MULTI-PROJECT LOCALIZATION TRANSLATION MEMORY SYSTEM

Chinedu Eze
Student number: 38186

Master of Science Thesis
Supervisor: Ivan Porres
Åbo Akademi University
Faculty of Science and Engineering
Embedded Systems Laboratory
May 2019

ABSTRACT

This thesis takes a general look at localization both as a concept and as a process. As a concept, it tries to give a general description, including how it is seen from academic sources and from companies that develop localized products.

As a process, this thesis suggests a way of conceptualizing it as a flow of smaller sub-process, which can be duplicated, rearranged and made to run in parallel if needed. This concept has the potential to make the entire process easier to understand and reason about and easier to implement in practice.

Finally, this thesis gives a description for a translation memory system. Translation is one of the most important sub-processes of localization. The translation memory system is one of the many tools that are used to make it more time and cost efficient. The system described in this thesis works for a situation that deals with multiple projects, both related and unrelated.

Keywords: localization, internationalization, translation, software process, translation memory

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Terms such as localization and internationalization are used interchangeably in business literature, although they do not mean the same thing and organizations might use different definitions [1]. This thesis attempts to give a clearer definition and explanation of the concept, together with other important sub-concepts and process involved in it.

This thesis also looks at localization as a process and how it is normally done in general. It goes on to list some of the challenges affecting it and suggests some solutions for them. One of the biggest problems facing localization as a process is that in most companies the processes are obscured and mostly not well documented or even documented at all. To mitigate this, a paradigm is suggested as a way to reason about and visualize localization in order to make it easier to understand and document, increase the speed of the process and make it easier to improve on.

Although localization also applies to other areas, such as documents and hardware, this thesis focuses mainly on software localization. The broader localization presents a scope too large for this thesis.

Finally, a setup for a translation memory system is proposed. The system has the capacity to cater for situations with multiple projects, providing a way to access and query for text segments from each one.

2 LOCALIZATION

The definition given by Localization Industry Standards Association (LISA) is

> Localization involves the adaptation of any aspect of a product or service that is needed for a product to be sold or used in another market. [2]

Although this is widely accepted as the standard definition, it is common to see different definitions depending on the material and context.

Sometimes referred to as *L10N* due to the number of characters between the L and N in the word, localization is also defined as the process of preparing and releasing a piece of software for a specific locale [3, 4]. It involves making the software usable by people of different environments and cultures - locales. It is a process with the goal of making a piece of software linguistically and culturally appropriate for members of a specific locale. It is the adaptation of a software product for a particular locale [1, 5]. It involves implementing the adaptation of a localized software into different locales by translation of texts, and applying region-specific configuration. It involves both translation and adaptation in order to make a piece of software conform to the language, culture, and legal requirements of a specific locale [3].

Localization leads to products that can cross cultural and language barriers [6]. It has to do with tailoring a product to a specific local market. The overall localization task comprises translation, project management and multimedia files adaptation. The process plays an important role in choosing the appropriate software development process for a software project.

Localization entails both a process and the accompanying technologies that make it possible. As a process, localization involves other processes such as translation of textual and non-textual materials into other languages, taking into account factors such as the specifics of the locale of the target audiences and their various differences in convections such as numeric values, colours, etc. The localization of a product is not just about translating the texts in the user interfaces, although it is in fact a central part

of it. Beyond that, it also entails other important activities such as adapting texts and visuals to the store listings, support channels, prices, payment methods, release notes, videos, audios and other marketing materials [7].

2.1 Locale

Locale is a term that is used to refer to a specific combination of a variety or "flavour" of a language together with a set of "cultural preferences" [1]. Although not standard, a locale is usually identified by a combination of the language and the name of the country with a set of cultural preferences. These sets of preferences are normally identified by the names of their country or location to form a unique identification for the locale. For example, in operating systems such Microsoft Windows and Ubuntu, the English language combines with various countries to form various English locales such as English (Australia), English (Canada), English (United States), English (United Kingdom), etc.

It is a set of parameters that enables the identification of a user's language, country and the associated preferences [8]. A language and geographical region, together with the cultural implications associated with them, combine to give an estimate of how the user is able to interact with the intended system.

The "language + country name" combo is just for identification, as a locale goes well beyond language and location. It identifies a group of people who share a language, a system of writing and other properties. A locale may comprise any thing from a region to an entire country [5]. While the name of a country plays a role in identifying a locale, it is not tied to a particular location. For example, an American who travels to Australia would prefer a user interface in the English (United States) locale. If there happens to be a large enough group of people in that category, there would be a need to produce software with a user interface that has the "English (United States)" locale option available, even though they would be used in Australia. Another fairly commons example is countries with foreign official languages. For example, there are two official languages in Cameroun, English and French. Hence, the two most common locales there are the various varieties of English and French locales.

Naturally, one may assume that having a locale, which comprises a language and country code, means that the language is an official or at least a recognized language in the territory or country referred to by the code. In reality, this is not always the case.

For example, there is a "en-IL" locale code, which represents the "English (Israel)" locale. However, although English is spoken in that country and understood by a good portion of the population, it is neither official nor even a recognized language [9, 10].

2.2 Why Localization

The international market is an important factor for most companies. Most software products are built to target both local and international audiences. In fact, some believe that many software manufacturers, in the U.S., for example, derive a large percentage of their revenues from international sales. This is especially true for software companies, as their products mostly comprises localizable components. In order to make the software accessible to its intended international audiences, it has to be localized [6, 4].

Generally, users prefer software products that are localized into a language in which they are most comfortable. In most cases, this is the most spoken language in their local community, which is often the mother tongue of most of its inhabitants. One major reason for this preference is that it increases the chance that they understand the product fully and will be more skilled in using it if they interact with it in a language in which they are most skilful. It also leads to fewer mistakes than with a second language. Companies not only want to have their products in the user's language, they want it in the particular dialect that each target user is used to [4, 1].

Users are also more comfortable and more likely to use a product if it's been adapted to their cultures. It makes it feel more like a local product even if it's foreign. Localization is therefore necessary, not only to make the software usable but also more appealing to the users.

Nowadays, most companies maintain an online presence through websites. If a company intends to reach a wider range of audiences with their product, localization is a necessity. It is a requirement for a global market presence [4, 11]. A successful localization makes a software program familiar and popular among people from various backgrounds just as it is in its home country. Localization takes the culture and dialect of the target users into consideration [12].

It is however not always the case that a company has the same message for all their target locales. There are situations where a company employs a different market strategy for a particular country or locale. In such situation, a different message might need to be presented to users in those locales. For example, as part of a campaign,

certain products may be on sale for the audience from a particular locale. The company may therefore want to display a large localized banner on the front page for that locale on their site, which would not be available on the same page for the other locales.

Another reason for localization is that sometimes it is necessary in order to fulfil government imposed requirements or comply with local regulations. For example, labelling instructions in an accepted local language is a requirement for selling certain products in the EU countries [6, 12].

2.3 Localization Approaches

Over the years the way localization is done has evolved. Years ago companies carried out localization by employing in-house teams and *language engineers*. The companies' offices in other locations take care of the localization efforts for the local users in those places [6].

There are a few approaches to carrying out localization depending on when the translated texts are added to the software [4]. The choice of the appropriate approach largely depends on the type of software being developed and the nature of the software development process. The first approach, being the oldest, is most likely no longer used in translation projects. The following is a brief explanations of the most common approaches:

2.3.1 Compile Time: code + language texts

This is one of the oldest approaches used. In this approach, the text and other non-textual materials to be translated, and the software source code are inseparable. Hence, in order to produce an adaptation for a specific locale, the translated materials for the target locale need to be added to the source code and a compilation carried out, producing a localized version of the program. Microsoft and Oracle used to employ this approach [6]. When the software is ready, the code is simply handed over to the team that takes care of localization. Hence, translation can only begin after the software development has ended, making it more suitable for software projects that use the waterfall software development model.

There are a few obvious disadvantages to this approach. For example, in order to make a correction or add an update to the software, the entire process has to be repeated for every supported locale. Tests need to be carried out all over and locale tests may be

necessary to check for locale specific errors. A change in the user interface would also effect a change in the source code and vice versa. Request for additional languages and version management become increasingly complicated [6]. Of the three, this approach is the most time and resource consuming.

Scalability is also a big issue with this approach. It is not possible to work on other locales' adaptation until the original version is ready. More so, work on translations can not begin until the software code is ready.

Yet another disadvantage of this approach is the lack of separation of concern. Since the translated texts are part of the source code, it means that the translators would have to work with source codes and developers with translated texts. Hence, people are made to handle things that are not in their areas of expertise. A repercussion for such a system is that translators who are likely not familiar with how source code and compilation work, are most likely to introduce bugs into the code base since the translated texts are tightly coupled with the source.

According to [6], companies still manage to separate software development from its localization by delegating the job of localization to their international offices. Of course, this would be impossible if the company doesn't have one. Earlier, localization departments were part of big companies. These departments could grow so large that they become too difficult to manage. Learning curves for new employees were also a problem. In addition to their required linguistics skills, new employees needed to have the technical skills required to understand the software code in order to make good translations.

There are situations where software programs are developed with only one language in mind. Having no intention to localize the program for other locales, the developers hard-code its texts into the source files. Because such approaches do not take localization into account they are not considered a compile time localization. A compile time localized software does not necessarily mean that the texts to translate are embedded in the actual source code files; though this may sometimes be the case. In fact, texts for translation are mostly stored into separate files in formats such as *xml* just like in other approaches. However, these files need to be present at compile time. Otherwise the code may not compile successfully. Also, once the compilation is done, a new localized version is produced and the translated texts or files can not be changed after. Changing them leads to software failure. In fact, if the software is compiled down into a single executable file format such as the *exe* file format, the

translated texts or files are embedded in that file. It's normally impossible to extract them without the use of specialized tools.

2.3.2 Link Time: code + language packs

In this approach the program is compiled apart from the locale specific materials such as translated text, images, media files, etc. After compilation the software program file produced is the same for every locale. A combination of the program and a locale-specific set of materials makes a software package for that locale. That is, the compiled program is the same for every target locale. The only differences are the locale-specific resource files.

This approach is more appropriate for applications that require less frequent switching of their user interface language, as a restart is usually necessary in order to switch from one locale to another. An example of a piece of software program for which this approach applies is the *Windows 7 Operating System*. When a copy of the software is installed, the default locale files are added as well. If in the future a different locale is required, the resource files for that locale, such as translation files, sometimes known as *language packs*, are downloaded and copied into the appropriate file system locations. To use the new locale, such programs require a restart, usually after selecting the new language.

Such software programs are normally distributed already bundled with specific (default) locale translation files. After download and installation, the user can later change the locale by downloading the files for another one. In most cases, the required files are downloaded by the software automatically when a user selects a new language. Sometimes, the program even restarts automatically when the locale files are ready.

