Possible Translation Problems, Their Causes, and Solutions in Agile Localization of Software

https://doi.org/10.3991/ijim.v17i01.36367

Marián Kabát
Comenius University in Bratislava, Bratislava, Slovakia
marian.kabat@uniba.sk

Abstract—This paper presents some initial insights into practical translation problems that can occur during agile localization. Although agile localization is not a novel approach to software localization, the range of possible translation problems and their causes and solutions have not yet been described. In order to write this paper, an ad hoc monolingual English corpus made up of user interface strings of one agile localized software product was used. The corpus was analyzed string by string, and various causes of translation problems, mostly relating to lacking context, were identified. The paper presents and discusses sample cases of these problems. This paper tries to show that while agile localization can be used as an alternative to traditional localization, the development team must be open to cooperation with the localization team in order for agile localization to be successfully implemented; otherwise, problems will occur.

Keywords—localization, agile, context, fragmentation, internationalization, translation problems

1 Introduction

In order for software products to be sold abroad, they need to be adapted to the intended target markets. This adaptation process is important since, due to globalization, a product can travel to different cultures in the blink of an eye. Well-adapted software can increase revenue and, of course, needs to have its intended original usability.

Software developers have usually regarded localization as an afterthought, as it was generally carried out only after a product's development. In such a classic waterfall workflow, the release date of a product depended on the translations and possibly resulted in delays and missed deadlines.

An alternative to the waterfall workflow is the agile workflow (or agile methodology) which requires minimal documentation and preferably an on-site presence. The agile methodology quickly spread throughout the development industry and became broadly adapted to the extent that it influenced the process of software localization and created the concept of agile localization.

The aim of this paper is to explore the challenges localizers face when they deal with agile localization. Given that there could be various translation problems in this process, the qualitative analysis mainly focuses on problems dealing with context issues and text

fragmentation since they can both be caused by the nature of the agile methodology reflected in agile localization. The purpose of the analysis is not to give quantitative data (e.g., how often a translation problem is present), as this can change with the application. The aim is focused on the limitations of agile localization.

In order to identify translation problems stemming from agile localization, an English monolingual corpus consisting of one application (a web-based software product) was analyzed.

The empirical part of this paper will show that while many issues can be solved with some success depending on various factors, they nonetheless still constitute translation problems. In this sense, a translation problem is a problem regardless of whether it can be solved or not and regardless of the experience of the translator.

This paper intends to show some of the translation problems that localizers face when they complete agile localization tasks. If these problems are not solved adequately, they can cause localization and translation errors and may even pollute the translation memory that is used. The qualitative analysis provided here is an attempt to explore the possible problems of agile localization.

Since this paper focuses on the novel concept of agile localization, Section 2 covers a brief literature review of the concept. Section 3 introduces the topic of agile methodology as it is understood in the field of software development. Section 4 describes localization before and after agile localization, focusing mainly on the process of agile localization and its benefits and shortcomings. Section 5 presents the methodology of the empirical study, and Section 6 presents the findings in the corpus. Lastly, Section 7 presents the conclusions of the paper.

2 Literature review

The concept of agile localization seems to have been overlooked by academic writing, as there is not a lot of academic literature dealing with it. In fact, only Malte Ressin seems to have dealt with the problems of agile localization, albeit by looking at relationships between a development team and a localization team working on the same project. Ressin's papers [1], [2] and [3] examine agile localization from a psychological standpoint, pointing at different goals, an understanding of each other's subjects, and concepts of the quality of two involved teams. Ressin also completed a dissertation [4] on this matter.

Authors like Esselink [5] or Roturier [6] deal with localization from a practical standpoint, but they only mention agile localization very scarcely, and do not examine the topic in more detail.

Another source of literature on agile localization, albeit not of an academic nature, is the plethora of informative blogposts by different translation and localization agencies. Some recent ones include Pereverzevs [7], Phrase [8], and Trusava [9]. These blogposts have a common structure: the text first introduces agile localization and how it works; then it describes all the benefits of agile localization; then it briefly mentions some negatives, followed by tips on how to implement agile localization; and at last of

all finishes with an offer by the translation or localization agency to implement or help with localization.

