain a satisfying degree of quality, they had the option to provide their own answers, however, all 19 participants that contributed to localized software and had considered quality assurance only chose pre-set answers.

As discussed in section 2.4.1, there are three main types of testing for localized software: cosmetic, functional and linguistic testing. A large majority of participants (15 out of 19, 78.9%) considered at least one type of testing as an important aspect of quality assurance, out of which 13 people (86.7%) had checked cosmetic testing, 12 people (80%) had checked functional testing and six people (40%) had checked linguistic testing. The reason why the latter seems to take a back seat could be that developers of open source

software often translate the software themselves (as seen in section 4.3.1) or that they do not have the time and resources to check the accuracy of translations provided by volunteers.

Next, nine people (47.4%) indicated that they considered peer review important, which is in line with what was discussed in section 2.4.1. Code ownership was mentioned by three people (15.8%), thorough planning was selected by two participants (10.5%) and task ownership as well as central management were chosen by one respondent (5.3%) each. Two participants (10.5%) did not answer this question.

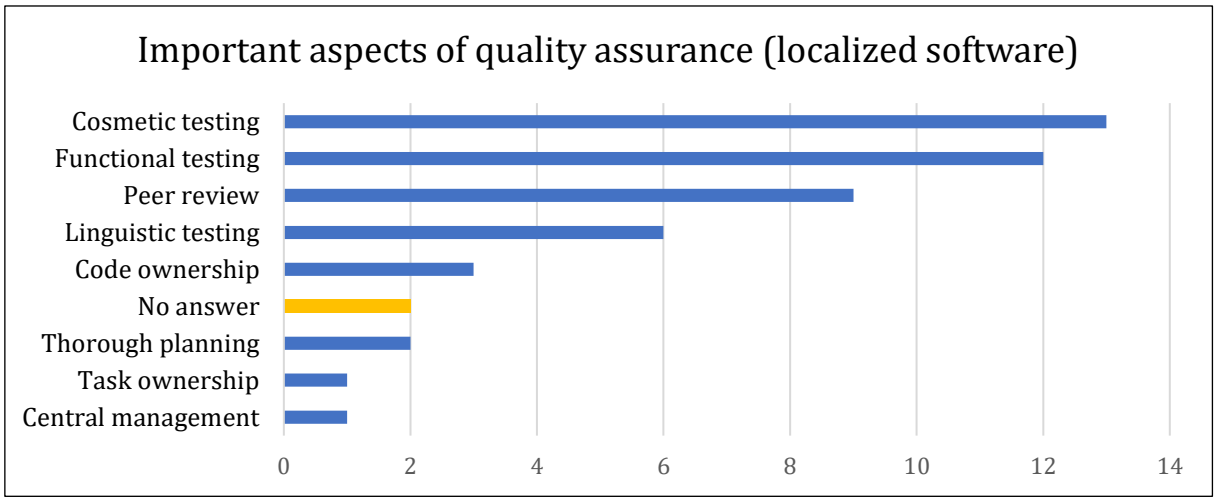


Figure 22. Important aspects of quality assurance (localized software)

Testing

As mentioned above, 15 participants had selected at least one of the three testing options. Out of these 15 participants, all of them (100%) indicated that the developer was responsible for the testing. Six people (40%) mentioned that additionally, the translators also acted as testers. External testers that were native speakers of the target language of the software were recruited in four cases (26.7%), while external testers that were non-native speakers of the target language were mentioned by three participants (20%). Lastly, three respondents (20%) said that testing was also done automatically.

When asked about how many rounds of testing were performed during the whole cycle, six participants (40%) said one round, two participants (13.3%) mentioned two rounds, one person (6.7%) reported four rounds, and surprisingly, five respondents (33.3%) indicated that they performed more than five rounds of testing. One participant (6.7%) did not choose any of the options.

Lastly, all of the contributors of localized software, no matter whether they had performed any quality assurance tasks or not, were asked to indicate how strongly they agree or disagree with the following statement: “High quality in open source software can be achieved by following the ‘release early, release often’ approach instead of focusing on high-quality milestone releases”. Figure 23 shows that five out of 32 people (15.6%) disagreed, 12 people (37.5%) neither agreed nor disagreed, nine people (28.1%) agreed and six people (18.8%) strongly agreed with the statement. Thus, 46.9% either seem to agree or strongly agree that in open source software, more frequent releases are better than waiting until the software has reached certain milestones before releasing it. Only 15.6% seem to disagree with this approach.

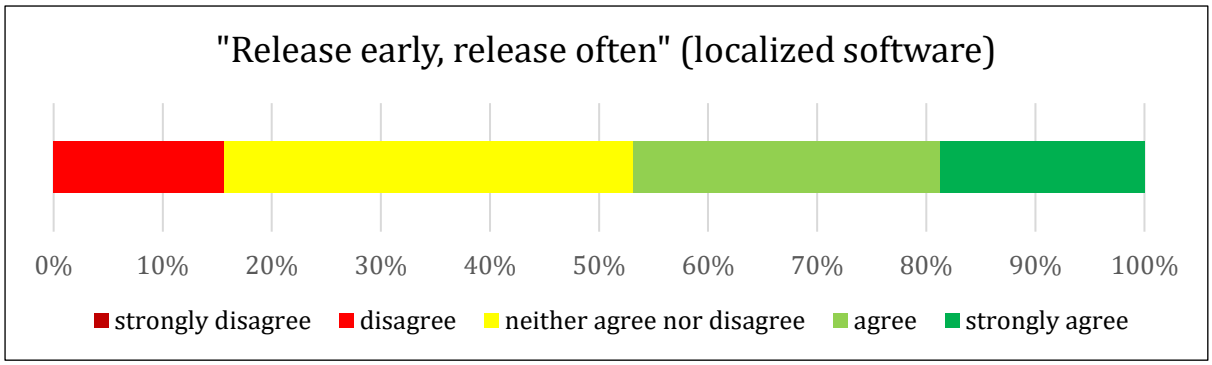


Figure 23. "Release early, release often" (localized software)

4.4 Monolingual software

This section is dedicated to the programs that were not localized. It will also go into detail about the reasoning behind this decision and whether they would consider localization in the future. Lastly, the quality assurance practices will be discussed and the results will be compared to those collected about localized software.

Out of the total of 61 participants, 29 (47.5%) claimed that their software is not localized. While 24 out of these 29 programs (82.8%) are not internationalized either according to the respondents, four people (13.8%) claimed that their software is at least partially internationalized, and one person (3.4%) even responded that it is fully internationalized.

4.4.1 Language of software

When asked what language the software is available in, unsurprisingly, 23 participants (79.3%) responded with English. It is interesting to observe that only seven out of these 23 respondents (30.4%) reported that their native language is English as well. This means that the other 16 participants (69.6%) wrote their software in a language that is not their own, most probably in order to reach a wider audience and because of the status of the English language as a lingua franca.

Next, two people (6.9%) each indicated that their software is in Chinese and German respectively, which after further analysis was found to be their respective mother tongue as well. Lastly, one person (3.4%) responded with a list of 24 languages, even though they answered negatively to the question in the beginning whether their software is available in another language.

4.4.2 Reasons why software is not localized

When asked why the software has not (yet) been localized, the participants could select multiple pre-set answers but also provide their own. The reasons given by the participants are presented in Figure 24.

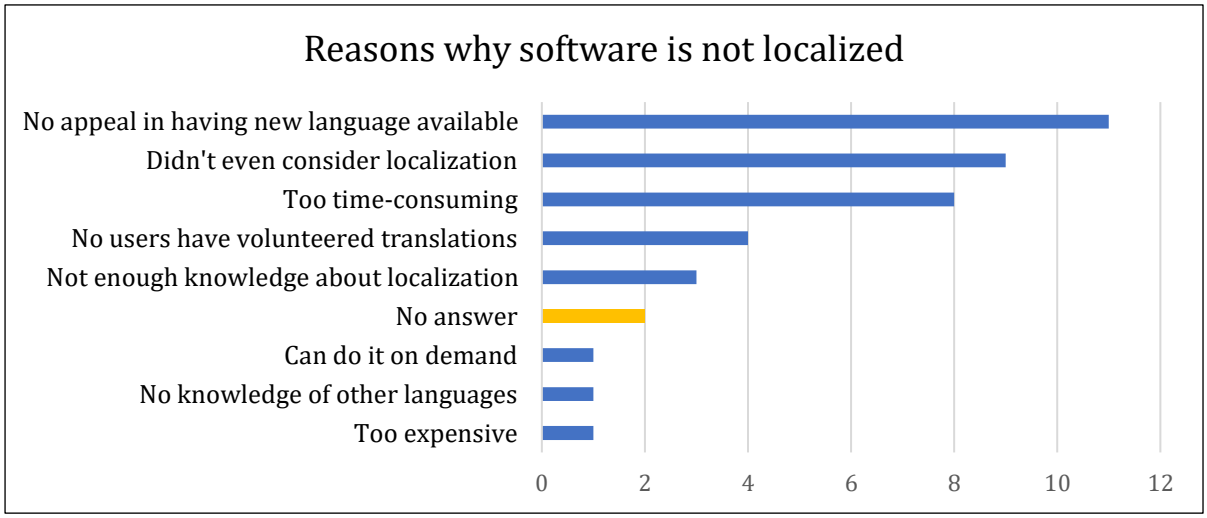


Figure 24. Reasons why software is not localized

Nine people (31%) said that they did not even think about localization, which is in line with what Murtaza and Shwan (2009, p. 36) suggested, namely that the reason why software is often not localized is due to a lack of awareness, not due to a lack of solutions. Similarly, three people (10.3%) claimed to not know enough about localization and one person (3.4%) only speaks one language and therefore has not been able to localize

it themselves. Four people (13.8%) are simply waiting for volunteers to contribute translations and another person (3.4%) responded that they could localize the software on demand. Another eight participants (27.6%) indicated that the localization process would have been too time-consuming, while one person (3.4%) assumed that it would have been too expensive. According to 11 participants (37.9%), there is no appeal in having the software translated into a new language, because either the user base is quite small and limited, the software has been written in an international language, the functionality of the program is being prioritized at the moment or the novelty of the method behind the program is more important than its localization. Lastly, two participants (6.9%) of the questionnaire neither selected any answers nor did they provide their own.

4.4.3 Reconsideration of internationalization and localization in hindsight

The participants were then asked whether they would consider internationalization and localization if they were to redo the project, to which 13 people (68.4%) responded positively. Regarding what languages the participants would want their software to be localized into, German was mentioned seven times (53.8%), French was mentioned five times (33.3%), Spanish, Chinese and Japanese were mentioned three times (20%) each, English and Italian were mentioned twice (13.3%) each, and lastly Romanian, Russian, Dutch and Hindi were mentioned once (6.7%) each. It stands out that even though the participants' answers only contain 11 different languages, the four "FIGS" languages (French, Italian, German and Spanish) that were discussed in section 4.3.1 were included. This confirms yet again that these languages have not lost their importance.

When asked why they would choose these languages in particular, the two most common answers were because they had a personal connection to the language and because of the popularity of the language. These two answers were selected by six participants (40%) each. One person (6.7%) explained that they chose the target language based on its similarity to the language of the original version. Two participants (13.3%) seemed to have skipped this question. Lastly, one person (6.7%) mentioned that their software is unexpectedly popular in Asian countries like China and Japan, which is why it would make sense to localize it into the respective languages.

Eight out of the 13 participants (61.5%) would recruit volunteers to localize their software. Five people (33.3%) each indicated that they would translate it themselves or have friends or acquaintances do it. One person (6.7%) explained that they would use Google translate or a similar machine translation engine.