In some cases, this approach is necessary. Language translations and localization resource files are not the only components of a software program that change from one locale to another. Sometimes, changing to a different locale may mean loading up or changing a software module. An example of a possible module change is changing the locale of a text editor with a spell checker feature. Since the spell checker module is a software program and not just a *locale resource file*, the program may need to restart in order to load and use the new module.

An obvious advantage of this approach is that the source code development and the translation of texts can be done simultaneously. This easily translates to shorter project execution time and an increased productivity. This also means that both the

software development and translation endeavours can be treated as different projects, hence, they can both employ different processes and can be handled by different project managers.

The approach makes it easy to outsource the localization project, allowing the software company to concentrate only on the development of the source code of the application [6].

2.3.3 Runtime: code + language runtime

It's basically the same as the link time approach except that a restart is not required. It's the most suitable for situations that require a quick and easy switch between locales. Due to enabling the ability to switch languages or locales without the need for a restart, it's the most popular approach used for the localization of web sites. In fact, most web application frameworks incorporate an easy system, with accompanying libraries, for making the runtime approach implementation possible. For example, the Django web framework has the "Django translation package" from the Django *utils library* for this purpose [13, 14, 15].

The main appeal of this approach to internet websites is that a user can easily switch from one language to another without changing their location on the site. This is sometimes necessary. For example, if a Spanish user receives a link to a help page. On following the link, they discover that the page is in English. All they have to do is change the language of the site. The page reloads and they are able to read the instructions in their preferred language. If the page throws them to the home page after switching language, then they would need to try to find the help page all over again.

With this approach, texts that need translation and localization are isolated and separated out into locale-specific files called *resource files*, and replaced with placeholders. At runtime, the correct language resource files are loaded by the browser, replacing the appropriate placeholders [13]. This process takes place when the software is loaded with the default language selected or when a different language is select by the user. This is exactly how angular translate works [16]. It could also work in such a way that the browser sends the required language parameter to the server. The server puts together the components of the web page with the requested language and sends back a response. This is how server side rendered applications frameworks like *nextjs* accomplish localization [17].

Placeholders are not only for texts to be translated. They are also used for cultural

and locale-sensitive texts such as dates, time, numbers, e.t.c.

This approach and the previous one can not be done without the internationalization step. The separation of the texts for translation is a crucial part of the internationalization step prior to packaging the software.

2.3.4 Online Dynamic Translation

This approach employs the use of translation services, such as Google Website Translator, to dynamically translate the texts on a website when it's loaded. It should be noted, however, that according to the Google Translate website, this service is now discontinued [18].

A plugin supplied by the translation service is installed in the website. The site visitors are presented with a few language choices. When a user selects a language, the plugin scans the page, extracting the appropriate texts for translation. These texts are sent to the translation service. The service sends back a response with the translated texts, replacing the original ones.

This approach has a few advantages. For example, it makes it easy to localize a piece of software without prior planning. The translation service plugin automatically extracts and translates the pages without additional efforts from the developers. It also makes it possible to localize a site into multiple languages in minimal time span. The previously mentioned Google Translator Service supports up to 90 languages [13]. Using the service, a website can be made accessible to multiple locales with relative ease.

On the down side though, localization is not only about translating a user interface. Using machine assisted methods could have some unintended consequences. For example, a page could be translated to Spanish while the dates are left in American English locale format. Users may find this unprofessional. Also, some texts that do not need translation may end up translated. For example, if as part of their marketing strategy, a company has a product named "fly easy", the translation plugin may mindlessly translate it into the new language, which may not be intended.

2.4 Localization Tools

The process of internationalization and localization is inherently complicated and involves repetitive tasks that, if done exclusive by humans, are very error prone. Hence,

the necessity of software tools [4, 19, 11]. These are tools developed in order to make the translator's job easier, faster and more efficient [20].

In the early days of software localization, companies had to create custom proprietary tools to support their localization efforts [6]. These ranged from tools that enable easy extraction of texts from source code for translation, to tools that assisted translators in keeping track of translated texts. The tools are tailored to their codebase and had unique file formats, mostly xml-based. These tools later gave rise to commercially produced alternatives that were suited for a wider range of companies and their various processes. They include tools for terminology management, translation memories, machine translation, localization workflow, project management systems e.t.c.

Translation Memory (TM) is one of the most useful tools in the localization process of most companies. In its simplest form, it's a dictionary of translated texts, with contexts from a vocabulary.

Software user interface localization tools also exist to make the extraction and translation of user interfaces easier and automatic. There are also tools for testing translated texts in user interfaces.

2.5 Localization Personnel

In order to ensure success in localization of big projects a few specialists are normally involved. These include localization project managers, translators, programmers, localization engineers and quality assurance engineers [4].

2.6 Localization Vendors

Localization vendors are companies that specialize in services associated with localization, such as translation services and translation project management.

2.7 Localization Service Providers

Companies have long realised that is it more profitable to outsource localization efforts to companies that specialize in such services [6]. Localization companies take care of various aspects of localization such as translation, *localization engineering* and localization project management.

3 INTERNATIONALIZATION

Also referred to as "I18N" due to the number of characters between the 'I' and 'N' in its spelling, internationalization is the process of adapting a product to the cultural identity of its users [3]. It is an engineering process that comes before localization. Its main objective is to reduce the cost of localization and translation, and make it more efficient and less error prone [1, 4]. Internationalization adapts and prepares software in order to support or enable localization. This process has become a necessity as the simultaneous release of all locale versions of software products has become a requirement for most companies. It reduces the challenges involved in localization. In order to be able to carry out localization, the software has be internationalized [6, 11].

The cost of localization could be so high it causes a company to narrow down its target market [12]. Internationalization reduces this cost, increases efficiency and reduces the time it takes to execute a localization project. The process of internationalization is done once, after which the product can easily be adapted to support other languages and locales.

The underlying principle of internationalization is the separation of all *culturally and linguistically sensitive* components from the core of the source code into special files, with special formats, in a process referred to as *"leveraging"* [4, 3, 1]. Some of the texts that need special treatments are Date and time formats, currency format, language character code sets for texts display, names and titles, phone numbers, addresses, international post codes, weights and measures, e.t.c. For example, a double-digit encoding standard like *utf8* is used to encoding the translatable texts instead of single-digit encoders such as *ASCII*. With this little but important change, the encoded texts can be translated into large varieties of non-English languages including Oriental languages such as Chinese [1].

Texts are not the only components that are linguistically and culturally sensitive. Others include media files such as images, audios and videos. In some cases even colours are to be taken into consideration [4].

Internationalization efforts are normally mostly focused on the architecture and design of a software product in order to make it easily adaptable to multiple languages and locales without the need to redesign the entire program [1]. The end result is a product that is in a form that can easily be adapted to other locales using special tools. Apps and web sites that were not built with localization in mind are very difficult to localize [7].

3.0.1 internationalization and localization

Together, internationalization and localization form a process of adapting a piece of software for non-native users in a cost effective and efficient way.

Some see internationalization as the accessibility of a product to multiple countries and cultures rather than as an engineering process [11]. Based on this view, a program that is available in more than one language is an internationalized program. Localization is therefore taking an internationalized software and adapting it to different regions and languages.

3.0.2 Internationalization and Translation

Translation is a major influencer of the internationalization, and by extension the localization process. It is during internationalization that texts that need translation are extracted and separated into language-specific files called resource files. Thus, internationalization prepares a software product for efficient localization [13, 5]. Common formats for storing translation resource files include exe, Resx, WPF(Xaml, Baml, dll/exe), *json* and *csv*.

Things to consider when implementing internationalization include date and time formatting, currency handling, language verbosity, handling of non-English characters such as Korean, Chinese and Japanese, character encoding, etc. Any type of content that might change based on the locale, such as text, images or other media files, must be externalized from the core application and put into external resource files. The process is known as Content Externalization. No string should be *hard coded*. For a smooth translation implementation experience, strings should be externalized into external files which can be applied at runtime according to specific languages and locales.

A common mistake with internationalization is to hard-code error messages in the original (source) language such as English. This can lead to confusion and frustration

when the software, running in a different language throws out errors written partially or entirely in a different language, rendering the messages useless to the user. To avoid this type of situation all texts that user might see (including error messages) must be separated out and subsequently translated. Also, care must be taken to avoid the crashing of the application due to improper handling of translated resources. Certain texts or variables like the system variables should therefore not be translated [21].

4 LOCALIZATION AND TRANSLATION

While localization basically means taking a product and "tailoring" it to specific local markets, translation means converting a string of texts to its equivalent in another language. It is an essential part of the overall localization process. The translation process involves linguistics as well as cultural transfer, and the communication of intention or function of target texts [1, 5].

A large portion of localization efforts is focused on the translation of software interfaces and their accompanying documents [6]. Bad translation can do a heavy damage to the reputation of a software company and its products [1, 11]. Effects range from minor irritation, to making a product unusable. It is therefore essential to ensure accurate transfer of translation texts from source to target languages. Users want to browse a website or follow instructions for a piece of software in their own language. They expect clear, unambiguous and easy to understand information. They also expect to not be offended by the language, images, colours, and so on [5]. The quality of translation is a representation of the image of the company. Good documentation is seen by most users as a sign of quality. Further more, inaccuracies or ambiguity may introduce legal liabilities [12].

An internal review process should be put in place to check translations for accuracy, quality and appropriateness. The reviewers should include in-house native speakers and professional representatives from the target country or locale [12].

Software components that need translation in a website include texts, pictures, and multimedia files such as audios and videos. Others are document files, and in some cases, dynamic contents such as posts on forums and chat messages [5]. Depending on the need of the software or websites, not everything or every asset types are translated.

Many materials interchange the usage of localization and translation. However, the Localization Industry Standards Association (LISA), gives a distinction between them. According to LISA, the main difference is that while translation is mostly focused on texts, localization goes beyond to include the adaptation of other aspects of the

software into not just the target language, but also their culture and norms - their locale. These aspects include colours, icons, and even layouts [2, 22].

4.1 Translation outsourcing

Translation outsourcing is a way to outsource the responsibility for international satisfaction [12]. Some companies that offer localization platforms and applications also offer translation services as well, or maintain liaisons with translation companies and with independent translators. Hence, they offer customers packages that include the management of translation projects as well as the actual translation of their product contents. An example of such company is Transifex [23].

Another approach to outsourcing translation is by crowdsourcing. Crowdsourced translation is more common among free and open source projects such as Telegram, but also used for commercial products by other companies, such as Netflix. Telegram refers to their localization process as “community-driven”. As described on their localization platform site, the process is simple and straight forward. Any user of the app can suggest a translation for any segment of texts on the user interface in any of their apps. They provide users with a style guide and an online localization platform accessible through a web browser, with which the users manage their translation efforts. Testing the translated texts to see how it works in real-life is done directly in the target app. Users are also able to view translations suggested by other users and, vote for the ones they feel is best. In the end, the final decision is reviewed by experts, who may as well be from the community of users or hired directly by Telegram. Although any user can translate any text, experts on the other hand have to apply for the position and be recruited by the company [24, 25].