As can be seen from the lack of academic literature, agile localization is an overlooked concept that is promoted mostly by the language services industry and has not yet been given much scholarly attention. This is why this paper seeks to shed some light on agile localization from an objective view on the subject rather than a sales-driven perspective.

3 Agile methodology

Agile methodology (agile software development) constitutes a group of software development methodologies that are based on iterative and cross-functional team collaboration approaches [10], [11]; with the increasing use of SaaS platforms, software developers have acquired the chance to update software, fix bugs, and add new features almost in real time. Agile development basically breaks the development process into smaller iterations, so the requirements for a developed product change to a more flexible need that is based on the current need of the product and the development team.

Agile software development can proceed according to several different frameworks and methodologies of software development (e.g., Lean, Kanban, Scrum, and Extreme Programming) which are all based on the central idea of agile development [12]; this idea is described by the Manifesto for Agile Software Development as "the Agile Manifesto".

The Agile Manifesto [13] describes twelve basic principles of agile development that can be summarized as follows: individuals and face-to-face interactions are valued more than processes and tools; working software takes precedence over a lengthy and comprehensive documentation; customer collaboration is more important than contract negotiation; and fast responses to changes are more important than following a strict plan.

Such fast-paced steps and just-in-time manufacturing leads to quick responses by the development team and to the already mentioned breaking up of a product into smaller parts, which in turn leads to faster smaller releases and adjustments to change. Such a process of fast-paced development does not take localization into account; it is a process that takes time, requires context or documentation, and is often regarded as the last step before a product release. What then is the place of localization in agile development?

4 Localization before and after agile

In order to better understand the "new" concept of agile localization, let us first take a look at the "old" or "traditional" form of localization. To make a distinction from agile localization (the "new" concept), the paper will use the term "waterfall localization" for the traditional way of software localization.

4.1 Waterfall localization

There are two main approaches to traditional waterfall localization that are being used [7]:

1. Post-release – localization of a product takes place after the release of a product. In other words, the localizer or the localization team starts to work on the translation after the final product has been delivered. This approach causes delays, as localization can take a lot of time. A company is therefore incapable of making a product release and loses revenue in the process as well.
2. String freeze – during this approach, there is a period during the development process called "string freeze" when strings (i.e., lines of code) that need to be translated are locked and cannot be changed in any way. A string freeze can last several weeks, during which time the localizer or localization team works on a translation which is then sent to the developers before the release of a product. While a portion of the strings is frozen, developers can work on debugging other portions of the software; while the string freeze process saves time, developers must identify the modified string in the code manually.

There is also the problem of two groups of professionals working separately [1] where the development teams do not always expect the impact of a new feature on localization; once this new feature gets to be localized (through either of the two mentioned methods), there is already a more advanced stage of development which may make any significant changes a difficult prospect.

There are also tasks that can be automated by dedicated tools, e.g., software developers manually extracting source content from strings or databases, project managers e-mailing files for localization, and software developers cutting and pasting translated strings from a spreadsheet into a source file.

The biggest issues of traditional waterfall localization are that it is time consuming, can cause delays, and requires a lot of manual work that today can be automated. But at what cost?

4.2 Agile localization

By contrast, in agile localization, "localization is not only an afterthought to software development" [1]; it is (or should be) integrated into the development process so that both processes operate simultaneously. This is usually done by a localization platform which automates the translation process.

The process of agile localization can be broken up into the following steps (adapted from [8]):

- Choosing a localization management tool: a localization management tool is basically a CAT tool with access to API (application programming interface) and often also to an external cloud-based system where new strings are uploaded by the development team. Choosing a localization management tool should be one (if not the first) step in agile localization as the tool will be used throughout the whole