This also leads us to the final question the participants were asked in this section, namely whether they would make use of machine translation, to which four respondents (30.8%) answered positively and nine respondents (69.2%) answered negatively.

4.4.4 Quality assurance

As mentioned in section 4.3.2, 19 out of the 32 participants (59.4%) whose software is claimed to be localized performed at least some quality assurance tasks. From Figure 25 it can be seen that the distribution is nearly identical for monolingual software, with 18 out of 29 participants (62.1%) indicating they performed at least some quality assurance.

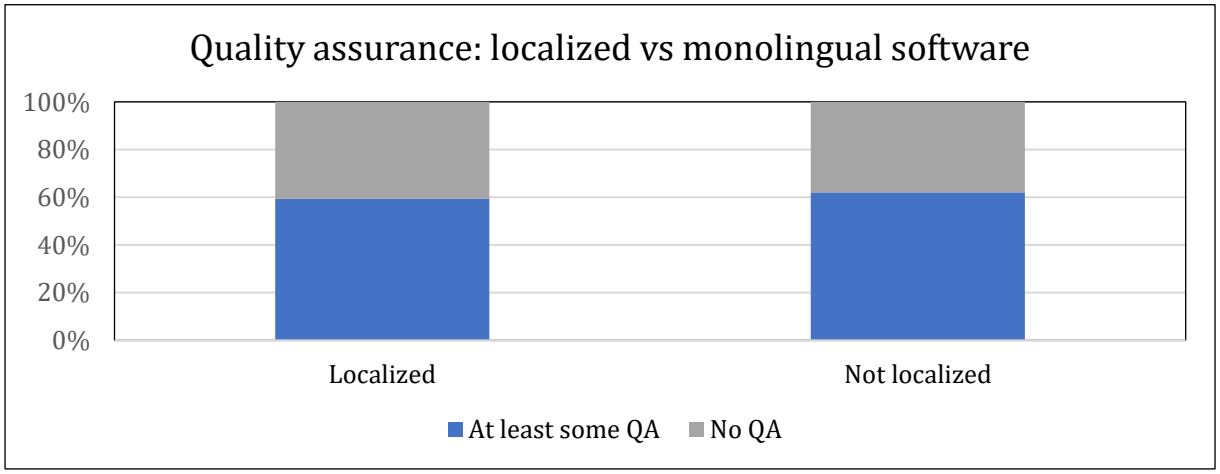


Figure 25. Quality assurance: localized vs monolingual software

When the 29 developers and collaborators of monolingual software were asked how long they each dedicated to planning and scheduling before the programming stage, seven people (24.1%) seemingly did no planning at all, three participants (10.3%) each set aside less than a week or one week respectively and four people (13.8%) responded that they spent between 2-4 weeks planning and scheduling, Lastly and probably must surprisingly, 12 participants (41.4%) planned ahead for more than a month before programming began, which is particularly interesting when comparing it with the

respective numbers for localized software, where only a 6.3% spent more than a month planning and 43.8% did no planning at all (Figure 26).

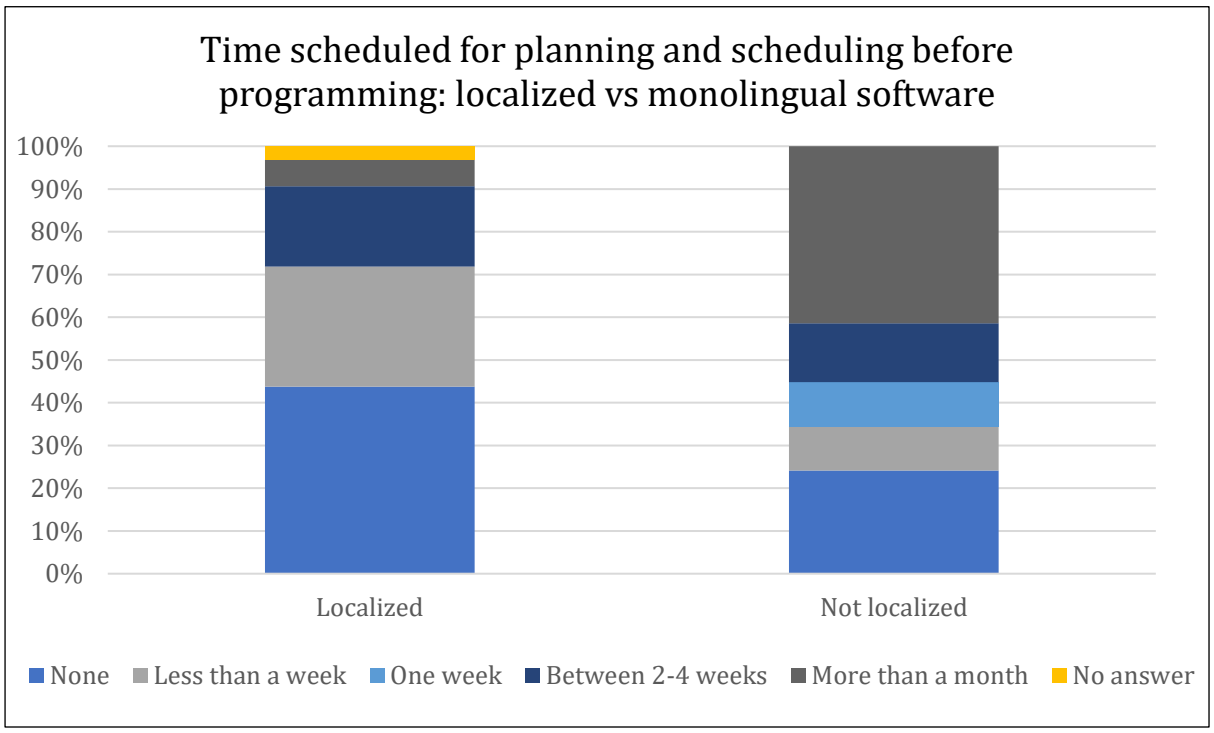


Figure 26. Time scheduled for planning and scheduling before programming: localized vs monolingual software

Next, the participants were asked what aspects they considered important in order to maintain a satisfying degree of quality, the answers of which can be seen in Figure 27.

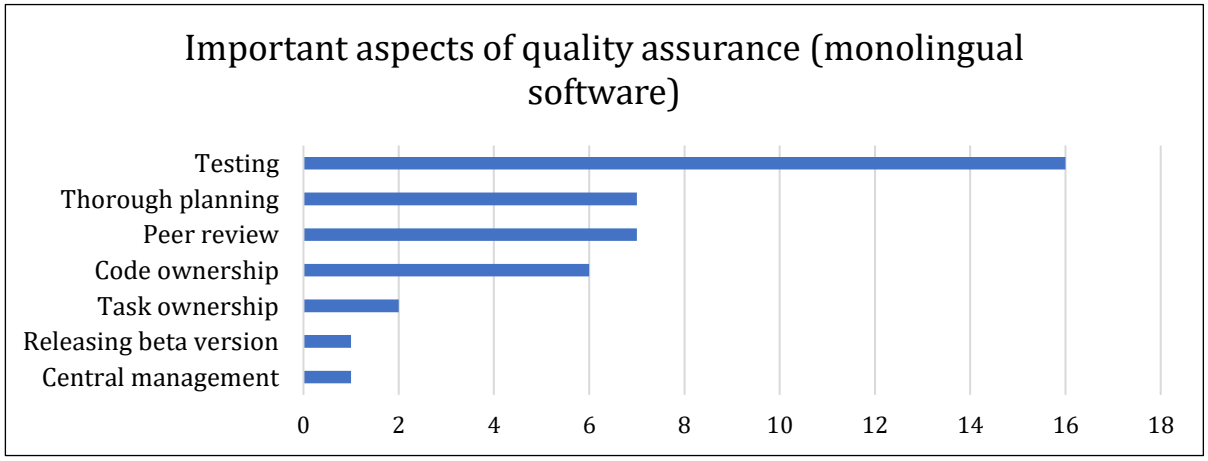


Figure 27. Important aspects of quality assurance (monolingual software)

The vast majority of participants (16 out of 18, 88.9%) claimed they performed testing, one person even specifying the type, namely regression testing. One person (5.6%) also explained that they publish the software early in order to attract more testers, which underlines the importance of the testing stage. Possibly for the same reasons,

another participant (5.6%) explained that they release beta versions. Next, seven participants (38.9%) each considered thorough planning and peer review to be important, and code ownership and task ownership were selected by six (33.3%) and two participants (11.1%) respectively. Lastly, one person (5.6%) selected central management as an important aspect of quality assurance.

Now, regarding testing and who is responsible for it, 15 out of 16 participants (93.8%) who had chosen testing in the previous question said that it was the developer's responsibility. Four participants (25%) also mentioned that they had external testers, while five people (31.3%) used automatic testing as well. It can therefore be observed that both in multilingual and monolingual open source software, automatic testing is not very common, with 9.4% of all developers of multilingual software and 17.2% of all developers of monolingual software indicating they used an automatic testing method.

When comparing these results to those from section 4.3.2 regarding localized software, it can be observed that the percentage of participants considering at least one type of testing is slightly lower (78.9%) than the percentage of participants who work on monolingual software and consider testing as an important aspect of quality assurance (88.9%). The results regarding who is responsible for testing are also very similar between monolingual and localized software as can be seen in Figure 28 (apart from the pre-set answer "Translators" which does not apply for testing of monolingual software).

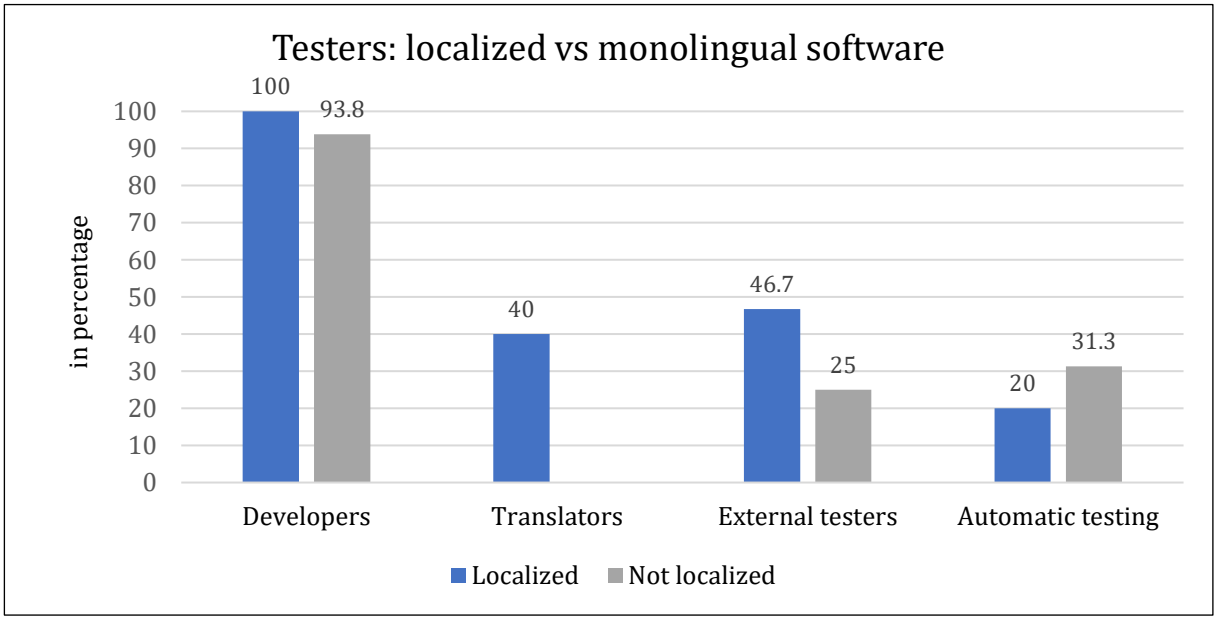


Figure 28. Testers: localized vs monolingual software

When it comes to the number of testing rounds performed during the whole development cycle, two people (12.5%) each selected the answers “one round”, “two rounds” and “three rounds”. One person (6.3%) conducted four rounds of testing, while another person (6.3%) did not select any answer for this question. Lastly, eight people (50%) indicated that they perform more than five rounds of testing.