The approach of outsourcing translation is not always as successful as in the case with Telegram. Netflix had to shut down their crowdsource translation project called “Hermes”. They stated that the main reason for the shutdown was because they became overwhelmed by the number of applicants. Now they rely on a few major localization and translation companies world wide for their translation and subtitling projects [26, 27].

5 LOCALIZATION AS A PROCESS

As a process, localization is comprised of every action carried out in order to produce localized versions of a software application. These actions range from designing the software, the release of updates to its user interface and how the updates are applied. The importance of established localization process is often over-looked leading to set-backs such as delayed software releases. In order to avoid bugs, localization should not be an afterthought but planned upfront from the beginning of the software project [7].

The absence of an official localization process in a software company can lead to problems, such as missed delivery deadlines [12]. Also, a localization process that is not properly designed could make the translation process a major bottleneck for the entire project. One way to mitigate bottlenecks in a localization process is to make as many phases of the process as possible parallel.

Managing translation done only by in-house translation is a daunting task. As is often the case, translators may not be available when needed to make an update to the software, as translation is normally not their primary jobs. This is one of the main reasons why most translation jobs are outsourced. However, in bigger companies with a huge number of projects, like Google or Microsoft, in-house translation is often a better choice than outsourcing.

The first step in the development of a localization process is internationalization. Sometimes a software program is only available in its original language, which, in most cases is English. If the software is not previously internationalized, a rewrite may be necessary. The main outcome of this step is an internationalized software, ready to be localized into the various supported locales for the project.

In organizations where multiple dissimilar projects are developed, the localization process may differ for each project. This is because the type of project or type of software being developed has great influence on the localization process. Another major influencer of the localization process is the overall software development process itself. Localization is not an isolated process. It needs to be part of the bigger software

development process.

Multiple projects also generally means multiple teams. In modern software development practice, each team determines the process it employs for its projects. In situations where there is need for information or data transfer between teams, this can pose a challenge. Localization tools often mitigate these problems.

5.1 Generic Localization Workflow

One good way to reason about the localization process as a whole is to visualize it as comprising a set of individual phases. Some phases might be repeated and others might be carried out in parallel. Although the localization process as a whole may differ between teams and projects, they all generally contain these basic phases in one form or another. The phases are just a way to visualize the localization process and to reason about it.

Since localization processes are generally unique to each team and the software they are developing, it is normally better to have the localization team create the localization phases together with the development team. After its creation, each individual phases should be be well reviewed and refined. After it is approved by all involved in the project, it should be documented, along with a brief visualization of how everything is connected. The documentation serves as a reference for everyone involved in the project, and also for future team members.

Most software development teams have documentations for their development process in general. In most cases, the localization process is only a part of the general software development process. Although this approach might suffice in smaller projects, in bigger ones or in organizations with multiple projects, it is a good idea to have a separate documentation for their localization process.

5.1.1 Basic Phases

Phase here refers to one or more activities that, together form a part of the localization process.

Text segments creation

This is the phase that leads to the addition of one or more new text segments into the project. It mostly happens when a new feature is being developed. It can also take place when a text segment is updated - corrected, changed or extended. For most projects, this is a recurring process that takes place from the beginning of the project until after deployment.

Most times, new text segments are added by developers, but this is not always the case. For example, in order to improve the speed of software releases, Memrise had to modify their process such that the texts are created by the team that does the specifications [7]. This way, although the developers later add the new texts into the source code, they have already been created and are already being translated by that time. Also, although it is most times the case that new text segments are added to the project through the source code, it is not always so. That is to say, adding texts to the source code does not always signify the creation of new texts. Sometimes, new texts are actually added directly into the translation memory system. The translators and developers, take the texts from there.

Quality Assurance (QA) Approval

At this stage the QA team approves or disapproves added text segments. In most teams, the project has to be put in a freeze state while the QA team performs tests on the program. During a freeze state, new texts may not be added to the codebase. The use of freeze stage could lead to bottlenecks in some projects. Bottleneck problems are most eminent in projects that use the agile system of software development. This is mostly because in agile workflows, new features are added relatively more frequently, leading to a fast paced text creation. To mitigate this, it is common to accumulate the new text segments and do a QA approval at certain designated stages of the process. However, with this approach it is no longer an agile process, or at least, it loses some of its advantages.

In most projects, the QA approval phase is usually not a single, one-time activity. Two of the most common stages for this phase is after new texts are added to the project and, when translations are available. Carrying out QA tests processes after new tests are added to the project is more suitable for projects where the texts are first added directly to the source codes. Translations are then carried out only after the added texts have been approved by the QA team. A new QA testing procedure is also done after

translations are made available by the translators. In some projects, it is also possible to at least do the first QA testing before they are added to the source code, or even before any code is written. This is usually the case where new texts are not added directly to the source code but first to the specification document or to a translation memory system. In this scenario, at least one more QA approval is needed when the texts, together with their translations, are added to the source code to produce the final program.

One strategy to reduce the number of localization related QA testing that may be necessary is to give the translators a means to deploy and test the new translations themselves. An example of where this strategy is most useful is in cases where translators need to make sure the translated texts fit properly into their allotted space on the user interface.

Translation of text segments

This is one of the most important steps in the entire localization process. During this stage the new texts are translated by the various translators working on the project. Usually the translation is outsourced to a translation or localization company. This is yet another stage that is carried out after the texts of the project are put in a freeze state. In some organizations, the translations are done in-house. After this phase, there is usually a repeat of the quality assurance (QA) tests before the project is deemed ready.

Integration of translated texts

This normally occurs after the translations are received from the translators. It is better that the translations are vetted by the QA before their final integration into the project, although this is not always the case. This stage is also sometimes done automatically. Some translation platforms provide a means to give access to translated texts once they are ready, for example through a *web api*.

5.1.2 The freeze state

This is a state in which no new texts are added to projects during software development. It is a common strategy used in different situations. The most common is when texts have been sent off to translators for translation. A good reason for doing this is to make

it easy to track texts for translation. If new texts are available they are normally added to other branches of the source code repository and held there until the translations for the previous ones are ready.

Another good reason for freezing projects during localization is to extract the texts for translation manually. In some organizations the texts to be translated are normally extracted manually with tools such as lingobit localizer [28]. The process is sometimes time consuming and therefore needs to be done only at specific intervals during the lifetime of the software development. Adding new texts would require a redo of the process, hence the freeze. After the texts are extracted, copies are made and sent off to the various translators working on the project.

Although useful, freezing often leads to delays in the software development process, as during this stage no new texts could be added to the source. This does not however mean that development stops altogether. It continues but mostly in another branch of the git repository. Hence, by its very nature could lead to a merge nightmare in the future. It is also possible to lose some texts during a git merge after a freeze state is lifted [7]. Further, it is difficult to make corrections to the code during this stage. If the new changes have texts that require translation, they have to be put on hold, or held in another branch. It is possible to generate a tracking id manually through the use of utility programs, such as the *msbuild command line* utility in CSharp projects.

A good strategy to mitigate the problems associated with the freeze stage is to provide a means for the easy flow of data from translators to the source code and vice versa. Some translation or localization companies provide tools that make this possible.

5.2 Putting it all together - a basic localization process

The following is a description of a generic localization process. First, the texts are added to the project through the source code, the translation memory or the specification document. The texts to be translated are extracted and sent off to the translators. The project is put on a text-freeze state. When translations are ready, they are integrated into the project. A quality assurance test process is carried out on the project. If corrections are needed, the translators are notified. If accepted, a version of the software is packaged and released.

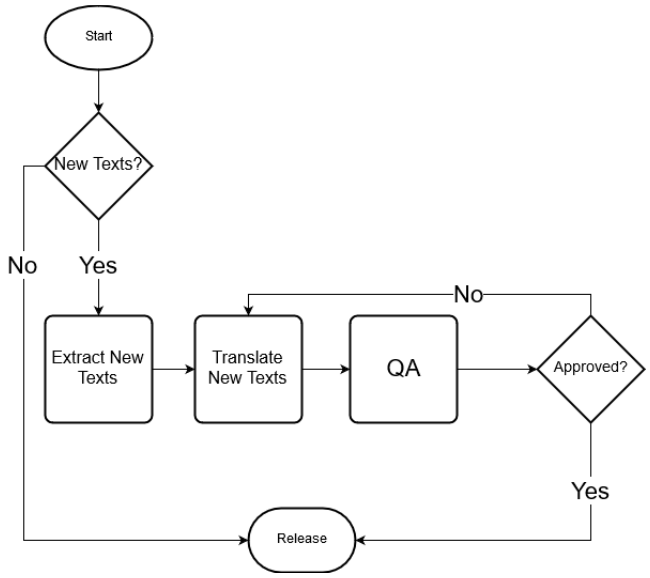

Figure 5.1: A basic localization process workflow

5.3 Advantages of Phase Reasoning Approach

Below are some advantages for applying this approach to localization projects. It should be noted that these advantages are not peculiar to the phase reasoning approach.

5.3.1 It makes it easier to visualize the localization process

The first obvious advantage of this approach is that it makes it easy to visualize the entire localization process. In bigger, more complex projects it may be relatively difficult to reason about the activities that make up localization. It becomes difficult to ascertain exactly when a project begins and ends. Thinking of it as a series of phases serves to mitigate this problem, by giving a name and clarification to each sub process that constitute the whole. Thus, it becomes easier to reason about it.

5.3.2 Makes it easier to assign individuals to tasks

Another important advantage of this approach is that it becomes easier to assign people to tasks in the localization process. Different people could be put in charge of different phases. For example, while the creation of new texts is mostly done by developers, adding translated texts back into the source code is not an activity suited for developers. Also, after the texts are translated, it is the QA team that is generally responsible for

its verification. The system makes it easy to see what areas of the process needs to be outsourced to a localization company.

5.3.3 Makes it easy to choose the right tools

The clear definition of a localization phase makes it easy to identify the appropriate tools that are needed in order to complete it. Sometimes, it might be that a customized in-house tool is needed. In that case, a good documentation for a localization phase would serve as the main reference for creating its software requirement specification. It also makes it easier to decide whether to develop an in-house tool or to acquire a commercial product.

5.3.4 Easier to identify bottlenecks in the entire process

Breaking up the localization process into steps or phases makes it easy to identify bottlenecks in the process. Once identified, the entire localization process can be improved by making little improvements on the individual phases that caused the bottlenecks. For example, the release of the software might be delayed due to late arrival of translated texts. In such a situation, one possible solution might be to send the texts for translation incrementally to the translators. Another possibility in such a scenario is to split the entire bulk of texts into parts and assign them to multiple translators simultaneously.

5.3.5 Makes it easy to improve the entire localization process.

The entire localization process is an aggregation of the phases it comprises. An improvement on any number of phases results in an overall improvement on the overall process. Once the process is visualized as a series of phases, it becomes easy to see which areas, when improved, would lead to an overall improvement on the entire process. For example, switching tools used for keeping track of translated text segments could make the process faster. Another way to improve the process is to carry out some of the processes in parallel. It is also possible to effect an improvement by re-assigning tasks in the project.