process, preferably by both teams (development and localization); it will contain a translation memory and a terminology database. The localization management tool basically represents a meeting point for the development and localization teams.
- Internationalisation: adapting a software product to different languages and locales. In other words, this is using programming mechanisms that change content (e.g., time and date formatting, currency, images, and texts) based on the selected locale.
- Locale file creation: internationalisation results in a set of resource files, usually with identifiers mapped to strings in the programming language. The strings in the resource files are then extracted and converted into a format the translators can work with, thus creating a locale file in the source language.
- File receipt and translation: the extracted and converted files are then forwarded (usually via a cloud-based platform) to the localization team and translated in a standard way (e.g., using a CAT tool, translation memory, terminology database, or machine translation).
- Review: the finished translation should undergo a review process. Since the whole process can take time, it is beneficial for the localization management tool to have an integrated review functionality. The review process should focus on standard translation properties like accuracy, consistency, and correctness, and it should also pay attention to the number of line breaks, HTML tags, and string length. Such checks can be automated depending on the tool that is used.
- Translation files integration: the translated files are then returned to the development team (usually via the same cloud-based platform) to be converted into the desired file format and then integrated into the code.
- Localization testing: after the translation file integration into the resource files of the target locale, the software product needs to be language tested. Some localization management tools provide an in-context preview system that can speed up the text or product adjustment process if there is a need for one.
- Publishing: once the language testing is finished and the target language version of the software product is complete, the localized software is published or released, thus concluding the process of agile localization.

In order for agile localization to work, it needs to be integrated into the whole product development cycle – starting with the choice of a localization management tool – so that new texts can be translated simultaneously along with the product development. A process like this has its benefits and challenges when compared with the traditional waterfall localization workflow; the following subsections will deal with these.

The benefits of agile localization. Agile product development has its benefits over the waterfall workflow. For instance, agile projects are more likely to be finished on time (65% compared to 40%), they tend to accomplish all goals (75% compared to 56%), and companies adopting agile methodologies grow their revenue 37% faster [14]. These benefits are naturally reflected in agile localization as well. Some of them include (adapted from [9]):

- Faster time-to-market: since localization occurs at the same time as other software development activities, publishing occurs sooner and the process of continuous localization becomes a part of the continuous deployment process. Products get

deployed even when there are imperfections in localization as these can be fixed in future updates.
- Reduced costs for the developer: the localization team only translates new or updated strings, so the development company saves money.
- The time saved by the localizer: since only new or updated strings are worked on, the localization team saves time and can work on other projects as well.
- Easier mistake detection: as the translations are rapidly implemented during the release, potential errors are detected earlier and can be corrected in the next release.
- Less manual work: since the localization management tool communicates with the cloud-based repository, tasks like manual string extraction can be automated and happen automatically.
- Faster localization testing: as translated strings get deployed automatically, localization and language testing can begin sooner.

The challenges of agile localization. The fast-paced processes of agile localization have several drawbacks as well (adapted from [9]):

- The importance of context: since precise documentation is not an important aspect of agile localization [13], translators can lose out on important context details. This is especially alarming when a new translator starts working on an already running project and they are not yet familiar with the product. The issue of lacking context can be remedied by using glossaries and product knowledge bases and adding comments that explain the placeholders and screenshots of the software product.
- Team interactions: the development and the localization teams need to communicate effectively. If the teams are working separately from each other, processes might not be synchronised or there may be synchronisation issues [1].
- Time zones: when a software product gets localized into several languages around the globe, time zones will be an issue as these will inevitably slow down the agile process. If we consider a possible lack of context and the need to consult parts of the text, a simple project can take longer than expected.
- Text fragmentation: as only new or updated strings get translated and prepared for localization, the strings a translator sees in a CAT tool will not necessarily follow each other in a logical way; the source text might become fragmented.

Summary. If speed is a central keyword in agile development or localization, agile localization can either speed the time-to-market up or slow it down depending on other aspects of the process. A functioning source of context, translation guidelines (e.g., a style guide or a glossary), good teamwork, and interactions between the development team and the localization team can speed up agile localization. If the localization process is followed up by localization and linguistic testing, the resulting localized software product can be of a high quality.

The opposite can also be true – if the localization team lacks context or additional information, or if interactions between the development team and the localization team are missing, the resulting localized software product might contain some forms of error which might not be detected in the localization and linguistic testing before release. However, even in this case, agile methodology might prove useful as frequent updates

and releases can contain fixes for errors in localization. On the other hand, fixes cost additional time that was spared in the first (successful) scenario.

5 Methodology

This is a qualitative and descriptive study based on a single monolingual English ad hoc corpus made up of strings of a web-based service applications which was agile localized. The corpus consists of 76,820 source words and 15,254 strings.

