New horizons in automation of translation QA

Mikhail Kudinov, Palex, Ltd

Mikhail Kudinov is a senior software developer at Palex, Ltd. In 2010, he was invited to Palex to make a dream of developing a proprietary translation quality assurance tool true. Since that, he's been working on making quality assurance easy, fast and safe. His professional interests include machine learning, human-computer interaction and agile software development.

Abstract

The market of automated translation quality assurance tools offers quite a limited set of applications with the same well-known functionality as they offered a few years ago.

The purpose of this paper is to review the automated translation quality assurance process, to describe several improvements we implemented in Verifika® and to provide measurement of their efficiency.

In this paper we will briefly examine the role of translation quality assurance tools and the most common scenarios of their use. By looking at these activities from the viewpoint of a user interaction designer we will find improvements to make quality assurance faster, more convenient and more accurate. The measurement for implemented improvements effect will be also provided.

This paper focuses only on those translation quality assurance tasks that may be formalised. Linguistic quality assurance is beyond its scope.

Key words: translation quality assurance, usability, user interactions, translation memory tools, computer-assisted translation, computer-aided translation, bilingual files.

Introduction

While translation memory tools (computer-assisted translation or CAT software) make a great help to save translators' time and language service providers' money, quality assurance continues to be a time-consuming part of delivering high quality translations.

Translation quality assurance tasks could be easily grouped into two categories. The first category requires human intervention to ensure that text looks as if it was originally written by a native speaker of the target language. The second category is about formal requirements for the translation. They include spelling according to dictionary, glossary compliance, translation consistency, pattern-based checks.

Formal requirements are quite easy to automate. As a result, some types of these checks were included into the majority of translation memory tools; several standalone utilities for these purposes are also available. There were a number of papers comparing their capabilities ([2], [3]). Consulting these papers shows that existing QA tools have quite similar functionality and could help to save time on QA tasks, but routine operations performed by humans are still a necessity. Avoiding waste operations would lead to improved performance of reviewers and possibility to deliver more accuracy for the translation in less time.

Reasons for quality assurance of bilingual files

When translation is performed with the help of a translation memory tool, the source text passes through several stages.

First, it is split in text chunks (segments) to allow translating them one-by-one. Then the software creates special bilingual files that contain both the source text and the translation for each segment. Initially the translation can be empty, copied from source or taken from older translations. Then the translator completes translation, which can undergo review and at the end, translated files containing only translated text are created.

Review could be performed both at the stage of bilingual files and at the final stage, but performing it at an earlier stage has a number of benefits.

- To compare the source text with translation and easily find inconsistency.
- To focus on text content instead of giving attention to all aspects of the final document.
- To resolve issues and update translation memory to avoid repetition of the errors in the future.
- To improve text corpus for machine translation training.

The market of quality assurance tools

The importance of bilingual files review for formal errors led to creating several standalone QA tools and adding these capabilities to translation memory software. We included questions about QA tools usage into our trial license application form to help us understand the current state of the market.

Of 212 valid responses, the results are the following:

- 42% use quality assurance features built into their CAT software.
- 30% use ApSIC Xbench.
- 15% use regular expressions.
- 6% use Yamagata QA Distiller.
- A few people use other tools (Okapi CheckMate, D.O.G. ErrorSpy, internal software).

Currently 15% of the respondents use Verifika in their QA workflow.

The results show that ApSIC Xbench is the most wide-spread QA software, and Yamagata QA Distiller is the most popular commercial one. And it seems they are leaders of the standalone QA tools' market.

QA workflow: current state and further improvements

The most common scenario of formal quality assurance includes selecting files and types of segments we want to verify, setting up all necessary checks and running them. Then the user resolves them one-by-one. After resolving all errors, he/she should re-check the files to make sure no new errors were introduced.

Resolving errors is the most time-consuming part of the quality assurance process, and we analysed what operations a user performs at this stage.

Market leaders (ApSIC Xbench and Yamagata QA Distiller) suggest the following workflow:

1) Select next reported issue.
2) Decide if it is a problem or not.
3) If it is a problem, double-click the error to launch bilingual file editor.
4) If it is not a problem, ignore it and go to step 1.
5) Place cursor on text that requires correction.
6) Complete correction.
7) Return to report and go to step 1.

Thus, if we want to reduce time on error resolution, we need to make these steps shorter or eliminate some of them. Alan Cooper ([4]) suggests the following criteria for separation of concerns between human and computer.

“The computer does the work and the person does the thinking.”

A similar principle was the basis of the computer-aided translating concept introduced by Martin Kay [1].