6 CHALLENGES WITH LOCALIZATION

Based on their nature, localization problems are generally of two types: technical and process [7]. Technical challenges with localization are mostly those that involve tools and how they are used to implement localization. Process challenges are those that have to do with the sub-processes that make up localization and the entire localization process as a whole.

This section discusses some of the challenges that face localization as a whole. They are issues to consider when thinking about producing a software product in a different locale than the original.

6.0.1 Cost

Cost is one of the biggest challenges to deal with. Issues with cost extend through every area of localization including translation, tools, internationalization, etc. Compounding the problem is the fact that it is not always easy to make the decision about what platform to use and how to choose packages so as to minimize costs.

6.0.2 Page Information Flow

The flow of information, as in how components such as texts and images are laid out on the pages of a user interface, is not always predictable. If working on only Latin text-based languages such as English, this is not usually a problem, but with situations that require handling of a larger language base, there is a new set of challenges to deal with.

Texts are not the only problem to deal with. Some languages, such as Arabic, in which texts flow from right to left also require user interfaces that reflect the same pattern. This means that other aspects of the user interface pages have to be flipped as well [22]. These include components such as the overall page layout, icons and images that indicate direction. For example, a control panel which was on the left in

the English version of the application, might need to be moved to the right. Usually it is not as straight forward as just flipping all user interface components such as panels, images and icons. Care needs to be taken to not flip images that may not need to be flipped. These include universally recognized images such as an image or icon for a phone, indicating a phone number. Also affected are icons and images that have inherent meanings in their default states, such as the play button of a media player. These may loose their meanings or otherwise not make sense when flipped.

6.0.3 Text Sizes and Truncation

This is a very common problem that needs to be dealt with in localization. It has to do with the size and number of characters that is needed to transfer the meaning in a statement or phrase from one locale to another. As previously stated, most companies strive for uniformity of the messages and ideas that they pass to their users. For example, on the user interface of a program, the phrase "Delete Project" has to mean the exact same thing irrespective of language. A misinterpretation could lead to unintended consequences. In such cases where this is very important, the size of the texts and the space they occupy could become a problem.

Text sizes vary significantly from language to language. This can be a problem for the user interface. It is known that an equivalent phrase in English would take as much as twice or more spaces in some other languages, such as French or German [13]. Text is mainly affected by factors such as number of characters in translated text, the language's characters' width and height. Such problems are normally manifested as unintended truncation of texts or overflowing of texts into other areas and new lines. The localization and development teams, together with the translators, have to look for a way to solve the problem together.

Possible solutions could come from a combination of solutions from any of those teams. From the development team, it is possible to program the way the texts are truncated in such a way that they still carry their intended meanings. They could also make the container interface more flexible so that it expands to show longer texts. If texts are to be truncated, care should be taken to make sure that the meaning is preserved and that the proper part of the texts is truncated, such as the middle or either of the extremes.

The translators, with the help of a translation memory program, could look for other fitting phrases that would convey a meaning that is close enough to the original.

The localization manager might decide that the user interface looks slightly different from the original in order that it might accommodate more or less information.

6.1 Challenges with Translation

The distinction between localization, as seen by some, and translation is not always clear. In this section the challenges discussed are mainly those that have to do with the transfer of materials from source languages or locales to their equivalents in target locales.

6.1.1 Accuracy

In terms of localization, the accuracy of translations depend on a few factors. One of the most important factors is the message that the organisation intends to pass across to their target audience. For a user interface label, for example, it means the clarity of the function of the item. Basically, they have to be sure that a user understands what a button does when they read its label.

Accuracy is sometimes difficult to achieve in software products. A typical problematic situation is when certain words are not available in the target language. For example, if translated directly, the word for "websites" in German, "Web seiten" does not mean the same thing [5].

The accuracy of a translation is also affected by the terminology of the community of professionals in the target locality. While a translated text might convey the "correct meaning" linguistically, there are times when that meaning is not quite acceptable within the community of professionals. An example case is discussed in [12], where a dental company translated a piece of dental equipment using a word considered crude by the dental community of the target country.

Issues with Dialect

Issues with dialects play an important role in translation accuracy. The closer the language of presentation is to the spoken dialect of the users the more confident they feel about how well they understand it, and hence, the more likely they are to adopt it for use. A good illustrative example is mentioned in [1]. The Spanish dialect spoken

in Spain and South America have important differences [12]. These differences, although sometimes subtle and might seem unimportant, need to be taken care of by way of proper localization. Failure to do so might lead to undesired consequences, ranging from minor user inconvenience to total dislike from repulse. An example case is discussed in [1]. A notification is displayed by an HP printer as "Estado civil: Activo" which suggests that the printer is in an active matrimonial relationship. This error in translation is likely due to a direct translation of the the English phrase, "Status: Active", which means that the printing job in the queue is actively being printed at the moment.

Uniformity and consistency of messages are not always a requirement for software translation. Companies may have different goals and targets for different localized versions of their software products [5]. For example, if a US company just established in Spain and has an ongoing sales campaign for that country, they may want to have that information only for the Spanish locales of their website. This will affect their localization effort. A company's local market strategy also plays an important role in how they localize their translations. For example, a company may try to project a message like, "easy to use" for a local market and one of "feature complete" for another. Hence, when translations are done, they would try to make it fit either direction depending on the market the message is meant for.

6.1.2 Consistency Across Projects

When a company has multiple projects or applications, consistency across those projects become a necessity. As Telegram stated on their localization platform website, "The same things need to have the same names everywhere." Consistency also implies that the same word means the same action across multiple applications and software [24].

It could be confusing and sometimes catastrophic for users, if the same word or phrase does not always mean the same thing in every part of an app or similar apps from the same company. For example, if the phrase "Delete project" means that the project is removed from the user's file system, but the process can be reversed and the project recovered later, the same phrase can not therefore be used in another part of the same app to describe a variant situation where the project is lost without the possibility to undo the action. Consistency could be very crucial in certain situations, but depending on the number of projects this is not always easy to achieve.

6.2 Challenges with Internationalization

As previously stated, internationalization has to do with the preparation of a piece of software in such a way that it can be localized into the various supported locales. This creates problems that are not easy to solve, both for the target software and its design, and also for the tools used for its development. This section discusses a few of such problems and possible ways to handle them.

6.2.1 Numbers and how they affect words - singular and plural forms

In English language, the word, "tab" in singular becomes "tabs" in plural no matter the number of tabs in question; be it 2 or a million. In some other languages, this is not the case; the word takes different forms depending on the exact number of tabs [29]. For example in Czech, the word for tab, "panel" takes the form "panely" for 2, 3, or 4 tabs, and "panelů" for more. It becomes a challenge to make the software handle this situation properly. The more languages to support the more complicated the problem is.

6.2.2 Issues with Character Sets and Character Encoding

Character sets are groups of characters for specific purposes and character encodings are a set of numbers that map to the characters. These numbers are used to locate the appropriate glyph for displaying a character. If the character encoding of a page is not appropriately set, people may not be able to read the contents of the page [30]. In addition, it could also make it difficult to find the page with search engines. This is usually common with web applications and is fairly easy to fix. Using the "utf-8" character encoding, which is the de facto for most situations, both for saving and serving the documents with the appropriate headers over the network is usually enough to solve this problem [31].

6.2.3 Issues with Dates and Time zones

This is one of the most difficult problems to solve in terms of internationalization of software programs, especially user-facing applications. It gets more complicated when some of the contents are produced by the users, e.g. forum comments. For example,

the way the date is stored may not align with how it's displayed back to the users on the forum pages.

There are many issues that have to be taken into consideration when dealing with dates and time in software [32]. For example, not everyday in a year is 24 hours long, the time zones and the rules they operate with change from time to time, dealing with hours and seconds gain/loss, etc. Although not all these problems affect all software, if the software deals heavily with dates and time, such as conference booking applications, then they have to be taken care of.

In simple cases, the UTC time is enough for saving dates and time [33]. To display it back to any user, the saved time is simply converted to the user time zone. It gets even more complicated when dealing with dates and time for future events. The time zone from which the user made the request that saved the time to the database may differ from the one it affects. For example, a user in London may book a conference room in New York for a meeting in a future date. The system needs to send an invitation to every user that would be in attendance of the meeting, all of whom may be in different time zones. Moreover, there may be daylight saving changes occurring in at least one of the locations of the affected users. This is a fairly typical situation for conference booking software applications.

6.3 Challenges with the Localization Process

One of the main sources of problems with localization is the lack of a clearly documented localization process [7]. New employees do not have a written reference material on how to deal with localization. In such cases, it is easy to do things wrong, as there is no established correct way of doing things.

The software development process employed by the team also play a crucial role in determining how the localization process is designed.

6.4 Challenges with Localization Tools

Modern localization is virtually impossible without the appropriate tools [34]. Although localization tools are essential for the overall localization process of a company, it is not a good idea to design the process around it. The tool may not be available in the future, become unaffordable, or become defunct and no longer maintained. In such

cases, if the process was designed with the tool as the central component, it may become difficult or even impossible to migrate to other tools. This is mainly because most tools rely on proprietary programs and systems.

This section take a quick look at a few problems that come with the use of tools in the process of localization.

6.4.1 Tools are expensive

There are so many tools for working with every aspects of the localization process; from internationalization of the code, to managing the localization project as a whole. One thing that is fairly common is that they are normally expensive. There are open source programs but these are usually not regularly maintained or have too few features.

6.4.2 Difficulty in migrating to other tools

Localization software tools are normally priced based on certain aspects of the projects, such as project size, number of translations, number of projects, etc. Sometimes requirements change and there is need for an upgrade. It is not always financially possible to acquire an upgraded version of the tools. Sometimes it becomes necessary to switch to other tools. This is not always possible. As is the case with most software tools, the team may get locked in to a particular tool or set of tools that they are not able to switch to others in good time.

6.4.3 Difficulty in Developing Customized Tools

Depending on the size and software development requirements of an organisation, there may be need to develop in-house tools to aid in one or more aspects of the localization process. Tools may range from something as small as a simple script, or a full fledged localization application. Most companies that carry out localization must setup some in-house tools one way or another, even if only a tiny single-file script. At Memrise for example, they make use of slack apps for various purposes to help with the process. Slack helps with activities such as sending notifications when translations are ready, triggering test builds by translators when they have made new translations and publishing links for test deployments to the development and testing teams [7].

All of this is done even though they make use of a localization platform known as PhraseApp.

6.4.4 Some tools are too complicated for small projects

Sometimes a project may start out small but over time become too large or complicated to be managed as a small one. Some projects may be small but have a big localization project. One reason for this is if the project, although small, supports multiple locales. In such situations it becomes difficult to choose an appropriate localization platform to manage it.

6.4.5 Choice of Tools is Difficult

There are a lot of localization tools and platforms available, that are relevant to one or many aspects of the localization endeavour, such as project management, translation, localization personnel management, text segments extraction, file management, etc. The commercial tools are generally very expensive and the open source tools are usually complicated or not well supported to rely on.