The analyzed software product was chosen because it was developed with an agile methodology and has been agile localized. Due to signed non-disclosure agreements, the translation problems that will be used as examples in the following section will be anonymized in order for them to be able to be presented in the findings. Anonymization will take place if a source segment contains a term which would allow for the software product or client to be identified. If a source segment contains such a term, it will be swapped for a generic word with the same or similar meaning (e.g., a segment like "Run Linux" would be anonymized as "Run system", and "Run Bluetooth" would be anonymized as "Run function").

The corpus was manually analyzed segment by segment. Since the aim of this paper was to observe a source text as translators do when they localize a product in order to identify potential problems, a CAT tool was used; the corpus was processed, and the segments were analyzed in the environment of the tool.

The corpus was processed in the same CAT tool, in which the software product is agile localized. The CAT tool is called Smartling Translation Management System, which is a cloud-based tool. The tool follows regular industry segmentation practices and includes standard features (e.g., translation memory and terminology).

In terms of the object of this research, only translation problems related to context have been selected. The following section describes the problems identified during the corpus analysis.

The aim of the analysis is not to give a frequency with which each translation problem comes up in the corpus, as this can change depending on the software product. It is merely to show the possible limitations of agile localization of applications.

6 Findings

This section will present various examples of practical problems related to context issues that can arise during agile localization and discuss their causes and possible solutions. Seven examples will be given. Each example will begin with example strings that will then be followed by an explanation of the problem, the possible cause and ways to remedy the given problem, so as to minimize the risk of producing an incorrect translation.

Example 1:

search box

The first example is a string containing a collocation. Although the collocation is self-explanatory and probably also easy to translate, the problem is that the string begins

with a lower-case letter. In this case, the string can have several functions in the software product:

- it can form a placeholder (the content of the string will be turned into a placeholder and will be inserted into various other strings at a later point): this is a problem for fusional languages, where the content of the string might need to undergo declension; however, once the placeholder content is translated in a grammatical case (the nominative would probably be standard), the same grammatical case will be used whenever the placeholder denoting this collocation is used.
- it can be part of a fragmented sentence: a sentence containing this collocation might have been split up into several strings during either step of agile development or localization. In this case, the translator needs to identify the rest of the sentence in the given project and translate accordingly.

A string like this could be easily translated if the localizer had either visual context in the form of a preview (this would be helpful in the case of fragmentation) or a comment by a developer explaining the purpose of the collocation (this would clarify whether the string will be used as a placeholder later).

Example 2:
Show Unsafe Content
Show unsafe content

This example contains two strings; although they follow each other here, they were separated by several other strings in the corpus. It might not be difficult to translate them, and a CAT tool would help with consistency since a translation memory would suggest a previous translation, but the different capitalization might create problems.

The first string, due to all three words being written with capital first letters, might be a button, a title, or the name of a dialogue box. Depending on the target language, these UI elements might require different grammatical forms (e.g., a button might require an infinitive verb, whereas a title or a dialogue box name might require a verbal noun).

The second string might be the beginning of a fragmented sentence or an option near a check box, again requiring different approaches depending on its position in the software product.

Visual or written context in the form of a picture or a comment would be helpful. The various grammatical forms used in different UI elements should be described in a style guide, but without more information a localizer would be unable to identify the correct element. The worst-case scenario in this case would be guessing, but this might result in a wrong or inconsistent translation and even the pollution of the translation memory.

Example 3:
Rewards has been turned off on this device by your administrator.

In this case, the localizer needs to know what “Rewards” means as this could be a new function of the software product. Otherwise, the English sentence would contain a

grammatical error (correct: "Rewards have…"). Again, a simple comment from a developer would clarify this issue. If the string does not contain any comment, the localizer might think the sentence has a grammatical error and translate it accordingly, which in this case would result in an error. Problems like this highlight the importance of team work as well. Localizers should be a part of the development team, so that they are informed about new functions that will be added to a software product in the upcoming development.

Example 4:
Hide favorites button from toolbar
Show Favorites Button in Toolbar

This example contains two strings that did not follow each other in the corpus. Although the strings are quite similar, the capitalization in the second string might be confusing. Notice that the element "Favorites" will also be used in Example 7, where it was identified as a UI element. In this case, it can be confusing as to why the first string does not capitalize the name of the button. The second string might point to a title since all the words are capitalized.