Regarding the QA area, it means that the person should decide if something is an error or not, and how to correct it. And the software should do everything it is capable of doing.

It sounds quite obvious, but following to this concept is not so easy.

In our case, the second step is a necessity because a human should decide what to do in each particular case. But virtually all the remaining steps can be safely removed to increase productivity.

First possible improvement is to provide instant editing without launching an editor. A user can see an error and click the highlighted error place to resolve it. This way we can remove steps 3 and 7.

The second improvement is to provide auto-correction for all issues that the computer could correct. In this case the user can see suggested corrections and confirm them instead of resolving manually. In this case we replace steps 5 and 6 with one “Confirm correction”.

And the third improvement is to go to the next error when the user corrects or ignores previous one. It removes step 1.

The final workflow for errors with auto-correction is the following:

1) Decide if the current issue is a problem or not.
2) If it is, confirm auto-correction.
3) If it is not, ignore it.

When we decided to implement these improvements, we needed to solve the following problem. As soon as we modify a segment that contains errors, the place of these errors and their existence become obsolete. Thus we should mark obsolete errors or refresh the report according to changes.

Kilgray memoQ team opted for the first way. It means that when you modify a segment containing errors, they keep them in report but remove any highlighting from the segment. Also, auto correction is not possible because of this modification.

We decided to implement the second way. It means that we refresh the report after each segment modification. In our tool the report always reflects the current state of segments, all errors are highlighted and can be auto corrected properly. Thus, a user does not need to re-check the same files after correction to find newly introduced errors. We implemented this re-checking function for all types of errors (spaces, numbers, spelling, terminology, inconsistency etc.)

Time measurement

It is quite obvious that improvements described above help to save time. We decided to measure this effect. To achieve it, we prepared a limited version of our tool that does not allow auto-corrections and provide editing only via launching a bilingual file editor. We intentionally did not compare Verifika to other tools as we wanted to measure the effect of only the above-mentioned improvements.

We trained people to use Verifika in limited and full versions. Then we selected a typical project which contained about 1000 detected errors. The same person corrected the project in the full version and in the limited one. We allowed a two weeks delay between these two sessions to ensure that the person does not remember earlier corrections and the project looks almost “new” for her. We measured only the time of resolving errors.

	Time, seconds
Full version	2,353
Limited version	3,082

Thus, time reduction for this project was about 24%. Of course, saved time depends on the type of project, user skills and false positives percentage, but it still should be significant in most cases.

Also, removing unnecessary repetitive actions leads to better user experience in all cases.

Future improvements

The other possible way to reduce time of resolving errors is to reduce the number of decisions that the human needs to make. We could make it possible in the following ways:

1) Reduce the number of false positives.
2) Correct or ignore several errors with one decision.
3) Remember older decisions to apply them automatically for new projects.

The first way is to analyse projects in different areas and languages to find common false positives and prevent them. We perform it regularly to improve the quality of our reports. However, the huge amount of translations makes it almost impossible to remove all false positives for each project type.

The second way is to provide a user the way to resolve or ignore several errors at once. It requires providing enough information to safely confirm a number of corrections simultaneously. We could allow user to correct all the same segments, all segments where the text before or after the error is the same etc. Currently we are investigating the efficiency and the best user experience for this approach.

The third way is to save decisions a user made in older projects to apply corrections or ignores to current project. This approach requires creating a base of quality assurance decisions and distinguishing reusable ones from project- or segment-specific those. Our research proved that similar problems occur in different projects quite rarely, so now we are analysing the efficiency of this approach for series of similar projects.

Conclusion

Existing translation memory tools vary in the level of quality assurance capabilities. The advantage of built-in solutions is that they are integrated to an environment the user already learned. On the contrary, standalone quality assurance tools provide consistent user experience across different file formats. Additionally, they can provide more checks and better user experience of error resolving, because these are the main goals of their interface. But it requires investigations of interaction designers to provide the most productive and convenient solutions. Verifika has proved the effectiveness of several improvements in resolving errors and we continue researching the best user experience regarding the QA and improving the tool accordingly.

References

1. Martin Kay (1997), The Proper Place of Men and Machines in Language Translation, Machine Translation Volume 12 Issue 1/2, p. 3 – 23.

2. Andrey Gerasimov (2007 Jan.-Feb.), Review of Translation Quality Assurance Software, Multilingual Magazine, p. 22.

3. Julia Makoushina (2007 Nov.), Translation Quality Assurance Tools: Current State and Future Approaches, Translating and the Computer 29.

4. Alan Cooper (2007), About Face 3: The Essentials of Interaction Design (with Robert Reimann and David Cronin) (ISBN 0-4700-8411-1).