7 TRANSLATION MEMORY

Also known as *technical glossary* or *product glossary* [19, 12], Translation Memory(TM) technology is a major component of a modern localization process. In its basic form, it is a database of text segments along with their translations [6, 20]. Translation memory tools are normally the main tools in most localization tool sets.

It is a convergence of content management and knowledge management. Documents are split into chunks of text or bits of knowledge items for future reuse. Software texts are not the only items, and user interfaces are not the only places where these are needed. Translation memories are also useful in other components such as user manuals, online help documents, customer support files, websites, etc. [5]. Working with a translation memory in all these areas leads to a uniformity of phrases and knowledge base throughout the various documents and user interfaces, and across projects in organizations.

The translation memory is a very important tool for translators. It automatically offers translations for text segments. The translator can then reuse the proposed translation, adapt it or create an entirely new translation. After a translation is verified, it is stored in the database for future reuse. Apart from being able to extract translated text segments from projects, users are able to add texts into the translation memory manually. Texts to be translated come from various sources in the project, such as graphic user interfaces mark-up source files, items such as dialogs and information messages, help files, documentation, etc.

More functionalities are being added to translation memory systems, for example context aware pre-translation, machine translation and project management capabilities. Generally, the core functionalities of a translation memory are coupled with others forming a localization software package. It is therefore generally not common to find software programs that only have translation memory software functionality.

Most software companies endeavour to release their programs in all supported languages at once. To achieve this the localization process has to be started at the same

time with the software development. Also, after release, most software products are updated at least once a year [6]. Most of these updates build on already established bases, hence the importance of reusability of translated materials.

One of the main ideas behind translation memories is the breaking down of texts into *information elements*, and storing them in a databases in the supported languages. These elements can later be rearranged to form new elements with corresponding meanings in the other languages, creating new *information elements* which are subsequently added back to the database [1]. New texts are only added if they are not already available in the database.

7.1 Translation Memory Workflow

Text segments comprise one or more words, forming single words, phrases, sentences or even entire paragraphs [20]. When a new translation is made, some translation memory software can automatically detect which text segments to store. They accomplish this by following a set of rules like breaking up a paragraph by full stops. The translation memory uses these rules to determine what constitutes text segments.

During translation, the software also breaks down the new (input) texts to be translated following the same rules it used to input the texts for storage. Translation suggestions are automatically offered for text segments with exact matches from the previously translated text data in the database. If no exact matches are found, a fuzzy search is made and translations of similar text segments are offered. Some systems allow the translator to set the percentage for the fuzzy search matches. The translators check to make sure that the translation offered is a good fit and, if not, adjusts it accordingly or replaces it entirely.

When a translation has been made and accepted, the new translated text segments are fed into the translation memory to be stored for later reuse. This is mostly done automatically by the translation memory system.

A very important feature of a localization process implementation is to foster reuse of text fragments [1]. To achieve this, it must be easy to find texts and their equivalents in the supported languages from the database.

There are companies that provide translation memories as services, mostly as part of a localization package. However, being company property, ownership problems might arise down the road in such situations, as pointed out by [12].

The translation memory database must be serviced over a long period of time [5]. This has to do with both technical maintenance, such as backups and software updates, and non-technical maintenance, such as making sure that the terms and languages translations data still reflect the intended meanings. Meanings might change over time and require updates to the data stored in the database. Updating the text database is sometimes achieved automatically by the software when new items are added.

7.2 Advantages of Translation Memories

Although translation memories are not essential for localization, they play a very useful role in the long-term success of it for companies and organizations. Below are a few of their advantages:

7.2.1 Increase efficiency

Translation memories increase the efficiency of translators. If a text has already been translated, there is no need to look it up afresh. The translation is suggested and sometimes even added automatically, thereby increasing the efficiency of the translator. A translation memory system also serves to provide contexts for texts that need translation. With contexts, translators are guided in the right direction when trying to pick out the correct translation for texts with multiple equivalent texts in target languages.

Translation memories also serve to increase the efficiency of the entire translation process and, by extension, the entire localization process as well. One of the prominent ways it does this is by automating some parts of the translation process. For example, when a new text is encountered, a translation memory software can automatically provide, or at least suggest, a translation text for it in all supported languages. However, not all translation memory software can do this automatically. Yet, even the ones that do not could provide some assistance for the translator to do it manually. This can dramatically increase the efficiency of the entire process, as translation texts can even be drawn from across multiple projects belonging to the company, or beyond. Basically, a new project has most of its translation texts already filled in from the start.

7.2.2 Support Consistency Across Documents and Software

One of the main problems that translation memories solve is supporting consistency with terminologies and their translation [12]. Most times, companies want a text to maintain its meanings across multiple documents and user interfaces. When translators encounter a text that has multiple equivalent texts in a target language, they usually check the translation memory to see if it has previously been translated. If it has, that's the translation that they choose. Thus texts take on consistent translations, not only within a document or software, but also across multiple projects.

7.2.3 Reusability of translated texts

When a company that has a translation memory creates a new project, they do not need to start from scratch with their translation endeavours. As soon as the project is created, they automatically have access to a wealth of previously translated texts, that have been refined and reused over the years.

There are times where a term is difficult for certain situations. Given certain contexts, it might be difficult to come up with an equivalent in a target language. Experts may be invited in order to resolve the situation, which may be time consuming and costly. With translation memories, this is done only once. Next time the same texts are encountered in the project, there is no need to go through the process all over again.

By promoting reusability, translation memories increase consistency and reduce translation times and overall costs. This has proven invaluable, not only with software user interfaces, but also with their accompanying documents, such as project documentation, user instructions, e.t.c.

7.2.4 Reference tool

A translation memory acts as a reference tool for internal reviewers. The accuracy of translated texts on a product is very important. One of the steps that companies have to take in order to ensure the accuracy of texts from translators is to have them internally reviewed. Translation memories are very useful tool for this purpose. Apart from a general dictionary correctness, reviewers make sure that the new translations do not deviate from previously accepted terms by the company.

7.2.5 Increase in translation accuracy

Translation memories improve accuracy. To some extent, a translation memory serves as a dictionary, but more useful. In addition to the texts and their translations in various languages, most translation memories also store the contexts for the translated texts. Serving as guidelines, text contexts help increase the accuracy of future related translations.

7.2.6 Decrease in translation costs

A translation memory helps to decrease localization time and costs [12]. Most times, translation firms charge less for translations that already exist in their translation memories, either in part or as a whole, thereby reducing localization costs. Further more, if a company hosts their translation memory system internally or have direct access to it, non-expert translators are able to fill in some basic translations using texts from the system. In effect, this reduces the amount of texts that need to be sent out to external translators. In some cases, the translation memory systems are able to detect new texts, determine if exact translations are available in the database and, fill them in automatically without human intervention. Hence, only the missing ones are sent out, reducing the overall costs.

7.3 Translation Memory System Models

Translation memory system models are a way to describe how the translation systems store and access the texts that make up their databases. There are two systems commonly used by translation memories:

7.3.1 The Referencing Model

The first system uses a way to reference translated text segment data directly from files in previously translated projects. The files and their contents are accessible to the translation memory system in real time. This system mostly suited for projects with frequently updated contents both from developers and translators. As soon as texts are translated and saved to their source files, the translation memory software is able to access and make them available for reuse immediately. A major advantage of this

approach is that text contexts are easily available, as not just the translated segments are available but also the entire file content as well. This means that the translators therefore have access to contexts to guide their translation. Since the source files are accessed directly, the texts' translations and their contexts can be edited live. Figure 7.1 by [20] shows a generic overview of a possible implementation of such a system.

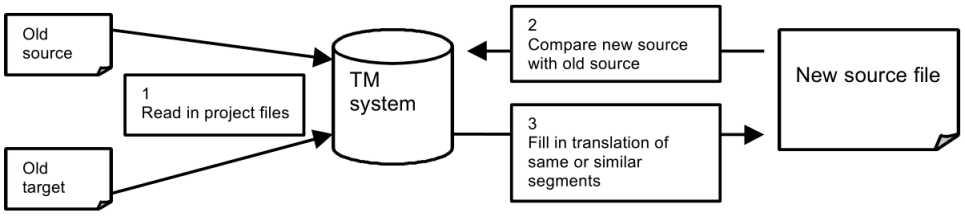

Figure 7.1: The Referencing Model

On start up, the system loads the project with the old files and their translations. Translations from new files are extracted and compared with existing ones. Suggestions for missing translations are offered from existing ones. The suggested translations are either chosen or new replacements added to the files. When this is done, the translations are extracted from the new files, forming new sets of data for future reuse.

It is relatively more difficult to implement. For example, multiple contributors may be working on the same project, and therefore the same set of files. Mechanisms have to be put in place to deal with clashes. Also, there has to be a steady, reliable connection to the source of data. Otherwise, the system would need to be designed to tolerate connection problems. A caching system can help to mitigate connection problems.

7.3.2 The Database Model

The database model system makes use of a database to store every translated text segment pairs ever made. Sometimes the translation contexts are excluded. The exclusion of text contexts, however, is a major disadvantage as they are normally needed by translators when dealing with similar text segments on other projects. Figure 7.2 by [20] describes an overview of a possible implementation of such system.

The use of a database gives the advantage of using database indexes for working with texts and performing searches. When a new file is encountered, the text segments

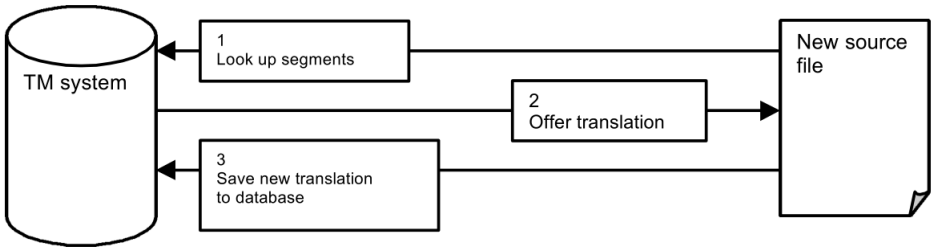

Figure 7.2: The Database Model

are extracted and new translations suggested for missing ones. The contributor choses the suggested translations or provides new ones. The new translation segments are stored in the database automatically. This system is relatively easier to implement, as the main component, which is the database, is a well understood technology and, there are robust, well-tested implementations, ranging from lightweights such as SQLite [35] to more feature-complete heavyweights, such as PostgreSQL [36] easily available for free.

7.4 Files and Data Storage in Translation Memory Systems

When translation memories work with files from different projects, which are of different types, or with files of different formats, there is need to convert each file to a generic format acceptable by the translation memory program. Some translation memory systems support more than one file formats. The conversion is done either by using the translation memory software itself or by the use of separate file conversion tools. Using a separate tool to do the conversion can introduce a bottleneck into the translation process. It is therefore carefully considered when deciding what software to choose. Also, after translations are made, they would have to be converted back into the original format to be re-inserted into the source project. The process could be time consuming.