In Figure 29 the answers given by contributors of monolingual software are compared to those provided by contributors of localized software (section 4.3.2). It can be observed that the percentage of participants that have performed more than five rounds of testing is slightly higher among monolingual software (50% vs 33.3%). Furthermore, 40% of respondents who claimed their software is localized selected the answer “One round”, while only 12.5% of contributors of monolingual software chose the same answer.

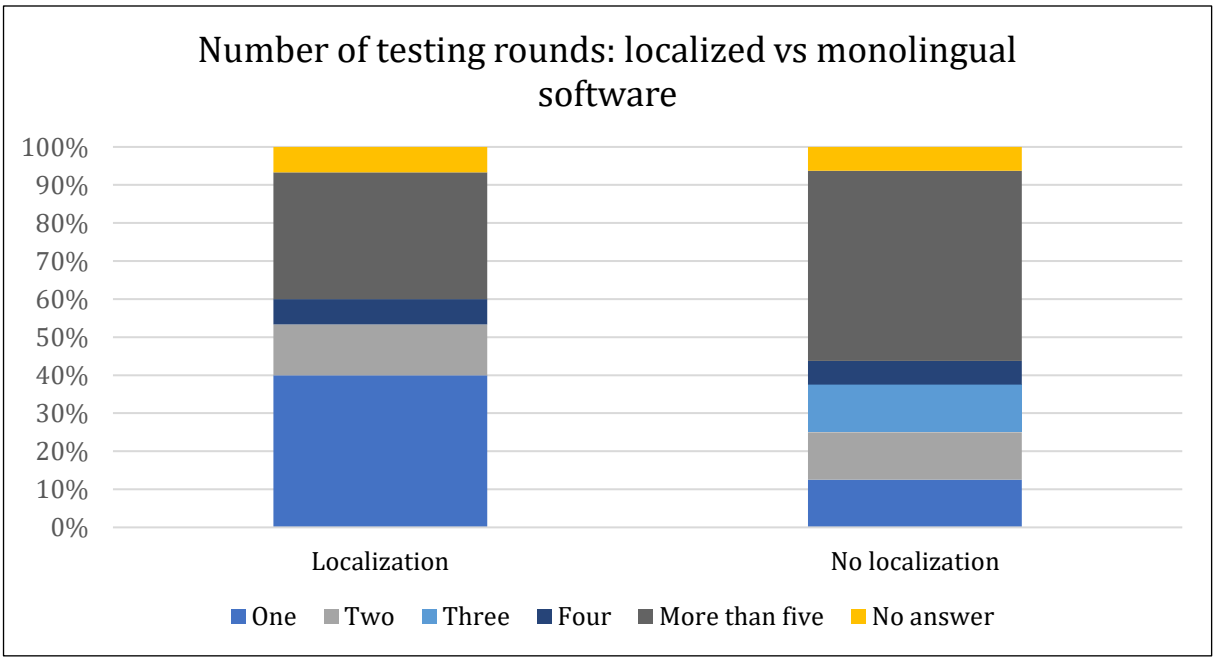


Figure 29. Number of testing rounds: localized vs monolingual software

Finally, the participants were asked how much they agree with the statement that high quality in open source can be achieved by releasing early and often instead of focusing on high-quality milestone releases. Two participants (6.9%) strongly disagree, seven participants (24.1%) disagree, 11 participants (37.9%) neither agree nor disagree, six participants (20.7%) agree, and three participants (10.3%) strongly agree with this statement. It is interesting that the percentage of respondents who either strongly disagree or disagree with this approach is equal to the percentage of respondents who agree or strongly agree with it. Furthermore, as can be seen in Figure 30, there are no

significant differences in distribution when comparing the answers of participants whose software is monolingual to the answers of those respondents whose software is localized.

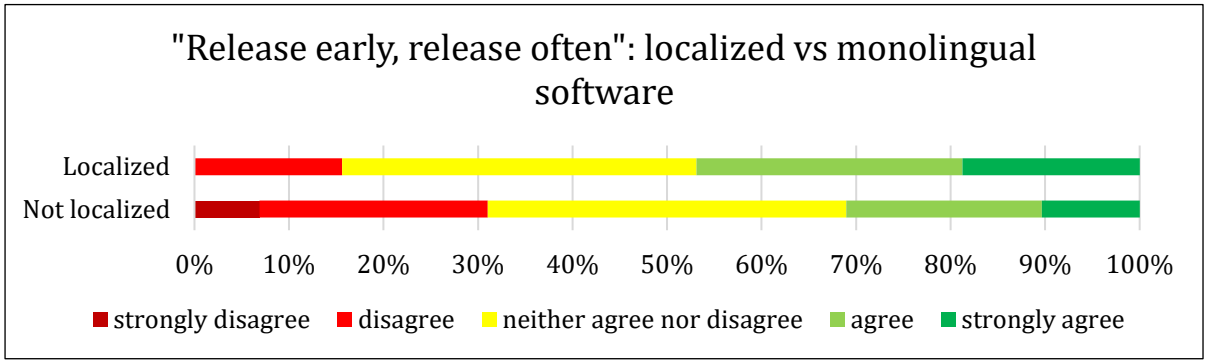


Figure 30. "Release early, release often": localized vs monolingual software

In this chapter, the results that were collected through an online questionnaire were presented and discussed. The next chapter will provide a summary of these results and highlight the most important findings. Furthermore, it will go into detail about the limitations of this thesis and describe topics for further investigations.

5 Conclusion

This research aimed to shed a light on current internationalization, localization and quality assurance practices in open source software. Oftentimes, literature on open source focuses on bigger and more popular projects. However, smaller projects that were developed by teams consisting of very few people arguably represent the majority of existing open source software. In order to be able to compare projects of different sizes, it was decided to limit the research to one type of software, namely image editors.

A list of open source image editors was then compiled by consulting the platforms SourceForge, GitHub and GitLab. To ensure that the collection includes popular image editors that are hosted on other platforms, a few programs were also added ad hoc.

For the purpose of this thesis, an online questionnaire was designed, which gathered information about internationalization, localization and quality assurance in open source. One section is also dedicated to developers of monolingual software. Each section contains specific questions that were deemed relevant in order to get an overview over the current practices in open source regarding internationalization, localization and quality assurance.

Firstly, it was found that while it was attempted to include larger and more popular programs in the data set, there were only very few of them judging by the number of weekly downloads and the number of collaborators. Only 9.8% of the 61 respondents claimed to have more than 1000 downloads per week. Similarly, the number of projects with less than ten collaborators was significantly higher (82%) than the number of projects with more than 50 collaborators (3.3%), indicating the relevance of smaller scaled projects in the domain.

Regarding internationalization, it was observed that 45.9% of all participants had fully internationalized their project, while 14.8% had only done partial internationalization. Interestingly, 78.9% of people who said their software was fully internationalized also claimed that their software contained hardcoded text, which goes against the general guidelines of internationalization and makes us wonder if those respondents were truly aware of internationalization best practices. It was also found that 97.3% of all participants who reported to have at least partially internationalized their software

would consider internationalization again if they were to redo the project, which speaks for how important and useful this step is in the development of software.

When it comes to localization, 32 out of all 61 participants (52.5%) mentioned that their software is available in at least one other language. Out of these 32 respondents, 46.9% said that they had fully localized their software, while the other 53.1% indicated to have at least translated the GUI. This confirms Abufardeh's (2009, p. 52) claim that for the user, having the GUI be available in their native language is very important.

It was also observed that the 32 localized programs in the data set were translated into 146 different languages according to the participants' answers. This large number can be explained by the fact that 21 out of the 32 respondents (65.6%) reported that the translations were provided voluntarily. Since open source is highly reliant on volunteers (see section 2.3.1), the developers generally cannot ask for specific languages and instead let the contributors decide. The contributors of translations are usually native speakers of the respective language and wish for the software they use to be accessible to other speakers of their language. This leads to a higher diversity in target languages than would be the case in commercial localization, where profitability has more value than ideology.

Another interesting observation was that 78.1% of participants indicated to have localized their software into at least one of the "FIGS" languages (French, Italian, German and Spanish). This shows a similarity to commercial localization, where these languages still represent some of the most important target markets.

Lastly, it was found that four out of the 32 localized programs (12.5%) seem to have been translated into 11-20 languages, while another ten programs (31.3%) had more than 20 target languages according to the participants' answers. This is surprising considering nine out of the 14 respondents (64.3%) claiming to have either 11-20 or more than 20 target languages also indicated that less than ten collaborators worked on their project. This could be due to their perception of translators and the fact that they do not consider them as collaborators but rather as external contributors.

When it comes to the file formats the translators work with, it was observed that 43.8% of the respondents reported to use Gettext PO and MO files, which confirms that they are still the de facto standard as literature suggests (Arjona Reina et al., 2013, p. 161; Morado Vázquez & Wolff, 2011, p. 75).

Then it was found that out of the 16 projects that seem to provide information on how to contribute translations, ten have more than 11 target languages according to the participants. This is in line with what Arjona Reina et al. (2013, p. 163) have said about the number of target languages mostly depending on whether there are clear specifications on how to contribute translations.

Lastly, the results of this research make it clear that machine translation is not very common in open source, with only 12.5% of developers of localized software actually using it and 30.8% of developers of monolingual software willing to consider it for future projects.

Next, it was observed that the percentage of developers of localized software who performed some quality assurance tasks was almost identical to the percentage of developers of monolingual software who considered quality assurance (59.4% vs 62.1%). However, when looking at the time the participants spent planning and scheduling before the programming stage, regarding monolingual software, 41.4% of participants dedicated more than a month to the task and 24.1% indicated not to have planned at all beforehand; meanwhile, regarding localized software, only 6.3% of the respondents spent more than one month planning and scheduling, while 43.8% said that they did no planning at all. Unfortunately, it remains unclear why it seems like developers of monolingual software did more planning in the beginning.

It was also found that testing was considered an important aspect of quality assurance for both localized and monolingual software, with 78.9% and 88.9% of participants indicating they performed at least one type of testing. Interestingly, regarding localized software, linguistic testing seems to be less commonly practiced compared to functional and cosmetic testing, probably because open source developers do not have the time nor interest to check the accuracy of translations provided by volunteers. There is certainly also less outside pressure in the open source community compared to commercial localization to achieve a certain level quality. Peer review also seems to be a common practice within open source, as 47.4% of developers of localized software and 38.9% of developers of monolingual software considered it important.

Lastly, it was found that 23 out of the 29 developers of monolingual programs (79.3%) said that their software was written in English. Having said this, only seven out of these 23 respondents (30.4%) said that their native language was English. This means that the

large majority did not write their software in their mother tongue, most probably because of the wider reach of the English language.