The problems in this example could be solved either by using comments from the developers or through a more careful approach by them, whereby they would only capitalize words that actually have to be capitalized (e.g., Favorites button).

Example 5:
Schedule

This string containing a single word is a common problem in localization, and not only in agile localization. Without further context, the localizer does not know whether the word should be translated as a noun or a verb, or, should there be more synonyms in the target language, which synonym they should choose.

The localizer could contact the development team, but due to the fast-paced nature of agile localization, an answer might not come in time before the deadline of the project (e.g., due to different time zones).

Example 6:
our team if this issue persists.
Try refreshing the page, and please
Contact.

This example contains three strings following each other in the corpus, and it represents an example of text fragmentation that occurs either during the export from code or during file processing in a CAT tool.

The localizer can be confused by the string order. The strings appear to form a single sentence if reordered ("Try refreshing the page, and please Contact. our team if this issue persists.") Although the sentence makes perfect sense and is probably easy to translate into the target language, the full stop at the end of the third string ("Contact.") is confusing. The third segment might actually not be part of the sentence at all and could be a part of a different fragment, or the full stop might be a typo made by a

developer. The solution to this problem could be provided by communicating with the development team or being given some form of visual context.

Example 7:
Reading list items have moved under "Favorites"

This example contains a UI element in quotation marks, which can be confusing since most other UI elements in the corpus were not inserted into quotation marks. This example could be a case of inconsistent labelling by the development team when one of the developers marks UI strings with quotation marks. Such inconsistency might be confusing, especially for new members of the localization team who are not familiar with the fact that while some UI strings are in quotation marks, they can be ignored. In this case, the developers should adopt a consistent labelling approach in order to not be ambiguous.

6.1 When context fails, communication should follow

Some of the problems stated above already touched upon the possible solution of communication with the developers. While having direct input from the development team would solve possible context issues, even this solution has its limitations.

Firstly, it is still common practice for localization experts to not be part of development teams. As a result, various internalization issues arise during development which then have an impact on the localization process [15]. Getting feedback from a localization professional would benefit the development process and would help with internalization issues (such as in Example 5).

Secondly, interactions and communication between the two teams is problematic due to the possibility of both teams being in different time zones as well as due to the outsourcing of localization. Even if the localization team raises a query caused by ambiguity, the query travels from one outsourcing company to another before reaching the developer; and then the answer has to travel back to the localization team. Such an exchange of information can take longer than a day, and with deadlines being strict and agile localization usually consisting of micro projects of no more than 500 words that need to be translated on the same day the project is received by the localization team, such team interactions or information transfer is simply not helpful.

Possible solutions to this problem could be hiring an in-house localization team, limiting the amount of outsourcing of a single project so that communication is more straightforward or investing adequate resources into localization and linguistic testing.

Of course, the present author is aware that such solutions are not always feasible, especially when localizing into less widely spoken languages. Nonetheless, in order to overcome translation problems in agile localization, resolving possible context issues, adopting a more careful development approach, and streamlining communication between the two teams should be a central concern.

7 Conclusions

Localization is stationed somewhere at the intersection between translation and technology [16], so it is no surprise that with the coming of new programming methodologies it would adapt and create new ways to bridge the gap between different locales, as in the case of agile localization following the lead of agile development.

Agile development has its benefits: it is fast, releases happen often, developers work in teams, and bugs are fixed frequently. These benefits impact agile localization as well: there are daily projects with small word counts to translate; the strings are often missing comments or visual context; and if a localizer is not part of the development from the beginning, the lack of information (given that agile development lacks proper documentation) might leave results up to guesswork.

As this paper has tried to show, agile localization has its drawbacks. If the issues with missing context and teamwork could be remedied, agile localization would surely improve the localization of various software products into other locales. But in order to remedy the shortcomings, localizers as well as developers need to understand the ways languages work and not see localization as just an afterthought. While it is true that developers and localizers live "in different worlds" [1], communication is key in order to bring a perfect product to market and minimize additional language-oriented fixes.