Most translation memory systems have functionalities to read in files automatically. These raw data are converted into a form that is manageable by the system and allows a way for missing translations to be added by translators.

Some translation memory systems need intermediary tools, such as Microsoft Word,

in order to connect and operate input files. This way, MS Word takes care of the file conversion. This is especially useful in certain situations where, for example, the translation memory system is not able to work with input files in certain formats. The approach works as long as MS Word can work with the source format, and is able to convert to yet another format, like xml, which the translation memory system supports. MS Word converts the file to the required format, which is fed into the system. Better yet, the translators may even work directly on the data using the input file. The intermediary program, MS Word in this case, serves as a "What You See Is What You Get" (WYSIWYG) environment for the translators.

Checking to make sure texts fit properly into a given space in the graphic user interface (GUI), is normally done with specialized software for that purpose, or by the localization software tool suite. Sometimes the program is deployed to the target environment just for testing. In such cases, a manual inspection is carried out either by the translators or by the test team.

Translation memory software tools offer other functionalities, such as word count, automatic pre-translation of new texts, search functionalities from a database of previously translated texts, terminology management, etc. Most tools have additional specialized functionalities.

7.5 File Formats

Initially, translation memory systems use proprietary formats for the storage and exchange of data. This posed a restriction on the ability to transfer data between different systems, since not all software tools would be able to implement every format possible. To solve this problem, open standard formats were developed that would be universally acceptable among translation tools. The most supported of these formats for storing and exchanging translation memory data are *Translation Memory eXchange* (TMX) and *XML Localization Interchange File Format* (XLIFF). TMX is mostly used for the exchange of Translation Memory data between different tools [6, 2] while XLIFF is mostly used by Sun Microsytems and Oracle.

7.5.1 XLIFF file format

XLIFF was first released in 2002. The current version is version 2. Compared to TMX it is slightly more verbose. Although supported by many tools both commercial

and free, its use is still limited among translators. Its inclusion of features such as the use of schemas and language categorizations make it difficult to use without fairly sophisticated tools. Figure 7.3 show an example of a translation unit from a XLIFF file [2].

```
<trans-unit id="1" restype="button" resname="btnSAVE" coord="12;124;16;80">
    <source xml:lang="en-us">
        Save
    </source>
    <target xml:lang="de-de" state="needs-translation">
        Speichern
    </target>
</trans-unit>
```

Figure 7.3: A XLIFF translation unit example

7.5.2 *TMX file format*

At its core, a tmx file is a collection of translation units. In tmx terms, a translation unit is an xml element represented with the <*tu*> tag, that represents a concept in zero or more languages. Each language equivalent of the concept or term is held in a Translation Unit Variant (TUV) element, represented with a <tuv> xml element. Each <tuv> element contains one or more Segment element, represented with a <seg> xml element. The only attribute requirement for a translation unit variant element is the "lang" attribute. A text data is normally broken down into meaningful segments by line breaks, paragraphs, etc. Each one is stored in <seg> element contained in a <tuv> element. Figure 7.4 shows a minimal sample from Wikipedia [37].

All xml documents require only one root element. The root element for a tmx file is the <tmx> element. All the translation unit elements are collected under a body element, which is the second child element to the <tmx> element. The first is a header element and is not required, although not stated in the specification. If included though, it does have four mandatory attributes, which are creationtool, segtype, o-tmf and data-type. Among the others is a prop element (<prop>), which can be a child element to other elements, such as the header, the tu and the tuv elements. The main function of this element it to include additional custom data in the file. It could be used for storing project or software specific data, file version, etc. However it is also common for tools to use the custom attributes of the elements as a way to store data such as id, creation

```
<tmx version="1.4">
  <header
    creationtool="XYZTool" creationtoolversion="1.01-023"
    datatype="PlainText" segtype="sentence"
    adminlang="en-us" srclang="en"
    o-tmf="ABCTransMem"/>
  <body>
    <tu>
      <tuv xml:lang="en">
        <seg>Hello world!</seg>
      </tuv>
      <tuv xml:lang="fr">
        <seg>Bonjour tout le monde!</seg>
      </tuv>
    </tu>
  </body>
</tmx>
```

Figure 7.4: A tmx file sample

data, file reference, etc.

The standard for the specification is maintained by Open Standards for Container/-Content Allowing Re-use (OSCAR). The current version of the specification is version 1.4b [38, 39].

One of the main advantages of the tmx file format is its flexibility. Even the required <xml> element from the xml specification is not required in the tmx file system [38]. This flexibility has given rise to three levels of compliance to the specification. The first level of compliance is when only the texts are stored. This includes the minimum of the <tuv> elements, together with their various accompanying segment elements. A missing segment means that the translation is missing, and does not result in an error.

The second level of compliance is achieved when an implementation includes formatting information for the text segments. According to the specification the segment element has no attributes, but it is not an error to include some.

The third compliance level includes everything else and more. This includes things like tools-specific information, such as IDs for the various text segments, user IDs, project names, user defined data information, etc. Most tools support level 1 and 2 compliance [20]. The specification does not impose a requirement on the level of compliance a tool should have.

7.6 Choosing a Translation Memory Application

When choosing or developing a translation memory tool, it is necessary to consider file support, among other things. At a minimum, there should be a level 1 TMX compliance

support, with or without the use of external tools. In modern translation scenarios, it is necessary to have a level 3 compliance support. This level of compliance allows for storing of extra information that other parts of the workflow might require. The translation memory tool might go as far as to affect how the translations are entered into the projects. For example, in some projects the translators do not work directly with the project files. In such cases, the translation memory tools automatically extracts the text segments to translate and the translators work with them using the tool. When they are ready, it makes the new translations available to the affected project directly.

Translation memory tools are generally very expensive, hence cost is also a major issue to consider when choosing one. Sometimes it is more cost effective to develop an in-house translation tool that features a translation memory program among other programs.

8 In-House Translation Memory System

It is not always possible to do everything with commercially available tools for localization and translation. Companies sometimes have the need to develop in-house tools to facilitate the implementation of their localization processes. This need may arise naturally due to unique requirements from the way the company operates, or as a way to reduce localization costs.

Depending on their unique needs, in-house tools range from simple tools, such as, small scripts to extract texts from source code repositories, to full blown localization applications. One of such tools is a translation memory system or application. This chapter discusses a proposal for the design and implementation of a basic translation memory system.

8.1 Why in-house?

As discussed in chapter 6, one of the main challenges faced by companies with localization is that of cost. Localization is an expensive process for software development. As mentioned in chapter 3, localization could be so expensive it causes a company to narrow down its target audience.

In a company with multiple applications, one of the main sources of expenses is the repeated translation of the same, or similar texts and text segments. Although not always obvious, in a sufficiently large project, there is usually no easy way to know if a piece of text or part of it has previously been translated. The situation is even more complicated in cases that deal with dissimilar projects, with different people working on each. For example, a new paragraph of text needed in an application may have already been translated in the application's documentation. Since the developers working on the application are most likely not the ones working on the documentation, they

may end up translating the same texts all over again for the application, incurring unnecessary translation expenses. Obviously, visual examination by a user or developer is not a good option, and is not efficient. The only way to guarantee that a previously translated piece of text is not sent to translators again for translation, is to keep track of texts and their corresponding translations. As described in chapter 7, this is one of the main functions of a translation memory system.

8.2 System Description

The following is a simplified description of a simple translation memory system built for a local company. The entire system uses *NGINX*, both as a reverse proxy and a load balancer [40].

As a reverse proxy, NGINX directs traffic to the smaller apps that make up the system, either the user interface or the api. With this setup the api and user interface only run locally on designated ports on the server machine. The ports are not open to the internet, hence, the applications are not accessible to users through the internet. Users' browsers direct http requests to the nginx server via port 80, the default and, the only open port. Nginx directs the requests appropriately. This makes it easy to implement security for the apps, as security policies have to be enforced only on the base server layer. For example, the nginx server can refuse a connection from a particular user, or filter out certain requests before they even get to their intended application.

As a load balancer, nginx is able to run multiple instances of the user interface and the api apps. Hence, it's able to respond to higher demands on the system. Running multiple instances of the api, when a request arrives, the server is able redirect it to the instance with the least amount of load. This is a fairly common setup for systems that use the nginx server.

There are 3 main standalone components that make up the system, each of which is developed and deployed separately: the indexer, the api and the user interface, Figure 8.1.

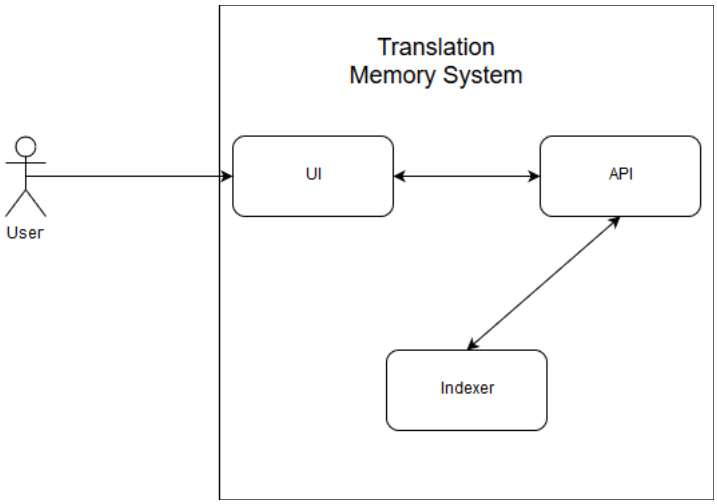

Figure 8.1: The Translation Memory System Diagram

8.3 The Indexer

The main functions of this system are to serve as a database for translated text segments and, indexing and retrieving the segments later in response to user queries. The main functional component of the system is an *elasticsearch* [41] software installation on the server machine.

Each time the system indexes the data, the entire current index database is dropped and a new one created, even if there is an existing one. This approach is necessary as there is no way to track the text segments or translation units from the different projects permanently with constant identifiers. This is a limitation imposed by the fact that when the tmx files are created, the text identifier information are lost and not available in the final combined file. Because of this limitation, the system always performs a fresh re-indexing operation every time it is (re)started.

8.3.1 Implementation Overview

When started the indexer app searches the server file system for a tmx database file that contains the translation data from all the projects. This file, which is located in a specific location in the server secondary storage, is loaded into memory for processing. Each translation unit element in the file contains at least two pieces of information that are needed to correctly store it in the indexer database. These are the project identifier and the translation unit identifier. These information are also necessary for perform-

ing searches on the indexer database. Other information, such as the language of the translation unit are already available, and are needed for organizing and displaying the search results on the user interface, when a search is performed.

If the tmx database file is not found, a specific language database git repository 8.3.2 is cloned. The file is "checked out" from the master branch of the repository. After processing, the translation data items from the file are extracted and converted into index entries. These entries are sent out to be indexed in the elastic search indexer database.