To sum up, it becomes clear that the current practices in open source internationalization and localization go against the guidelines described in the literature review. The software still seems to include hard coded text, even though this practice is discouraged. A majority of open source software also still uses the PO and MO files for localization, even though literature suggests that XLIFF should be favoured because of its numerous advantages. Translators are also provided little to no resources to help them with the localization. Therefore, it can be inferred from the results of this research, that the practices in open source do not seem to follow well-established practices in internationalization and localization.

It should be mentioned here again that the focus of this thesis was on image editors in particular. However, because they are usually not specialized applications and are targeted towards the general public, similar results can be expected in other open source software categories such as video editors, operating systems or project management applications.

5.1 Limitations and further work

One limitation of the present thesis is that once the repositories from SourceForge, GitHub and GitLab were compiled into a list, they were manually sifted through to make sure that there were only programs that were (i) current, that (ii) fit the definition of an image editor, that were (iii) open source, that (iv) were standalone programs and that (v) had a graphical user interface. Because this process was carried out by one person, some subjective bias is inevitable. This makes it more difficult for the study to be repeatable. One solution to this issue could be to write a code that would automatically go through the long list of repositories compiled from the platforms and eliminate the irrelevant ones. However, this would probably result in many relevant projects going unnoticed.

Because there were only a few mandatory questions in the beginning of the questionnaire, this meant that sometimes participants skipped a question without selecting any answer. However, the risk with making more questions mandatory is that

participants would lose interest and patience more quickly and could potentially abandon the questionnaire without completing it.

Furthermore, there were 61 participants in this questionnaire (the response rate being 44.2%) which is technically considered a reasonable amount for a master's thesis and its inherent time and resources limitations. However, the structure of the analysis meant that the participants were divided further into smaller sections based on their answers to certain questions, which could have possibly weakened the statistical data set, thereby leaving room for the conclusions to be further investigated to confirm their veracity.

Another shortcoming of this research is that the group of participants did not just consist of developers, who were in fact the target audience for this questionnaire. This meant that some participants might not have known the answers to some of the questions and therefore either skipped the question or worse, chose an answer at random. This is especially problematic since this research solely relies on the participants' answers to draw conclusions. This issue could have been solved by asking the participant in the beginning of the questionnaire to confirm that they are indeed the main developer or at least very involved in the project. However, this would have reduced the number of respondents even further.

For further research, it could be interesting to see whether a similar questionnaire would produce comparable results, if the collection of software was not limited to image editors. Moreover, while the goal of this research was to provide a general overview of the current internationalization, localization and quality assurance practices in open source software, future research could focus on one of these aspects in order to get more in-depth knowledge of the respective area. It could also be interesting to carry out a qualitative study and conduct interviews with developers. Moreover, further studies could also include translators in order to gain a different perspective.

To conclude, this thesis has provided a deeper insight into how internationalization, localization and quality assurance are currently performed in open source, and it has laid the groundwork for future research in these areas. It was found that regarding quality assurance, planning before software development does not seem to be a priority for developers, however, the majority of participants consider testing to be important. Lastly, current practices in open source do not seem to follow well-established practices in internationalization and localization.

6 Bibliography

Aberdour, M. (2007). Achieving Quality in Open-Source Software. *IEEE Software, 24*(1), 58-64.

Abufardeh, S. O. (2009). A framework for the integration of internationalization and localization activities into the software development process. Fargo, North Dakota: North Dakota State University.

Alami, A., Dittrich, Y., & Wąsowski, A. (2018). Influencers of Quality Assurance in an Open Source Community. *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (pp. 61-68). Gothenburg, Sweden: IEEE.

Alshaikh, Z., Mostafa, S., Wang, X., & He, S. (2015). A Empirical Study on the Status of Software Localization in Open Source Projects. *27th International Conference on Software Engineering and Knowledge Engineering*, (pp. 1-4). Pittsburg.

Arjona Reina, L., & Robles, G. (2012). Mining for localization in Android. *Proceedings of the 9th IEEE Working Conference on Mining Software Repositorioes (MSR)* (pp. 136-139). Zurich: IEEE.

Arjona Reina, L., Robles, G., & González-Barahona, J. M. (2013). A Preliminary Analysis of Localization in Free Software: How Translations Are Performed. *IFIP International Conference on Open Source Systems*, (pp. 153-167).

Atlassian. (2022). *Open Source Project License Request*. Retrieved from Atlassian: https://www.atlassian.com/software/views/open-source-license-request (accessed: 20 June 2022)

Atlassian Community. (2022). *Solved: Search public repositories in Bitbucket?* Retrieved from Atlassian Community: https://community.atlassian.com/t5/Bitbucket-questions/Search-public-repositories-in-Bitbucket/qaq-p/844415 (accessed: 19 June 2022)

Bahamdain, S. S. (2015). Open Source Software (OSS) Quality Assurance: A Survey Paper. *Procedia Computer Science, 56*, 459-464.

Behrens, A. (2016). *Lokalisierbarkeit von User-Interface-Strings: übersetzerische Aspekte der Internationalisierung und Lokalisierung von Software, untersucht anhand der Übersetzungsrichtungen Englisch-Deutsch und Englisch-Russisch.* Frankfurt am Main: Peter Lang Edition.

Berthouzoz, M. (2019). Localization choices in indie games: A study on the influencing factors. Université de Genève.

Bitbucket. (2022). *Bitbucket Overview*. Retrieved from Bitbucket: https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket (accessed: 19 June 2022)

Bitbucket. (n.d.). *Repositories*. Retrieved from Bitbucket: https://bitbucket.org/repo/all (accessed: 29 July 2022)

Bitbucket Support. (2022). *Search in Bitbucket Cloud*. Retrieved from Bitbucket Support: https://support.atlassian.com/bitbucket-cloud/docs/search-in-bitbucket-cloud/ (accessed: 19 June 2022)

Cánovas, M., & Samson, R. (2011). Open source software in translator training. *Revista Tradumàtica: tecnologies de la traducció, 9*, 46-56.

Chan, D. (2020). *Sunsetting Mercurial support in Bitbucket*. Retrieved from Bitbucket: https://bitbucket.org/blog/sunsetting-mercurial-support-in-bitbucket (accessed: 20 June 2022)

Codecadamy. (2022). *What is an IDE?* Retrieved from Codecademy: https://www.codecademy.com/article/what-is-an-ide (accessed: 18 July 2022)

DiBona, C. (2015). *Bidding farewell to Google Code*. Retrieved from Google Open Source: https://opensource.googleblog.com/2015/03/farewell-to-google-code.html (accessed: 21 June 2022)

Dino, G. (2021). *What You Need to Know About Image Localization*. Retrieved from atltranslate: https://www.atltranslate.com/articles/image-localization (accessed: 7 June 2022)

Dunne, K. J. (2015). Localization. In C. Si-Wai (Ed.), *The Routledge Encyclopedia of Translation Technology* (pp. 550-562). New York: Routledge.

Esselink, B. (2000). *A Practical Guide to Localization.* Amsterdam/Philadelphia: John Benjamins Publishing Company.

Free Pascal team. (2019). *Free Pascal - Advanced open source Pascal compiler for Pascal and Object Pascal*. Retrieved from Free Pascal: https://www.freepascal.org/ (accessed: 18 July 2022)

Frimannsson, A., & Hogan, J. M. (2005). Adopting Standards-based XML File Formats in Open Source Localisation. *Localisation Focus*, 9-23.

GALA. (2020a). *Language Services*. Retrieved from https://www.gala-global.org/knowledge-center/about-the-industry/language-services (accessed: 25 March 2022)

GALA. (2020b). *Language technology*. Retrieved from https://www.gala-global.org/knowledge-center/about-the-industry/language-technology (accessed: 25 March 2022)

Garcés, C. V., & Toudic, D. (2015). Technological Innovation and Translation. Training Translators in the EU for 21st century. (S. S. Martinez, Ed.) *Verbeia, 0*, 183-202.

Garcia, I. (2015). Computer-aided Translation. In C. Sin-wai, *The Routledge Encylopedia of Translation Technology* (pp. 68-87). London and New York: Routledge.

Gaspari, F., Almaghout, H., & Doherty, S. (2015). A survey of machine translation competences: Insights for translation technology educators and practitioners. *Perspectives, 23*(3), 333-358.

GitHub. (2022). *About - GitHub*. Retrieved from GitHub: https://github.com/about (accessed: 17 June 2022)

GitLab. (2022). *About GitLab*. Retrieved from GitLab: https://about.gitlab.com/company/ (accessed: 17 June 2022)

GitLab. (n.d.). *GitLab Advanced Search*. Retrieved from GitLab: https://docs.gitlab.com/ee/user/search/advanced_search.html#faster-searches (accessed: 29 July 2022)

GNU Operating System. (2020). *gettext*. Retrieved from https://www.gnu.org/software/gettext/ (accessed: 20 April 2022)

Hill, S. (2020). *What is Video Game Localization and Why Should I Care?* Retrieved from Localize direct: https://www.localizedirect.com/posts/what-is-localization/ (accessed: 9 June 2022)

Ishida, R., & Miller, S. K. (2005). *Localization vs. Internationalization*. Retrieved from W3C: https://www.w3.org/International/questions/qa-i18n (accessed: 28 March 2022)

Jiménez-Crespo, M. A. (2017). *Crowdsourcing and Online Collaborative Translations: Expanding the limits of translation studies.* Amsterdam: John Benjamins.

Jiménez-Crespo, M. A. (2018). Crowdsourcing and Translation Quality: Novel Approaches in the Language Industry and Translation Studies. In J. Moorkens, S. Castilho, F. Gaspari, & S. Doherty, *Translation Quality Assessment: From Principles to Practice* (Vol. 1, pp. 69-93). Springer.

Kelly, N., Ray, R., & DePalma, D. A. (2011). From crawling to sprinting: Community translation goes manstream. *Linguistica Anterpiensia, New Series - Themes in Translation Studies, 10*, 75-94.

Launchpad. (2022). *Launchpad*. Retrieved from https://launchpad.net/ (accessed: 26 July 2022)

Lim, S. N. (2022). *Best Machine Translation Software for Enterprise in 2022*. Retrieved from redokun: https://redokun.com/blog/machine-translation-software (accessed: 18 April 2022)

LingoHub. (2020). *Five Great Machine Translation Engines*. Retrieved from https://lingohub.com/blog/2018/11/find-good-machine-translation-engines (accessed: 18 April 2022)

Melby, A. K. (2019). Future of machine translation - Musings on Weaver's memo. In M. O'Hagan, *The Routledge Handbook of Translation and Technology* (pp. 419-436). London and New York: Routledge.

Moorkens, J. (2018). What to expect from Neural Machine Translation: a practical in-class translation evaluation exercise. *The Interpreter and Translator Trainer, 12*(4), 375-387.

Morado Vázquez, L., & Wolff, F. (2011). Bringing industry standards to Open Source localisers: a case study of Virtaal. *Revista Tradumàtica*, 74-83.

Muntés-Mulero, V., Adell, P. P., España-Bonet, C., & Màrquez, L. (2012). Context-Aware Machine Translation for Software Localization. *Proceedings of the 16th Annual conference of the European Association for Machine Translation* (pp. 77-88). Trento: EAMT.

Murtaza, M., & Shwan, A. (2009). Guidelines for Multilingual Software Development. Göteborg, Sweden: University of Gothenburg.