It is now clear that companies need to think about localization from the very start of product development, and they need to provide adequate tools (e.g., visuals or textual information) for localization processes. This need is even stronger in agile localization. Although "agile is the new black" [17], if the end client or developer is not ready to fully adopt an agile localization approach (even if they use agile development) with all its peculiarities, it might be best to wait and prepare, because context, communication, and inclusion are key in agile localization.

8 References

[1] M. Ressin, J. Abdelnour-Nocera, and A. Smith, "Of code and context: collaboration between developers and translators," In CHASE 2011 Conference Proceedings, 2011, pp. 50–52. https://doi.org/10.1145/1984642.1984653

[2] M. Ressin, J. Abdelnour-Nocera, and A. Smith, "Lost in agility? Approaching software localization in agile software development," in *XP 2011, LNBIP 77*, A. Sillitti et al, Eds. Berlin: Springer-Verlag, 2011, pp. 320–321. https://doi.org/10.1007/978-3-642-20677-1_25

[3] M. Ressin, "Defects and agility: localization issues in agile development projects." In Agile Processes in Software Engineering and Extreme Programming – 12th International Conference.

[4] M. Ressin, "An empirical examination of interdisciplinary collaboration within the practice of localization and development of international software," PhD. thesis, University of West London, 2015.

[5] B. Esselink, *A Practical Guide to Localization*. Amsterdam/Philadelphia: John Benjamins Publishing Company, 2000.

[6] J. Roturier, *Localizing Apps. A practical guide for translators and translation students*. London/New York: Routledge, 2015. https://doi.org/10.4324/9781315753621

[7] A. Pereverzevs, "Agile localization: the complete guide," 2019. [Online]. Available: https://lokalise.com/blog/agile-localization/. [Accessed Sept 29, 2022].
[8] Phrase. "Agile localization: the only guide you'll ever need," 2020. [Online]. Available: https://phrase.com/blog/posts/translation-management-agile-localization/. [Accessed Sept 29, 2022].
[9] K. Trusava, "Agile localization: what is it and how is it managed?," 2020. [Online]. Available: https://dzone.com/articles/agile-localization-what-is-it-and-how-is-it-manage. [Accessed Sept 29, 2022].
[10] Martin, Robert Cecil (2002). Agile software development: Principles, patterns, and practices. USA: Prentice Hall.
[11] S, Alsaqqa, S. Sawalha, and H. Abdel-Nabi. "Agile Software Development: Methodologies and Trends," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 11 (Jul. 2020), pp. 246-270, 2020. https://doi.org/10.3991/ijim.v14i11.13269
[12] D. Rigby, J. Sutherland, and H. Takeuchi, "Embracing agile," 2016. [Online]. Available: https://hbr.org/2016/05/embracing-agile. [Accessed Sept 29, 2022].
[13] Agile Manifesto Signees, "Manifesto," 2001. [Online]. Available: https://agilemanifesto.org/principles.html. [Accessed Sept 29, 2022].
[14] Pulse, "Capturing the value of project management," 2015. [Online]. Available: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2015.pdf. [Accessed Sept 29, 2022].
[15] E. de la Cova, "Translation challenges in the localization of web applications," *Sendebar*, vol. 27, no. 1, pp. 235–266, 2016.
[16] M. Á. Jiménez Crespo, "The intersection of localization and translation: a corpus study of Spanish original and localized web forms," *Translation and Interpreting Studies*, vol. 5, no. 2, pp. 186-207, 2010. https://doi.org/10.1075/tis.5.2.03jim
[17] M. Magdyiary, "Why we need to adopt an agile approach to translation and localization," 2021. [Online]. Available: https://www.gala-global.org/knowledge-center/professional-development/articles/why-we-need-adopt-agile-approach-translation-and. [Accessed Sept 29, 2022].

9 Author

Marián Kabát is an assistant professor at the Department of British and American Studies, Faculty of Arts, Comenius University in Bratislava, Slovakia. He teaches translation courses (both literary and specialized translation). His research focuses on localization (of software, websites, and video games), machine translation, and post-editing. In 2020 he was awarded the Rising Star Scholarship by GALA. Other than being an avid teacher and researcher, he is also a practicing translator and mostly localizer of various software products.

Article submitted 2022-09-24. Resubmitted 2022-11-14. Final acceptance 2022-11-15. Final version published as submitted by the author.