After all the data items have been indexed, the indexer is sent into a ready/waiting state. Before going into the ready state, a repository checker system 8.3.3 is activated. The checker runs after a specific amount of time. It checks the translation tmx database repository and if there is a new commit in the master branch, the indexer system is triggered to re-index the search database. After the ready state is activated, a signal is sent to the PM2 orchestrator.

8.3.2 The tmx file repository

A script is run at intervals to extract all the tmx data from each project, combining them together and saving them into a single tmx file. This file is referred to as the tmx database file and is composed of all the translated text segments from all the projects combined into a single tmx file and committed into a specific repository.

A repository is set aside for storing it. It is a regular git repository. A git repository is chosen for this purpose for the following reasons:

Easy to maintain

A git repository would be easier to maintain than a regular database for this purpose. Creating a normal database solely for the purpose of storing the tmx data would be too cumbersome to maintain.

Automatic version control

Sometimes there is the need to maintain the various versions of the translated texts. Using a regular database or the file system for this purpose would require the development of yet another system in order to keep track of the file history and states. A git repository database already has this functionality by default.

Secure way for the data builder system and the indexer to access the data

The file stored here would be accessed by at least two systems: the script that creates the list and the api system. This is yet another functionality already available in a git repository by default. It can also be achieved using a regular database system like PostgreSql [36]. However, in order to use that, it will have to be setup specifically for this purpose. Git on the other hand, already has this system in place.

Avoid re-inventing the wheel

The company already has a git repository and a few authorized bots with access. There is no need to implement a new system to achieve the intended use. All that is needed in this situation is to create a new repository, give one of the bots a read access and the system is ready.

8.3.3 The Repository Checker System

This is a small sub system of the indexer that runs checks on the tmx database file repository for new commits. By default, the system checks the master branch, although a different branch can be specified. It achieves this using a tiny library wrapper over the command line git program. On completion, a "callback function" is called with a boolean flag that indicates whether a new commit is available; true for yes and false for no.

By design the system does not perform checks at regular time intervals. It only performs a check after a given amount of time after a previous callback completes. A new check is scheduled only after the current one is done. The reason is so that the program does not trigger overlapping checks, since there is no definite way to know in advance how long a currently running callback function needs to complete its task. Hence, a new check is only scheduled after the completion of the current one.

8.4 The Search Api

This is a small search api application that serves to send queries to the elastic search indexer for available translated text segments. This component determines the nature and structure of queries that are made against the search index.

The api only has one endpoint, which is the "/serve" endpoint. The endpoint receives search queries as post requests. It processes the requests and creates the appropriate search request queries for the indexer. It is also responsible for processing the returned response data from the search index, and converting them into result lists appropriate for the user interface.

By serving as a mediator between the user interface and the search index, a separation of concern is achieved through abstraction. The search api abstracts away the functionality of the search indexer, creating the separation of concern. As a result, the version of the indexer or, even the entire indexer technology stack may change in the future without having any effect on how the searches are perform and, by extension the user interface.

Another important reason for the separation is the ability to easily cache search results. This serves to make subsequent searches faster. The cache can also service search requests in situations where the search api is busy.

8.4.1 Implementation Overview

The main technology that powers the search is a node.js server application built using the express.js framework [42]. A node.js/express.js combination was chosen for the following reasons:

The combination easy to set up

Setting up node.js on a Windows machine only requires the running of a single .exe installation file and on a Linux machine, only a single command. Setting up a server application requires only a minimum of one JavaScript source file.

It's easily extendable

The express.js framework is built around a simple middleware pattern. To add a new functionality, for example, only requires implementing a JavaScript middleware function and adding it into the appropriate place in the application. The express.js documentation gives very clear explanation as to when, and how a middleware is executed. This makes it very easy for developers to implement a functionality that they need, if it's not already available in the framework. In most cases, these functionalities are

already implemented by other developers and are available as open source projects on the internet.

It's easy to work with

Both the node.js and the express.js frameworks have extensive and, very rich documentations accessible via the internet.

It has a very large userbase

The main benefit of using a framework with a large userbase, among others, is the ability to easily find solutions to problems. When stuck on a problem, a quick search query on a search engine reveals someone else who has had that same problem, what they have tried and, possible fixes. Using a technology with a small userbase, on the other hand, leaves the developer to solve common problems on his own.

Flexibility

Most web application frameworks such as Django, need a server in order to run. A node.js application is itself a web server, while at the same time is able to run inside another more advanced server. Hence it is up to the developer to choose whether to run the application as part of another one on the same server or to run it on its own as a standalone program. This also makes setting up a developer environment easier.

It should be noted, however, that these advantages are not peculiar to the node.js or the express.js framework.

8.5 The User Interface Application

The user interface is a small web application that provides a means for users to query the translation index. It is the only component of the system that the user can access. It provides limited but vital use: a way for users to perform searches and display the search results. Based on the user input, it prepares a query in a format acceptable by the api backend, and formats the response in a presentable format to display to the user. Figure 8.1 shows a mock-up of the user interface.

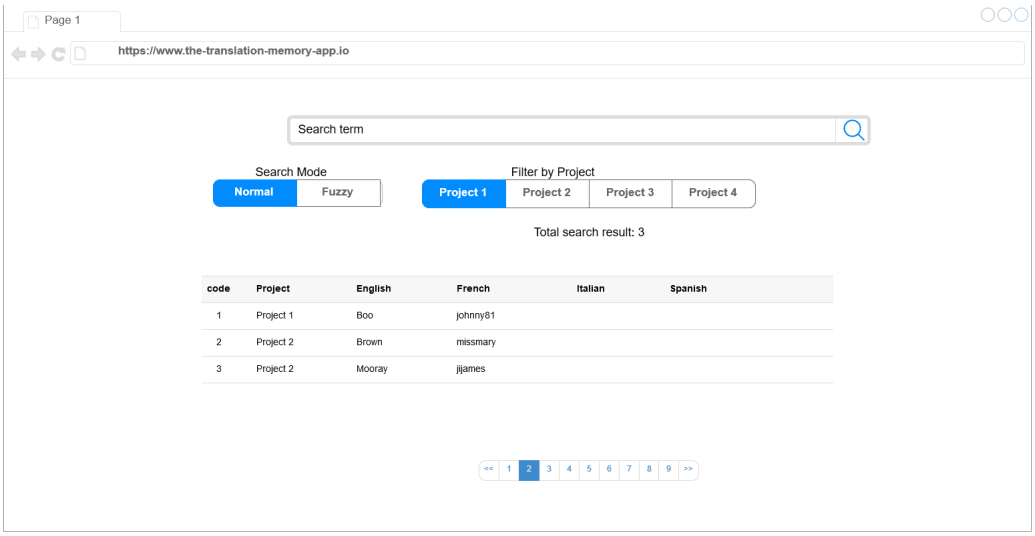

Figure 8.2: A Mock-up Diagram of the User Interface

8.5.1 The User Interface Components

The user interface application is a small one. Hence, it is composed of only a few user interface components. These components are the bare minimum it needs in order to perform its basic tasks.

Search Bar

The search bar component is used for entering queries and executing a search. It is composed of a text input and a button. After typing in a text the user clicks the button or presses the enter key on the keyboard to send a search request.

Search Mode Toggle Buttons

These are a set of two buttons for toggling between fuzzy and normal search. The default is normal search. The normal search performs an exact match query while the fuzzy mode performs a fuzzy search.

Project Filter

This is a user interface component composed of buttons for each supported project. Its functionality is to help build a query that filters the search results by projects. This is

useful for situations where the user only wants to search the index for texts from only one or more projects.

Info Display

This area shows the state of the application or the number of results received from the api. For example, it displays the text, "loading..." while a query is being processed by the backend api. The area also displays an error message if a search fails. One reason could be that the api is down, unreachable or busy. After the query is processed and the results retrieved, this area displays the number of hits for the search query. If there are no results it shows a "There are no results for this query" message.

Result List

This component displays the results of a search to the user. It displays fifty (50) results at a time on the page. The results are displayed in a tabular format, arranged by projects and languages.

Pagination

Sometimes a search query has a large amount of response. While the results are displayed in order of relevance, there may be a need to see results from the less relevant areas of the list. This is the purpose of the pagination component. It gives the user a way to page through the result sets up to the end of the list.

8.5.2 Implementation Overview

The user interface is built using the Vue web application framework. It is a progressive web application (PWA).

Progressive Web Apps (PWA)

Progressive web apps are web applications that are developed such that they incorporate the advantages of both web and native applications [43]. That is, they offer features of both web apps, as well as some of native apps. Just like native applications, they can be installed on user devices so that the next time the user wants to use the app, they do

not need to remember the web address. Instead, they can just launch it from the home screen or desktop of the device.

The user interface components for the application are also cached in the storage of the user's device. This allows the app to load faster in the future. Progressive web apps have a more extensive access to user device storage than normal web applications. Hence, they are able to store a larger amount of data on the device, and for extended periods. In most cases, this allows the application to run offline, without access to the internet.

Just like a web application, users can share the app with other users via links. After visiting the application on a web browser, the user is offered a possibility to install it in their devices. If they choose to not install it, they can continue using it simply as a web app. Also, being a web app offers yet another powerful advantage: it can be built using web app technology and frameworks, which are nowadays ubiquitous.

8.5.3 It uses the vuestrap css framework

The vuestrap css framework was created by Yu Che [44] as a Bootstrap JavaScript implementation using the vue.js web framework. The Bootstrap framework is one of the most popular frameworks for developing user interfaces for websites. Normally, to use the JavaScript functionalities it provides, it was necessary to include the jQuery library, adding an extra dependency. With VueStrap, jQuery is not needed.

8.5.4 The UI is served directly by the Nginx server

The user interface resources are served directly from the server file system using the Nginx server. The Nginx server already comes with everything needed to serve the files. The application itself is actually run by the user browser. The server only needs to send the appropriate files; a task the Nginx server does with great efficiency.

One of the main advantages of such a setup is scalability. Since the user interface is just a set of files hosted on the webserver file system, it becomes easy to scale it effortlessly. Nginx comes with such a functionality. It can seamlessly run multiple instances of the app if user traffic increases.

8.6 Limitations of this System

One of the most important functionalities missing from this system is the lack of automatic text insertion. Although it can suggest translations for new text segments, it is not able to insert those into the intended project automatically.

Another limitation is that the system does not provide a way to estimate how translated texts would fit into the user interface. This is very useful feature, although it is not strictly a feature of a translation memory system. Some modern translation tools are able to give the user an estimate of how much space is needed for a text on the user interface without having to actually run the user interface program.

This might seem like a trivial problem of counting the number of characters. On closer examination, it is not. For example, what constitutes a single character might vary widely from one language to another. Also, the length of characters alone might not be relevant for determining if the texts fit a space on the user interface, as there may exist a possibility for the texts to expand vertically as well as horizontally. The easiest solution, in most situations, is to run the application and observe how the texts turn out. Adjustments can be made later and the app rerun to see the result of the new changes. However, this is time consuming. At Memrise, the problem of time consumption is mitigated by giving the translators the ability to deploy and test a version of the software without the involvement or help of the development team [7]. Hence, they can see how their changes affect the system, and only submit their final work after they reach a satisfying result.