OASIS. (2014). *XLIFF Version 2.0*. Retrieved from https://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.html#d0e4789 (accessed: 25 April 2022)

Open Source Initiative. (n.d.). *The MIT License*. Retrieved from https://opensource.org/licenses/MIT (accessed: 14 July 2022)

Opensource. (2022). *What is open source?* Retrieved from Opensource.com: https://opensource.com/resources/what-open-source (accessed: 13 May 2022)

Otte, T., Moreton, R., & Knoell, H. D. (2008). Applied Quality Assurance Methods under the Open Source Development Model. *2008 32nd Annual IEEE International Computer Software and Applications Conference* (pp. 1247-1252). Turku, Finland: IEEE.

Peng, W., Yang, X., & Zhu, F. (2009). Automation technique of software internationalization and localization based on lexical analysis. *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (pp. 970-975). Seoul, Korea: Association for Computing Machinery.

Peslak, A., & Hunsinger, D. S. (2020). Open Source Software: A Detailed Analysis of Contributions and Quality. *Proceedings of the Conference on Information Systems Applied Research* (pp. 1-11). Virtual Conference: ISCAP.

Poedit. (2022). *Poedit Translation Editor - Poedit*. Retrieved from https://poedit.net/ (accessed: 26 July 2022)

POEditor. (2022). *POEditor - Software Localization Management Platform*. Retrieved from POEditor: https://poeditor.com/ (accessed: 22 July 2022)

Pym, A. (2004). *The Moving Text: Localization, translation, and distribution* (Vol. 49). Amsterdam/Philadelphia: John Benjamins Publishing Company.

Pym, A. (2011). Translation research terms: a tentative glossary for moments of perplexity and dispute. In A. Pym, *Translation Research Projects 3* (pp. 75-110). Tarragona: Intercultural Studies Group.

Qt Wiki. (2022). *About Qt - Qt Wiki*. Retrieved from Qt: https://wiki.qt.io/About_Qt (accessed: 18 July 2022)

Qun, L., & Xiaojun, Z. (2015). Machine Translation. In C. Sin-wai, *The Routledge Encyclopedia of Translation Technology* (pp. 105-119). London and New York: Routledge.

Ramler, R., & Hoschek, R. (2017). Process and Tool Support for Internationalization and Localization Testing in Software Product Development. *Product-Focused Software Process Improvement - 18th International Conference, PROFES 2017* (pp. 385-393). Innsbruck, Austria: Springer.

Raspberry Pi Foundation. (n.d.). *What is a Raspberri Pi?* Retrieved from Raspberry Pi Foundation: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/ (accessed: 17 July 2022)

Rigby, P. C., German, D. M., Cowen, L., & Storey, M.-A. (2014). Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. *ACM Transactions on Software Engineering and Methodology, 23*(4), 33 pages.

Roturier, J. (2015). *Localizing Apps: A practical guide for translators and translation students.* London/New York: Routledge.

Ryan, L., Anastasiou, D., & Cleary, Y. (2009). Using Content Development Guidelines to Reduce the Cost of Localising Digital Content. *Localisation Focus - The International Journal of Localisation, 8*(1), 11-28.

Saldanha, G., & O'Brien, S. (2014). *Research Methodologies in Translation Studies.* London and New York: Routledge.

Sanchez Espinoza, M. V. (2015). The localization of open-source video games: a descriptive and explanatory analysis of SourceForge's most popular video games. Université de Genève.

Sarigül, S., & Ross, J. M. (2020). Volunteer vs. Professional Community Translation in Video Game Localization: The Case of the Steam Translation Server in Turkish. *transLogos, 3*(2), 1-22.

Schäler, R. (2010). Localization and translation. In Y. Gambier, & L. van Doorslaer, *Handbook of Translation Studies* (pp. 209-214). Amsterdam/Philadelphia: John Benjamins Publishing Company.

Shuttleworth, M. (2015). Translation Management Systems. In C. Sin-Wai, *Routledge Encyclopedia of Translation Technologies* (pp. 678-691). London and New York: Routledge.

Singh, S. (2021). *A Review of 5 Machine Translation Engines*. Retrieved from K International: https://us.k-international.com/blog/a-review-of-5-machine-translation-engines/ (accessed: 18 April 2022)

Sirshar, M., Ali, A., & Ibrahim, S. (2019). A Comparitive Analysis Between Open Source And Closed Source Software in Terms of Complexity and Quality Factors. Preprints. doi:2019120063

Smith, B. (2022). *A Quick Guide to GPLv3 - GNU Project - Free Software Foundation*. Retrieved from GNU Operating System: https://www.gnu.org/licenses/quick-guide-gplv3.en.html (accessed: 14 July 2022)

Soh, M., Nkenlifack, M., & Fotso, L. P. (2016). A New Hybrid Process for Software Development and Localisation. *International Journal of Scientific & Engineering Research, 7*(6), 945-953.

Souphavanh, A., & Karoonboonyanan, T. (2005). *Free/Open Source Software: Localization.* United Nations Development Programme - Asia Pacific Development.

SourceForge. (2022a). *About SourceForge*. Retrieved from SourceForge: https://sourceforge.net/about (accessed: 17 June 2022)

SourceForge. (2022b). *Turbo Pascal (With DOSBox) download*. Retrieved from SourceForge: https://sourceforge.net/projects/turbopascal-wdb/ (accessed: 18 July 2022)

Stallman, R. (2022). *Why Upgrade to GPLv3 - GNU Project - Free Software Foundation*. Retrieved from GNU Operating System: https://www.gnu.org/licenses/rms-why-gplv3.html (accessed: 14 July 2022)

Stewart, K., & Gosain, S. (2006). The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Quarterly, 30*(2), 291-314.

Torres del Rey, J., & Morado Vázquez, L. (2015). XLIFF and the Translator: Why Does it Matter? *Revista Tradumàtica: tecnologies de la traducció, 13*, 584-607.

van Wendel de Joode, R., & de Bruijne, M. (2006). The Organization of Open Source Communities: Towards a Framework to Analyze the Relationship between Openness and Reliability. *Proceedings of the 39th Hawaii International Conference on System Sciences* (pp. 1-7). Kuaui: IEEE.

W3C. (2016). *Internationalization tools*. Retrieved from https://www.w3.org/International/tools (accessed: 18 April 2022)

Wang, X., Chen, C., & Xing, Z. (2019). Domain-specific machine translation with recurrent neural network for software localization. *Empirical SOftware Engineering, 24*(6), 3514-3545.

Wolff, F. (2011). *Effecting Change Through Localisation.* Translate.org.za.

Zhao, L., & Elbaum, S. (2000). A Survey On Quality Related Activities in Open Source. *Software Engineering Notes, 25*(3), 54-57.

Appendix A: List of source code hosting sites

Blog site[10]:	Opensource.com	Fossnytes	Techxerl	Ionos	Guru99	It' s Foss	Beebom	Yalantis	Tecmint	Cybercity.biz	Sofftware Testing Help	G2	Blog.deslinux.net	DZone	Software Sustainability Institute	Altamira	**Total**
Github	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	16
Gitlab	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	15
SourceForge	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	15
Bitbucket	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	14
Launchpad		✓				✓			✓		✓		✓	✓	✓		7
Apache Allura			✓	✓	✓	✓			✓		✓		✓				7
Beanstalk					✓		✓		✓		✓	✓	✓	✓			7
Gitea					✓	✓			✓	✓	✓		✓	✓			7
Cloud Source Repositories				✓	✓	✓	✓						✓	✓	✓		6
Gogs					✓	✓			✓	✓			✓	✓			6
Gitbucket					✓				✓	✓	✓		✓	✓			6
Phabricator					✓	✓			✓	✓			✓	✓			6
Gitkraken		✓		✓			✓				✓			✓			5
AWS CodeCommit					✓	✓	✓						✓	✓			5
CodeGiant					✓								✓	✓			3
RhodeCode					✓								✓				2
Trac					✓								✓				2
GNU Savannah										✓					✓		2
Buddy												✓	✓	✓			2
Assembla												✓	✓		✓		2

[10] All of the links in this table were last accessed in May 2021.

RubyForge			✓														1
Tarc			✓														1
Google Code			✓														1
TaraVault					✓												1
GitPrep										✓							1
Kallithea										✓							1
TuleapL										✓							1
Azure DevOps Server												✓	✓				1
Jfrog Artifactory												✓	✓				1
Helix Core												✓	✓				1
Sourcehut													✓				1
Codeberg														✓			1

Appendix B: List of all ad hoc software

Google search:	(image OR graphic OR photo) editor "open source"									(image OR graphic OR photo) manipulation "open source"	
Blog site[11]:	goodfirms	opensource	Fixthephoto	geekflare	lifewire	medevel	openw3	saasgenius	Techwebstuff	shareable	**Total:**
Gimp	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
darktable	✓	✓	✓			✓				✓	5
RawTherapee	✓		✓			✓		✓		✓	5
Inkscape	✓		✓	✓	✓				✓		5
Digikam	✓		✓			✓		✓		✓	5
Krita		✓	✓	✓	✓			✓			5
Paint.net			✓		✓	✓	✓				4
Photivo	✓					✓				✓	3
shotwell		✓	✓							✓	3
LightZone			✓			✓				✓	3
Fotoxx	✓									✓	2
Skencil				✓					✓		2
FontForge				✓					✓		2
PhotoScape X						✓	✓				2
ImageOptim	✓										1
Pinta		✓									1
Luminar 4			✓								1
Pixen			✓								1
Synfig Studio				✓							1
Dia				✓							1
Blender				✓							1
Pencil2D				✓							1
SVG-Edit				✓							1
GIMPShop						✓					1
GIMPPhoto						✓					1

[11] All of the links in this table were last accessed in May 2021.

PhotoFlare						✓					1
Apple Photos						✓					1
Fotor Photo editor						✓					1
PhotoDemon						✓					1
Pixia							✓				1
Photo shine							✓				1
Online Photoshop Editor								✓			1
mtPaint									✓		1
UFRaw										✓	1
Rapid Photo Downloader										✓	1
F-Spot										✓	1
Photoflow										✓	1
Pixls										✓	1
Hugin										✓	1

Appendix C: Reasons for exclusion of software (with examples)

Reason for exclusion	Examples[12]
Not an image editor according to the definition established for this thesis	• DigitalPalette (SourceForge) • taesungp/swapping-autoencoder-pytorch (GitHub) • ToaruOS / toaruos (GitLab) • pH_7 / Simple Java Text Editor (Bitbucket) • FontForge (ad hoc)
No open source license	• photopea/photopea (GitHub) • Xuân Tiến Nguyễn / Ckeditor_upload_image (GitLab) • dawenzhang/image_editor (Bitbucket) • PhotoScape X (ad hoc)
Only a software component	Library: • Agar (SourceForge) Plug-ins • Thomas Fihla / gimp-icns (GitLab) SDK • imgly/pesdk-ios-examples (GitHub) Wrapper: • kus/php-image (GitHub) Module: • selenium-health / react-native-image-editor (Bitbucket)
No GUI	• GraphicsMagick (SourceForge) • processone/eimp (GitHub) • Ryan Hanson / Keyboard Image Editor (GitLab)
Empty repository	• gridarta (SourceForge) • StudyNeoflex / GraphicsEditor (GitLab)
Mercurial repository	• aklexel / Sikuli-Eclipse_7f9d6f_hg (Bitbucket)

[12] All of the links in this table were last accessed in May 2021.

Appendix D: Full list of selected software

	Software[13]	SourceForge	GitHub	GitLab	Ad hoc
1	AwesomeBump	✓			
2	BRIC	✓			
3	CinePaint	✓			
4	CirPaint	✓			
5	Crowd3	✓			
6	DelphiDoom Voxel Editor	✓			
7	Draft Notes	✓			
8	Drawj2d	✓			
9	ENDEDIT	✓			
10	Evolvotron	✓			
11	FidoCadJ	✓			
12	Flexxi - Batch Image Resizer	✓			
13	Frappe Charts	✓			
14	GamEditorDS	✓			
15	Grandshop	✓			
16	GuitarRosette	✓			
17	JDrafting	✓			
18	JIVAM	✓			
19	JaxoDraw	✓			
20	KediCAD	✓			
21	LCARS Item Tool	✓			
22	LDPatternCreator	✓			
23	LaTeXDraw	✓			
24	LodePaint	✓			
25	MagicPhotos	✓			
26	Mike's Sprite Editor	✓			
27	PRICE - image filtering and manipulation	✓			
28	Paint 2d	✓			
29	Paintbrush	✓			
30	Pix	✓			
31	Pixelitor	✓			
32	Screentake - Screen Capture	✓			

[13] All of the links in this table were last accessed in May 2021.

33	Sniptool - Screen Capture	✓			
34	Sodipodi	✓			
35	SpriteSinger2	✓			
36	TeXCAD	✓			
37	The Seashore Project	✓			
38	Tux Paint	✓			
39	WAD Painter	✓			
40	Xpaint	✓			
41	ZuneFIG	✓			
42	darktable	✓			
43	jPicEdt	✓			
44	jfPaint	✓			
45	mtPaint	✓			
46	nufraw	✓			
47	pixFront	✓			
48	qScreenshot	✓			
49	qvge	✓			
50	rEASYze - Batch Image Resize & Edit	✓			
51	nhn/tui.image-editor		✓		
52	siwangqishiq/ImageEditor-Android		✓		
53	glimpse-editor/Glimpse		✓		
54	viliusle/miniPaint		✓		
55	fengyuanchen/photo-editor		✓		
56	eventtus/photo-editor		✓		
57	chrishutchinson/cardkit		✓		
58	congnd/FMPhotoPicker		✓		
59	maoschanz/drawing		✓		
60	tannerhelland/PhotoDemon		✓		
61	scaleflex/filerobot-image-editor		✓		
62	ApoorvaJ/Papaya		✓		
63	bgrabitmap/lazpaint		✓		
64	MauriceGit/Voronoi_Image_Manipulation		✓		
65	Symbian9/azpainter		✓		
66	Catrobat/Paintroid		✓		
67	microsoft/Windows-appsample-photo-editor		✓		
68	jogendra/phimpme-iOS		✓		

69	zenden2k/image-uploader		✓		
70	ttddee/Cascade		✓		
71	Oberto/php-image-magician		✓		
72	AntonKast/LightZone		✓		
73	OnurErtugral/Photo-Editor		✓		
74	jsh-me/simple-android-editor		✓		
75	aurbano/nuophoto		✓		
76	TanTanDev/vimnail		✓		
77	GimelStudio/Gimel-Studio		✓		
78	dan335/nimp		✓		
79	alexge50/gie		✓		
80	malulleybovo/SymbolArtEditorOnline		✓		
81	he-dhamo/simsg		✓		
82	artf/grapesjs-tui-image-editor		✓		
83	bitsed/qosmic		✓		
84	interwebsug/photo-editor-sdk-react-native		✓		
85	yomboprime/colorator		✓		
86	ImEditor/ImEditor		✓		
87	yang-han/GraphicsEditor		✓		
88	vinniciusgomes/photoeditor		✓		
89	bvirxx/bLUe_PYSIDE2		✓		
90	YaoKaiLun/react-img-editor		✓		
91	VISNode/VISNode		✓		
92	ncruces/RethinkRAW		✓		
93	speechly/photo-editor-demo		✓		
94	linKhehe/Zane		✓		
95	bharathvaj-ganesan/picart		✓		
96	TridentBot/Trident		✓		
97	gonzaloruizdevilla/image-manipulation-assemblyscript		✓		
98	codenameakshay/image-editor		✓		
99	omnister/piglet		✓		

100	shiyangzhaoa/image-editor		✓		
101	andrewyancey/ImageManipulation		✓		
102	igorski/bitmappery		✓		
103	Eduard Malokhvii / 3D Master			✓	
104	surt / GfxPaint			✓	
105	Stephan Sandriesser / image-editor			✓	
106	Adam Friedrich Schrey / image_and_buttons			✓	
107	Seachal Zhang / ImageEditor-Android			✓	
108	Inkscape / inkscape			✓	
109	Azkar Moulana / phimpme-android			✓	
110	genaro gonzalez / Pics			✓	
111	Matej Frankic / pixEd			✓	
112	Roberto B. / PyShoot			✓	
113	Alsharif / SPixel			✓	
114	TechLord / UEFITool			✓	
115	homebrew-browsergames / virtuelle_welt			✓	
116	deconbatch / Break Up to Make Up.			✓	
117	Fabrizio Ruggeri / caravaggio			✓	
118	SLCU / teamHJ / henrik_aahl / imgmisc			✓	
119	Ján Koloda / imutils			✓	
120	Tamil Selvan / Laravel Imager			✓	
121	Pix / Pix			✓	
122	Michael Anthony / Simp			✓	
123	Jerome KASPER / Faint			✓	
124	fatemeh / Graphic_editor			✓	
125	Jerome KASPER / Graphite			✓	
126	Jamie Davie / pentool			✓	
127	Pablo Martin / PraetoriansMapEditor			✓	
128	Romain Milbert / RaZor			✓	

129	Colin Kiama / UWPVectorGraphicsToolExample			✓	
130	Werxehuja / WebGraphicMathFormulasEditor			✓	
131	Jerome KASPER / XaraLx			✓	
132	Fotoxx / Fotoxx photo editor			✓	
133	Gimp				✓
134	Photivo				✓
135	RawTherapee				✓
136	Digikam				✓
137	Krita				✓
138	shotwell				✓

Appendix E: Questionnaire

Internationalization and Localization Practices in Open Source Image Editors

This survey is conducted by Anupama Nedungadi as part of a master's thesis at the University of Geneva. It aims to understand the internationalization and localization practices in the open source community, specifically image and graphics editors, as little research has been done in this field recently.

The survey is separated into sections and should take you around **10 minutes or less**, depending on some of your answers. Your participation is voluntary and you can choose to withdraw at any time without having to explain yourself or having to suffer any negative consequences. It's also possible to save your answers and resume the survey at a later time. The survey will stay open until **Wednesday, 16 June 2021**.
There is no personal information collected through this survey and all your answers will be stored confidentially on a server owned by the University of Geneva.
Here is a list of contacts that you may approach if you have any questions or complaints:

- Anupama Nedungadi (responsible for this study)
- Lucía Morado Vázquez (supervisor of the master's thesis)

Please note that by participating in this survey, you confirm that you agree with the information above and grant permission to use and publish the collected data for academic purposes only.

There are 63 questions in this survey.

Age

Are you older than 18? *

- ○ Yes
- ○ No

If answer was No: *Unfortunately you cannot take part in this survey if you are a minor. Thank you nonetheless for taking your time!*

General questions about the software

Is the software open source? *

- ○ Yes
- ○ No

If answer was No: *Unfortunately you cannot take part in this survey if your software is not open source. Thank you nonetheless for taking your time!*

What is the name of the software you have been contacted about for this survey? *

In some cases, multiple developers of the same software have been sent this survey. This information is only collected in order to avoid any distortion of the data. Please note that this information will be anonymised after downloading the answers by assigning a unique and random code to each software. This answer will neither be revealed in the final research.

What license is the software under?

- ○ 0-clause BSD License (0BSD)
- ○ 1-clause BSD License (BSD-1-Clause)
- ○ 2-clause BSD License (BSD-2-Clause)
- ○ 3-clause BSD License (BSD-3-Clause)
- ○ Academic Free License 3.0 (AFL-3.0)
- ○ Adaptive Public License (APL-1.0)
- ○ Apache Software License 1.1 (Apache-1.1) (superseded)
- ○ Apache License 2.0 (Apache-2.0)
- ○ Apple Public Source License (APSL-2.0)
- ○ Artistic license 1.0 (Artistic-1.0) (superseded)
- ○ Artistic License 2.0 (Artistic-2.0)
- ○ Attribution Assurance License (AAL)
- ○ Boost Software License (BSL-1.0)
- ○ BSD-3-Clause-LBNL
- ○ BSD+Patent (BSD-2-Clause-Patent)
- ○ CERN Open Hardware Licence Version 2 - Permissive
- ○ CERN Open Hardware Licence Version 2 - Weakly Reciprocal

- CERN Open Hardware Licence Version 2 - Strongly Reciprocal
- CeCILL License 2.1 (CECILL-2.1)
- Common Development and Distribution License 1.0 (CDDL-1.0)
- Common Public Attribution License 1.0 (CPAL-1.0)
- Common Public License 1.0 (CPL-1.0) (superseded)
- Computer Associates Trusted Open Source License 1.1 (CATOSL-1.1)
- Cryptographic Autonomy License v.1.0 (CAL-1.0)
- CUA Office Public License Version 1.0 (CUA-OPL-1.0) (retired)
- Eclipse Public License 1.0 (EPL-1.0) (superseded)
- Eclipse Public License 2.0 (EPL-2.0)
- eCos License version 2.0 (eCos-2.0)
- Educational Community License, Version 1.0 (ECL-1.0) (superseded)
- Educational Community License, Version 2.0 (ECL-2.0)
- Eiffel Forum License V1.0 (EFL-1.0) (superseded)
- Eiffel Forum License V2.0 (EFL-2.0)
- Entessa Public License (Entessa)
- EU DataGrid Software License (EUDatagrid)
- European Union Public License 1.2 (EUPL-1.2) (links to every language's version on their site)
- Fair License (Fair)
- Frameworx License (Frameworx-1.0)
- Free Public License 1.0.0 (0BSD)
- GNU Affero General Public License version 3 (AGPL-3.0)
- GNU General Public License version 2 (GPL-2.0)
- GNU General Public License version 3 (GPL-3.0)
- GNU Lesser General Public License version 2.1 (LGPL-2.1)
- GNU Lesser General Public License version 3 (LGPL-3.0)
- Historical Permission Notice and Disclaimer (HPND)
- IBM Public License 1.0 (IPL-1.0)
- Intel Open Source License (Intel) (retired)
- IPA Font License (IPA)
- ISC License (ISC)
- Jabber Open Source License (retired)

- LaTeX Project Public License 1.3c (LPPL-1.3c)
- Lawrence Berkeley National Labs BSD Variant License (BSD-3-Clause-LBNL)
- Licence Libre du Québec – Permissive (LiLiQ-P) version 1.1 (LiliQ-P)
- Licence Libre du Québec – Réciprocité (LiLiQ-R) version 1.1 (LiliQ-R)
- Licence Libre du Québec – Réciprocité forte (LiLiQ-R+) version 1.1 (LiliQ-R+)
- Lucent Public License ("Plan9"), version 1.0 (LPL-1.0) (superseded)
- Lucent Public License Version 1.02 (LPL-1.02)
- Microsoft Public License (MS-PL)
- Microsoft Reciprocal License (MS-RL)
- MirOS Licence (MirOS)
- MIT License (MIT)
- MIT No Attribution License (MIT-0)
- MITRE Collaborative Virtual Workspace License (CVW) (retired)
- Motosoto License (Motosoto)
- Mozilla Public License 1.0 (MPL-1.0) (superseded)
- Mozilla Public License 1.1 (MPL-1.1) (superseded)
- Mozilla Public License 2.0 (MPL-2.0)
- Mulan Permissive Software License v2 (MulanPSL - 2.0)
- Multics License (Multics)
- NASA Open Source Agreement 1.3 (NASA-1.3)
- Naumen Public License (Naumen)
- Nethack General Public License (NGPL)
- Nokia Open Source License (Nokia)
- Non-Profit Open Software License 3.0 (NPOSL-3.0)
- NTP License (NTP)
- OCLC Research Public License 2.0 (OCLC-2.0)
- Open Group Test Suite License (OGTSL)
- Open Software License 1.0 (OSL-1.0) (superseded)
- Open Software License 2.1 (OSL-2.1) (superseded)
- Open Software License 3.0 (OSL-3.0)
- OpenLDAP Public License Version 2.8 (OLDAP-2.8)
- OSET Public License version 2.1
- PHP License 3.0 (PHP-3.0) (superseded)