One other very important feature that is missing is the ability to store and make use of text contexts. Since text segments are stored along with their translations only, cases arise when a word or phrase is repeated with different meanings for other languages. In such a case, the context for the terms are needed in order to determine the source-target pair that's most suitable for a particular situation [29].

9 FUTURE WORK AND CONCLUSIONS

There are three broad areas of localization that were covered in this thesis: general description of localization, localization as a process and a little about the tools used. The thesis also gave a description of a translation memory system implementation for a multi-project translation project. Following is a rundown of some of the conclusions drawn and lessons learnt in the research for the thesis.

9.1 Sources of information

Sources of information about localization and all the tasks and procedures involved can be broadly divided into three categories: academic research sources, localization tools vendors' sources and articles from companies describing how they actually implement their localization processes.

The least reliable source of information about localization and its processes are those from localization tools and software vendors. These sources are normally filled with ads in the form of information that makes the task of sifting out useful information much more difficult. Also, their definitions of important concepts are, in most cases, tailored to fit their products. These sources also include terms that are mostly made up for their products. However, there are advantages to these sources. For example, they usually have up-to-date information about the latest problems faced by translators and other personnel working with localization and how they are solved. Mostly, their offered solutions depend on their products, making it difficult to tell if these are actual problems or just means to showcase their products.

Sources from academia are usually the most accurate. Accuracy here refers to the definition of terms and the distinctions between them. However, since localization is not a very popular topic in the academic world, there are fewer academic studies than in the other areas of software development. The real disadvantage of this category of information sources is their applicability, that is, they are generally the least applicable

in real-life situations.

The most reliable category of sources of information in this area of endeavour are articles from real-life companies that describe how they carry out localization and the tools that they use. However, they are usually the least accurate as regards definition of terms. Most often, there is no clear distinction between localization and translation, but in terms of applicability, they are usually the best.

Indeed there are sources that have a mixture of two of those, such as [12], an academic study which details how an actual American dental equipment manufacturer company go about localization, including the challenges they face and how they are handled. Such sources are usually also very reliable, as they combine the advantages of academic sources and those of real-life companies.

9.2 Localization Processes

As so often in software development there are no standard procedures for localization. Every organization has its own way of carrying it out. The main determining factors for the localization processes of companies are the tools and particular software development processes the team uses. Most companies build their localization processes around the tools they use.

The localization process could be messy and difficult to manage or reason about. This thesis suggests a system of reasoning about it. It employs a way to break down the entire process into smaller tasks that can be used as a way to organize one's thoughts about it. Each task can be optimised individually.

9.2.1 Future work on the proposed localization visualization

There are many tools for visualizing processes. In the future, a simple system of flowcharts can be used to give a visual representation of any company's localization process. A simple tool could be developed that allows modifying the system by a simple drag and drop, and connecting the processes together. The tool could be used to give a general time estimate of the entire system. Finally, the tool could automatically generate documentations for the entire process from its visual representation.

9.3 Localization Tools

There are many tools involved in the localization process, ranging from simple tools such as slack bots to sophisticated localization management software tools. Most often companies have to develop an in-house tool to aid in at least one area of the task. One of such in-house tools is a translation memory system. A translation memory system helps to store and index text segments that have already been translated and provides a way to query them. When the texts are coming from multiple, possibly unrelated projects, the task of building such a system becomes complicated. This thesis proposes a simple design and implementation for such a system.

9.3.1 Future work on the translation memory system

Most companies and organizations use slack for communication [45]. Slack could be used to trigger an immediate re-indexing of the text segments database instead of waiting the designated timeout period. This could be done to make newly added text immediately available for searching on the user interface. To trigger a re-index, a user only needs to send a command message to a slack bot. This system is used at Memrise [7]. It gives non-technical users the ability to perform technical tasks such as triggering builds.

Another area where the system would benefit from a slack integration is in the area of notifications. The system could be used to notify the localization personnel, such as the localization managers or translators, of new texts segments that are not yet translated. It could give a list of missing translation texts together with the projects that they are from. Thus, not only the translation personnel are kept up to date with the situations but also every other interested party including the development team.

BIBLIOGRAPHY

[1] Anthony Pym. Localization and linguistics. In *SLE conference*, 2001.

[2] Dimitra Anastasiou. Survey on the use of xliff in localisation industry and academia. In *Proceedings of Language Resource and Language Technology Standards–State of the Art, Emerging Needs, and Future Developments Workshop, 7th International Conference on Language Resources and Evaluation (LREC), Malta*, pages 50–53, 2010.

[3] Transifex. Localization 101: A beginner's guide to software localization. URL: https://www.transifex.com/.

[4] Elvis Hau and Manuela Aparício. Software internationalization and localization in web based erp. In *Proceedings of the 26th annual ACM international conference on Design of communication*, pages 175–180. ACM, 2008.

[5] Peter Sandrini. Website localization and translation. In *EU-High-Level Scientific Conference Series MuTra*, pages 131–138, 2005.

[6] Bert Esselink. The evolution of localization. *The Guide from Multilingual Computing & Technology: Localization*, 14(5):4–7, 2003.

[7] Memrise. *Automating Localisation Workflows at Memrise*, 2017. URL: https://engineering.memrise.com/automating-localisation-workflows-at-memrise-139743c32a42.

[8] Rosann Webb Collins. Software localization: Issues and methods. *ECIS 2001 Proceedings*, page 4, 2001.

[9] SS64. Windows locale id table. URL: https://ss64.com/locale.html.

[10] Israel Ministry of Foreign Affairs. About israel. URL: https://mfa.gov.il/MFA/AboutIsrael/Pages/default.aspx.

[11] R. Ramler and R. Hoschek. How to test in sixteen languages? automation support for localization testing. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 542–543, March 2017. doi: 10.1109/ICST.2017.63.

[12] D. Ledet and R. A. Bailie. Following the road untraveled: from source language to translation to localization. In *IPCC 2005. Proceedings. International Professional Communication Conference, 2005.*, pages 32–39, July 2005. doi:10.1109/IPCC.2005.1494157.

[13] Abdulmajeed Alameer, Sonal Mahajan, and William GJ Halfond. Detecting and localizing internationalization presentation failures in web applications. In *Software Testing, Verification and Validation (ICST), 2016 IEEE International Conference on*, pages 202–212. IEEE, 2016.

[14] Django Framework. Django - the web framework for perfectionists with deadlines. URL: https://www.djangoproject.com/.

[15] Django Framework. Source code for django.utils.translation. URL: https://docs.djangoproject.com/en/2.1/_modules/django/utils/translation/.

[16] Angular Translate Framework. Angular translate - i18n for your angular apps, made easy. URL: https://angular-translate.github.io/.

[17] Zeit. Next.js. URL: https://nextjs.org/.

[18] Google Inc. Google website translator. URL: https://translate.google.com/intl/en-GB/about/website/.

[19] Y. Achchuthan and K. Sarveswaran. Language localisation of tamil using statistical machine translation. In *2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 125–129, Aug 2015. doi:10.1109/ICTER.2015.7377677.

[20] Angelika Zerfass. Evaluating translation memory systems. *LREC 2002: Language Resources in Translation Work and Research*, 28:49–52, 2002.

[21] J. Archana, S. R. Chermapandan, and S. Palanivel. Automation framework for localizability testing of internationalized software. In *2013 International Conference on Human Computer Interactions (ICHCI)*, pages 1–6, Aug 2013. doi:10.1109/ICHCI-IEEE.2013.6887796.

[22] Tryggvi Gylfason Spotify Labs. (right to left (the mirror world, Apr 2019. URL: https://labs.spotify.com/2019/04/15/right-to-left-the-mirror-world/.

[23] Transifex. A localization platform that moves as fast as you do. URL: https://www.transifex.com/.

[24] Telegram. Translating telegram. URL: https://translations.telegram.org/.

[25] Telegram. Telegram faq. URL: https://telegram.org/faq#q-can-i-translate-telegram.

[26] Netflix. Netflix. URL: https://tests.hermes.nflx.io/.

[27] Slator Esther Bond. Why netflix shut down its translation portal hermes. URL: https://slator.com/demand-drivers/why-netflix-shut-down-its-translation-portal-hermes/.

[28] Lingobit Technologies. Lingobit - localization on demand, May 2019. URL: http://www.lingobit.com/.

[29] Stas Malolepszy mozilla hacks. Fluent 1.0: a localization system for natural-sounding translations, Apr 2019. URL: https://hacks.mozilla.org/2019/04/fluent-1-0-a-localization-system-for-natural-sounding-translations/.

[30] W3C. Richard Ishida. Character encodings for beginners, Apr 2015. URL: https://www.w3.org/International/questions/qa-what-is-encoding.

[31] W3C. Richard Ishida. Handling character encodings in html and css (tutorial), Aug 2010. URL: https://www.w3.org/International/tutorials/tutorial-char-enc/.

[32] Lau Taarnskov. Falsehoods programmers believe about time and time zones, Jan 2015. URL: http://www.creativedeletion.com/2015/01/28/falsehoods-programmers-date-time-zones.html.

[33] Lau Taarnskov. How to save datetimes for future events - (when utc is not the right answer), Mar 2015. URL: http://www.creativedeletion.com/2015/03/19/persisting_future_datetimes.html.

[34] Peter Sandrini. Localization and translation. *LSP Translation Scenarios*, 2:167–191, 2008.

[35] SQLite Consortium. Sqlite - small. fast. reliable. choose any three. URL: https://mfa.gov.il/MFA/AboutIsrael/Pages/default.aspx.

[36] PostgreSQL. Postgresql: The world's most advanced open source relational database. URL: https://www.postgresql.org/.

[37] The Wikipedia Foundation. Translation memory exchange. URL: https://en.wikipedia.org/wiki/Translation_Memory_eXchange.

[38] OSCAR (Open Standards for Container/Content Allowing Re-use). Tmx format, Nov 1997. URL: http://xml.coverpages.org/tmxSpec971212.html.

[39] LISA OSCAR Standards. Translation memory exchange (tmx). URL: https://www.gala-global.org/lisa-oscar-standards.

[40] NGINX. Nginx, Apr 2019. URL: https://www.nginx.com/.

This document was created: Sunday 18th August, 2019 16:49 by Chinedu Eze

[41] Elasticsearch. Elasticsearch - the elastic stack. URL: https://www.elastic.co/products/elasticsearch.

[42] Express.js. Express - fast, unopinionated, minimalist web framework for node.js. URL: https://expressjs.com/.

[43] MDN web docs. What is a progressive web app?, Apr 2019. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction.

[44] Yu Che. Vuestrap - bootstrap components built with vue.js., Apr 2019. URL: https://yuche.github.io/vue-strap/.

[45] Slack. Slack - imagine what you'll accomplish together. URL: https://slack.com.