- ○ PHP License 3.01 (PHP-3.01)
- ○ The PostgreSQL License (PostgreSQL)
- ○ Python License (Python-2.0) (overall Python license)
- ○ CNRI Python license (CNRI-Python) (CNRI portion of Python License)
- ○ Q Public License (QPL-1.0)
- ○ RealNetworks Public Source License V1.0 (RPSL-1.0)
- ○ Reciprocal Public License, version 1.1 (RPL-1.1) (superseded)
- ○ Reciprocal Public License 1.5 (RPL-1.5)
- ○ Ricoh Source Code Public License (RSCPL)
- ○ SIL Open Font License 1.1 (OFL-1.1)
- ○ Simple Public License 2.0 (SimPL-2.0)
- ○ Sleepycat License (Sleepycat)
- ○ Sun Industry Standards Source License (SISSL) (retired)
- ○ Sun Public License 1.0 (SPL-1.0)
- ○ Sybase Open Watcom Public License 1.0 (Watcom-1.0)
- ○ Universal Permissive License (UPL)
- ○ University of Illinois/NCSA Open Source License (NCSA)
- ○ Upstream Compatibility License v1.0
- ○ Unicode Data Files and Software License
- ○ The Unlicense
- ○ Vovida Software License v. 1.0 (VSL-1.0)
- ○ W3C License (W3C)
- ○ wxWindows Library License (WXwindows)
- ○ X.Net License (Xnet)
- ○ Zero-Clause BSD (0BSD)
- ○ Zope Public License 2.0 (ZPL-2.0)
- ○ zlib/libpng license (Zlib)
- ○ Other:

_____

When was the software last updated?

- ○ This month
- ○ Within the last six months

- ○ In 2020
- ○ In 2019
- ○ Before 2019

What platform(s)/operating system(s) does your software run on?

- ☐ Android
- ☐ Chrome OS
- ☐ iOS
- ☐ iPadOS
- ☐ Linux
- ☐ MacOS
- ☐ Windows
- ☐ Other:

_____

Which programming language did you use? If you used more than one or the one you used is not present in the list, please specify in the comment box.

- ○ 4th Dimension/4D
- ○ ABAP
- ○ ABC
- ○ ActionScript
- ○ Ada
- ○ Agilent VEE
- ○ Algol
- ○ Alice
- ○ Angelscript
- ○ Apex
- ○ APL
- ○ AppleScript
- ○ Arc
- ○ Arduino
- ○ ASP
- ○ AspectJ

- Assembly
- ATLAS
- Augeas
- AutoHotkey
- AutoIt
- AutoLISP
- Automator
- Avenue
- Awk
- Bash
- (Visual) Basic
- bc
- BCPL
- BETA
- BlitzMax
- Boo
- Bourne Shell
- Bro
- C
- C Shell
- C#
- C++
- C++/CLI
- C-Omega
- Caml
- Ceylon
- CFML
- cg
- Ch
- CHILL
- CIL
- CL (OS/400)
- Clarion

- Clean
- Clipper
- Clojure
- CLU
- COBOL
- Cobra
- CoffeeScript
- ColdFusion
- COMAL
- Common Lisp
- Coq
- cT
- Curl
- D
- Dart
- DCL
- DCPU-16 ASM
- Delphi/Object Pascal
- DiBOL
- Dylan
- E
- eC
- Ecl
- ECMAScript
- EGL
- Eiffel
- Elixir
- Emacs Lisp
- Erlang
- Etoys
- Euphoria
- EXEC
- F#

- Factor
- Falcon
- Fancy
- Fantom
- Felix
- Forth
- Fortran
- Fortress
- (Visual) FoxPro
- Gambas
- GNU Octave
- Go
- Google AppsScript
- Gosu
- Groovy
- Haskell
- haXe
- Heron
- HPL
- HyperTalk
- Icon
- IDL
- Inform
- Informix-4GL
- INTERCAL
- Io
- Ioke
- J
- J#
- JADE
- Java
- Java FX Script
- JavaScript

- JScript
- JScript.NET
- Julia
- Korn Shell
- Kotlin
- LabVIEW
- Ladder Logic
- Lasso
- Limbo
- Lingo
- Lisp
- Logo
- Logtalk
- LotusScript
- LPC
- Lua
- Lustre
- M4
- MAD
- Magic
- Magik
- Malbolge
- MANTIS
- Maple
- Mathematica
- MATLAB
- Max/MSP
- MAXScript
- MEL
- Mercury
- Mirah
- Miva
- ML

- Monkey
- Modula-2
- Modula-3
- MOO
- Moto
- MS-DOS Batch
- MUMPS
- NATURAL
- Nemerle
- Nimrod
- NQC
- NSIS
- Nu
- NXT-G
- Oberon
- Object Rexx
- Objective-C
- Objective-J
- OCaml
- Occam
- ooc
- Opa
- OpenCL
- OpenEdge ABL
- OPL
- Oz
- Paradox
- Parrot
- Pascal
- Perl
- PHP
- Pike
- PILOT

- PL/I
- PL/SQL
- Pliant
- PostScript
- POV-Ray
- PowerBasic
- PowerScript
- PowerShell
- Processing
- Prolog
- Puppet
- Pure Data
- Python
- Q
- R
- Racket
- REALBasic
- REBOL
- Revolution
- REXX
- RPG (OS/400)
- Ruby
- Rust
- S
- S-PLUS
- SAS
- Sather
- Scala
- Scheme
- Scilab
- Scratch
- sed
- Seed7

- Self
- Shell
- SIGNAL
- Simula
- Simulink
- Slate
- Smalltalk
- Smarty
- SPARK
- SPSS
- SQR
- Squeak
- Squirrel
- Standard ML
- Suneido
- SuperCollider
- TACL
- Tcl
- Tex
- thinBasic
- TOM
- Transact-SQL
- Turing
- TypeScript
- Vala/Genie
- VBScript
- Verilog
- VHDL
- VimL
- Visual Basic .NET
- WebDNA
- Whitespace
- X10

- xBase
- XBase++
- Xen
- XPL
- XSLT
- XQuery
- yacc
- Yorick
- Z shell
- Other:

_____

How many downloads do you have on average per week?

- Less than 10
- 11-50
- 51-100
- 101-500
- 501-1000
- 1001-10’000
- More than 10’000
- I don’t know

How many people collaborate on your software?

- Less than 10
- 10-30
- 31-50
- More than 50

Is your software available in more than one language? *

- Yes
- No

Internationalization

Internationalization is the design of software code bases and resources that allow an application to be adapted to various locales without requiring changes to the code base. (Globalization and Localization Association, GALA)

Is your software internationalized?

- ○ Yes
- ○ No
- ○ It is partially internationalized

Does your software contain any hardcoded text?

- ○ Yes
- ○ No

Hard coded text refers to text strings that are directly embedded into the source code as opposed to stored in external sources.

At what stage of the development of the software did you think about internationalization?

- ○ Before development
- ○ Early stage of development
- ○ Near the end of development
- ○ After completion or release in its first language

What aspect(s) did you consider most important when internationalizing the software?

- ☐ Accommodation for text string expansion
- ☐ Character encoding
- ☐ Text directionality (e.g. right-to-left vs left-to-right)
- ☐ Progress bar directionality
- ☐ Page layout
- ☐ Number formats (e.g. period vs comma as a decimal separator)
- ☐ Calendar system
- ☐ Date and time formats (e.g. MM/DD/YYYY vs DD/MM/YYYY or 12-hour vs 24-hour clock)
- ☐ Sounds

☐ Other:

What tool(s) did you use in the internationalization process?

☐ Unicode code converter

☐ UniView

☐ Encoding converter

☐ List characters tool

☐ W3C Internationalization checker

☐ Globalyzer

☐ Gettext

☐ Other:

In retrospect, if you were to redo the project, would you consider internationalization?

○ Yes

○ No

Why wouldn't you consider internationalization?

☐ It was too time consuming.

☐ It was too expensive.

☐ It was not worth it for the localization process.

☐ Other:

Localization

Localization is the process of adapting a product or service to a specific locale. The aim of localization is to give a product or service the look and feel of having been created specifically for a target market, no matter their language, cultural preferences, or location. (Globalization and Localization Association, GALA)

Is your software fully localized?

- ○ Yes
- ○ No

In this case "fully localized" refers to the software's user interface (including error messages, tooltips, etc.) as well as any additional material (e.g. help files, documentation, website, etc.) being fully available in another language.

How much is translated into another language?

- ☐ User interface
- ☐ Help files & documentation
- ☐ Website
- ☐ Other:

_____

Are you continuing to localize parts of your software?

- ○ Yes
- ○ No, I am satisfied with the degree of localization

At what stage of the development of the software did you think about localization?

- ○ Before development
- ○ Early stage of development
- ○ Near the end of development
- ○ After the software was released in its first language

When was/were the localized version(s) released?

- ○ Simultaneously with the original version
- ○ After the original version
- ○ Other:

_____

Which language(s) has your software been translated into? If you chose a specific locale, please specify that as well.

Please write your answer here:

_____

A locale consists of a base language and a country or territory. This accounts for regional differences as some languages are spoken in multiple countries and regions.
Example: Spanish (Spain) vs Spanish (Argentina) or French (Canada) vs French (Switzerland)

Why did you choose this/these language(s)?

- ☐ Because I have a personal connection to the language(s)
- ☐ Because of the similarity to the language of the original version
- ☐ Because of the popularity of the language(s) and the wide audience it would therefore reach
- ☐ Other:

_____

Do you plan to add more languages in the future?

- ○ Yes
- ○ No

How does the software switch from one language to another?

- ○ Statically, by restarting
- ○ Dynamically without restarting
- ○ Other:

_____

Did you use machine translation?

- ○ Yes
- ○ No

What machine translation engine(s) did you use?

- ☐ Google Translate
- ☐ DeepL or DeepL Pro
- ☐ Microsoft Translator
- ☐ Yandex.Translate
- ☐ Apertium
- ☐ Moses

- ☐ Amazon Translate
- ☐ Systran
- ☐ Other:

Did you train the machine translation system with documents related to your software?

- ○ Yes
- ○ No

How did you use the machine translation input?

- ○ The machine translation is used in combination with a translation memory system.
- ○ The machine translation is post-edited (meaning a human alters the machine-generated translation to achieve an acceptable final product)
- ○ The machine translation is used as is without any reviewing
- ○ Other:

What format(s) did the translator(s) work with?

- ☐ XLIFF
- ☐ Android XML
- ☐ YAML
- ☐ JSON
- ☐ PO and MO files
- ☐ Plaintext
- ☐ Other:

Did the translator(s) have access to the source code?

- ○ Yes
- ○ No

What tool(s) did you use in the localization process?

- ☐ Transifex
- ☐ Lokalise
- ☐ LingoHub
- ☐ POEditor
- ☐ Crowdin
- ☐ SDL Passolo
- ☐ Weblate
- ☐ Webtranslateit.com
- ☐ Smartcat
- ☐ Alchemy Catalyst
- ☐ RC WinTrans
- ☐ Multilizer
- ☐ ResX Localization Studio
- ☐ Lingobit Localizer
- ☐ Visual Localize
- ☐ Other:

_____

Translators

Who localized the software?

- ☐ You translated it yourself
- ☐ You had (a) friend(s) or (an) acquaintance(s) translate it
- ☐ Your employee(s)
- ☐ You hired (a) freelance translator(s)
- ☐ You recruited (a) volunteer translator(s) via crowdsourcing methods
- ☐ Other:

_____

Please select the resource(s) that you provided for the translator(s):

- ☐ Translation memories
- ☐ Style guides

- ☐ Old project files (translations)
- ☐ Terminology databases
- ☐ Wikis or forums
- ☐ None of the above
- ☐ Other:

Did you provide instructions or documentation specifically for translators on how to submit the work?

- ○ Yes
- ○ No

What version did the translator(s) work with?

- ○ They translated the files of the released version
- ○ They were provided up to date files

How often did you interact with the translator(s)?

- ○ Once a week
- ○ Once every two weeks
- ○ Once a month
- ○ Sporadically
- ○ Never
- ○ Other:

Did you accept partial translations?

- ○ Yes
- ○ No

Quality assurance and testing

How much time did you dedicate to planning and scheduling the project before programming?

- ○ None

- ○ Less than a week
- ○ One week
- ○ Between 2-4 weeks
- ○ More than a month

Did you perform any quality assurance?

- ○ Yes
- ○ No

What aspect(s) did you consider important in order to maintain a satisfying degree of quality? (for localized software)

- ☐ Peer review (i.e. the evaluation of the work by one or multiple people with similar competences as the producer of the work, for example other developers, translators, etc.)
- ☐ Linguistic testing (i.e. reviewing every dialog box, menu and as many strings as possible)
- ☐ Cosmetic testing/User interface testing (i.e. reviewing the visual aspects of the software, e.g. number of options in a menu, sizing and alignment of buttons, regional settings in dialog boxes)
- ☐ Functional testing (i.e. verifying that there is no damage to the functionality of the product because of the localization process)
- ☐ Central management
- ☐ Code ownership
- ☐ Task ownership
- ☐ Thorough planning
- ☐ Other:

_____

What aspect(s) did you consider important in order to maintain a satisfying degree of quality? (for monolingual software)

- ☐ Peer reviewing (i.e. the evaluation of the work by one or multiple people with similar competences as the producer of the work, for example other developers, translators, etc.)
- ☐ Testing

- ☐ Central management
- ☐ Code ownership
- ☐ Task ownership
- ☐ Thorough planning
- ☐ Other:

_____

Who was responsible for the testing? (for localized software)

- ☐ Developer(s)
- ☐ Translator(s)
- ☐ External tester(s) (native speaker(s) of the localized version)
- ☐ External tester(s) (non-native speaker(s) of the localized version)
- ☐ Automatic testing
- ☐ Other:

_____

Who was responsible for the testing? (for monolingual software)

- ☐ Developer(s)
- ☐ External tester(s)
- ☐ Automatic testing
- ☐ Other:

_____

How many rounds of testing did you perform during the whole cycle? (for localized software)

- ○ One
- ○ Two
- ○ Three
- ○ Four
- ○ Five
- ○ More than five

How many rounds of testing did you perform during the whole cycle? (for monolingual software)

- ○ One
- ○ Two
- ○ Three
- ○ Four
- ○ Five
- ○ More than five

How much do you agree with this statement: High quality in open source software can be achieved by following the "release early, release often" approach instead of focusing on high-quality milestone releases

- ○ strongly disagree
- ○ disagree
- ○ neither agree nor disagree
- ○ agree
- ○ strongly agree

Internationalization and Localization

Internationalization is the design of software code bases and resources that allow an application to be adapted to various locales without requiring changes to the code base. (Globalization and Localization Association, GALA)

Localization is the process of adapting a product or service to a specific locale. The aim of localization is to give a product or service the look and feel of having been created specifically for a target market, no matter their language, cultural preferences, or location. (Globalization and Localization Association, GALA)

In what language is your software available?

Why is your software not localized into other languages?

- ☐ I don't know enough about localization
- ☐ It is too expensive

- ☐ It would take too much time
- ☐ I didn't even think about having it localized
- ☐ I don't see the appeal of having a new language available
- ☐ Other:

_____

In retrospect, if you were to redo the project, would you consider internationalization and localization?

- ○ Yes
- ○ No

Which language(s) would you localize your software into?

_____

Why would you choose this/these language(s)?

- ☐ Because I have a personal connection to the language(s)
- ☐ Because of the similarity to the language of the original version
- ☐ Because of the popularity of the language(s) and the wide audience it would therefore reach
- ☐ Other:

_____

How would you recruit the translator(s)?

- ☐ I would translate it myself
- ☐ I would have (a) friend(s) or (an) acquaintance(s) translate the software
- ☐ I would recruit (a) volunteer translator(s) via crowdsourcing methods
- ☐ I would hire (a) freelance translator(s)
- ☐ I would turn to translation agencies
- ☐ Other:

_____

Would you integrate machine translation?

- ○ Yes
- ○ No

Demographic questions

What is your age?

- ○ 18-24
- ○ 25-34
- ○ 35-44
- ○ 45-54
- ○ 55-64
- ○ 65 and over

What is your mother tongue?

How long have you been involved in open source projects?

- ○ I'm very new to open source projects
- ○ Less than a year
- ○ 1-5 years
- ○ 6-10 years
- ○ More than 10 years
- ○ I am not involved in open source projects

How would you rate your knowledge of internationalization and localization?

Internationalization:

- ○ 1 – I have never heard of it
- ○ 2
- ○ 3
- ○ 4
- ○ 5 – I am an expert

Localization

- ○ 1 – I have never heard of it
- ○ 2
- ○ 3
- ○ 4
- ○ 5 – I am an expert

Internationalization is the design of software code bases and resources that allow an application to be adapted to various locales without requiring changes to the code base. (Globalization and Localization Association, GALA)
Localization is the process of adapting a product or service to a specific locale. The aim of localization is to give a product or service the look and feel of having been created specifically for a target market, no matter their language, cultural preferences, or location. (Globalization and Localization Association, GALA)

What is your role in the project?

If you wish to be informed about the results of this study, please enter your email address below. You will be contacted as soon as the results are available.

Please note that your email address will only be used for communication purposes and within the framework of this thesis. It will be stored separately from the other answers so you won't be identifiable and only the main researcher will have access to it.

Thank you for taking the time to fill out this survey! Your contribution is really valuable to my research and it is greatly appreciated. Have a good day!

Appendix F: Invitation and reminder for the questionnaire

1. Invitation

Dear _____,

My name is Anupama Nedungadi and I am a Master's student at the University of Geneva. For my thesis, I am looking at internationalization and localization practices in open source image editors, meaning how and why (or why not) the software has been translated into another language.

I have curated a list of image editors, graphic editors and the likes hosted on SourceForge, GitHub and GitLab, and this list includes your software _____.

I am contacting you to ask for your participation in a survey. This is a key element of my master's thesis. I am aware of the fact that we don't know each other and that you don't owe me anything, but I am really dependent on your help since my topic is quite niche. I also believe we should shine a spotlight on open source software and its translation in particular, as little research has been done in this area recently.

The survey will be open until the 16th of June 2021, and it should only take you around 10 minutes as most questions are multiple choice. Your answers will be anonymous.

https://formulaire.unige.ch/outils/limesurveyfac/traduction-interpretation/index.php/697247?lang=en

Again, thank you for taking the time to read this email and helping a slightly stressed student out!

Best regards,

Anupama Nedungadi

2. Reminder

Dear _____,

My name is Anupama Nedungadi and I am a Master's student at the University of Geneva. I had already contacted you about the software _____ and your participation in a survey for my master's thesis about internationalization and localization practices in open source image editors last week.

I hope you don't mind me contacting you again, but your participation would be really valuable to my research as I won't be able to draw any quantitative conclusions without a certain number of responses.

The survey will be open until the 16th of June 2021, and it should only take you around 10 minutes as most questions are multiple choice. Your answers will be anonymous.

https://formulaire.unige.ch/outils/limesurveyfac/traduction-interpretation/index.php/697247?lang=en

I understand that you must be really busy, so thank you for helping me out, I really appreciate it!

Best regards,

Anupama Nedungadi

Appendix G: List of all target languages of localized software

Language	Mentions
French	21
German	21
Spanish	18
Chinese China (simplified)	16
Japanese	16
Italian	15
Dutch	14
Russian	13
English	12
Portuguese	12
Swedish	12
Czech	11
Turkish	11
Catalan, Valencian	10
Polish	10
Portuguese Brazil	10
Greek modern	9
Hungarian	9
Serbian Serbia	9
Arabic	8
English UK	8
Finnish	8
Galician	8
Hebrew	8
Twi	8
Chinese Taiwan	7
Danish	7
English US	7
Korean	7
Slovak	7
Slovenian	7
Vietnamese	7
Albanian	6
Bulgarian	6
Croatian	6
Hindi	6
Macedonian	6
Romanian	6
Tamil	6
Thai	6
Afrikaans	5
Basque	5
Esperanto	5
Farsi	5
Indonesian	5

Kabyle	5
Latvian	5
Lithuanian	5
Norwegian Bokmal	5
Ukrainian	5
Azerbaijani	4
Bengali	4
English Canadian	4
Estonian	4
Gaelic Irish	4
Gujarati	4
Icelandic	4
Malay	4
Malayalam	4
Marathi	4
Nepali	4
Norwegian Nynorsk	4
Occitan	4
Punjabi India	4
Sindhi	4
Urdu	4
Belarusian	3
Bosnian	3
Breton	3
Central Khmer	3
Chinese Hong-Kong	3
English, Australian	3
Kannada	3
Kazakh	3
Kinyarwanda	3
Maithili	3
Sinhalese	3
Spanish Mexico	3
Tagalog	3
Telugu	3
Welsh	3
Xhosa	3
Amharic	2
Armenian	2
Assamese	2
Asturian	2
Bodo India	2
Dogri	2
Dzongka	2
Georgian	2
Interlingua	2
Kashmiri Devanagari	2
Konkani	2
Kurdish	2
Low German	2
Manipuri	2
Oriya	2

Pashto	2
Sanskrit	2
Santali	2
Swahili	2
Tajik	2
Uzbek	2
Walloon	2
Acholi	1
Akan	1
Aragones	1
Bambara	1
Bengali Bangladesh	1
Berber languages	1
English South Africa	1
Faroese	1
Farsi Iran	1
French Canada	1
Fula	1
Gaelic Scottish	1
Gronings	1
Hausa	1
Igbo	1
Inuktitut	1
Kashmiri Perso-Arabic	1
Kiga	1
Klingon	1
Latin	1
Luganda	1
Luxembourgish	1
Magyar	1
Malayalam India	1
Miahuatlan Zapotec	1
Mongolian	1
Ndebele	1
Northern Sami	1
Northern Sotho	1
Norwegian	1
Okibwe	1
Serbian Czech	1
Shuswap	1
Songhay	1
Standard Moroccan Tamazight	1
Sundanese	1
Tibetan	1
Uighur	1
Venda	1
Venetian	1
Vlaams	1
Western Frisian	1
Zulu	1