Localizability and World-Readiness for Software

WOJCIECH FROELICH
CTO – ARGOS MULTILINGUAL

Short Description

- What are localizability and World Readiness?
- Find out how building localizability into the front end of your development process can get your software on the road to international success.
- Discover the critical steps to improve the localization process for your software:
    - Pseudo-localization
    - Giving your language partners what they need to succeed

ARGOS multilingual

Localizability and World-Readiness for Software

Sleepless Nights

When a software development team is in the product design phase, it is easy for localization to be overlooked. Development teams are busy; gathering user requirements, developing components and testing usability in the source language (probably English) – until the moment when the Product Manager says:

"OK, this is great! Let's sell it in 15 countries"

If localizability hasn't been built into the product, developers can be hurled into a costly and time-consuming spiral of retrofitting global needs onto a product, delaying global release cycles and creating inelegant, provisional workarounds which add unnecessary complication to both native and international release processes. What seemed like a great idea - to amortize development costs more quickly through global expansion - can rapidly become the source of sleepless nights for both the Product Manager and Development team.

The ultimate goal of software development for the global marketplace should be to create a "world-ready" application. This process is often referred to as internationalization, and ensures that software can be run anywhere in any language. This means the application should be able to support any locale, any regional time and date or numbering formats, as well as ensuring the application is ready to be translated into any language, even those using different scripts and alphabets or bi-directional languages such as Hebrew or Arabic.

The key to localizability is the design of software and resources that enable a software localization process which does not require re-engineering or modifying code after translation processes have been completed.

The first step on the road to a "world-ready" application is achieving the clear separation of User Interface (UI) resources from functionality-related resources. The ideal scenario is to achieve this separation using file formats that suit localization teams – enabling both the use of professional translation tools and potential automation, which becomes a key element when developing and localizing in agile environments.

The second step is to be able to measure the success of any localizability efforts. The easiest way to confirm that a product can be easily localized is by running localizability testing tasks.

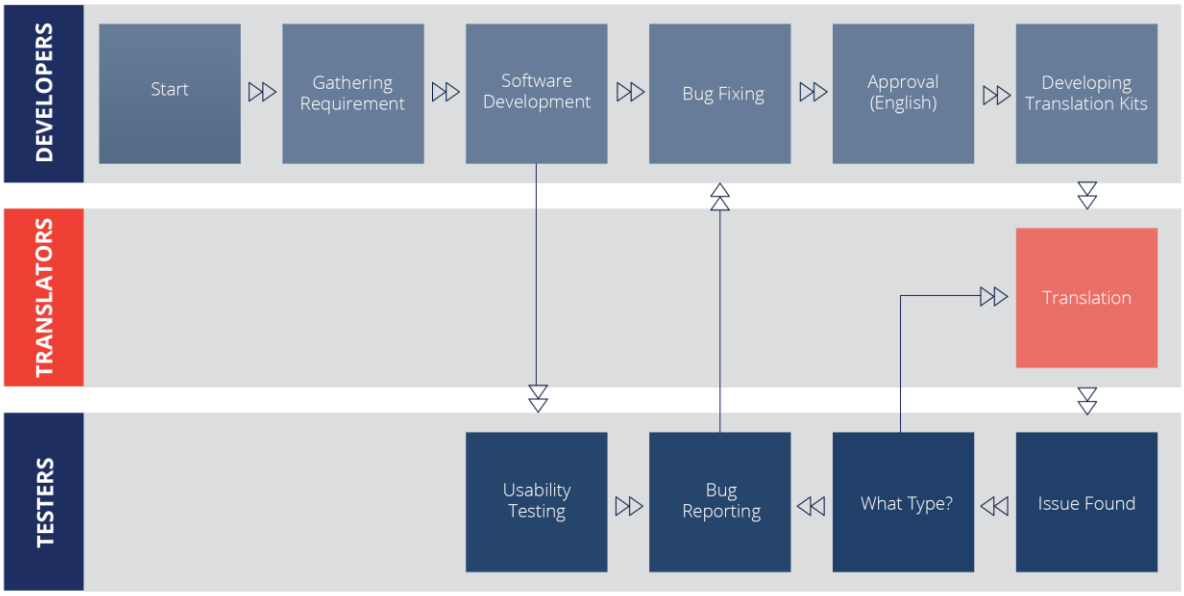
Function and Process

There are typically three interdependent functional teams involved in launching a global product: Developers, Translators and Testers.

As the following process flow illustrates, usability testing takes place prior to launch of software in its original language, but also after translation. Localization testing can result in two types of issue:

- Translation issues can be fixed by getting the linguist to amend the translation. This type of issue is normally either because the original translation is found not to reflect a string's real meaning in the context of the running application or where a string needs to be shortened to match the space available within the User Interface.

- Functional issues are those issues which need to be routed back to developers, and which cannot be fixed by making a linguistic change. These can range from bugs which are introduced because of the translated language, string length issues which cannot be fixed by reducing the length of a translation or functional issues which were not apparent in the English build.



Context is King

It might seem self-evident, but the key to a successfully localized software product really is to enable linguists to do the best possible job.

Professional linguists are normally highly qualified; as well as having a degree in translation, they will often have many years of professional experience before having specialized in a particular area. But even the best linguists need to have a solid understanding of the original product if they are to produce a great translation.

Reducing the number of functionality issues relating to world-readiness is one of the factors which will allow linguists to get the translation right first time.

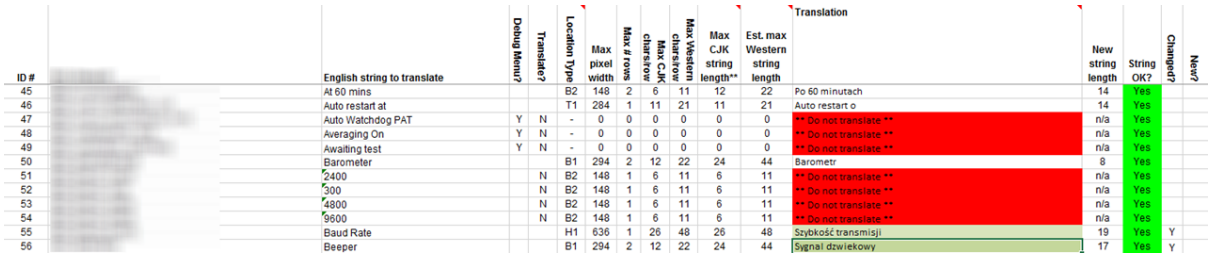
When it comes to giving linguists what they need to do the job correctly, context is the single biggest factor. In software localization, context is KING. Provide as much context information as possible, and your product will reap the benefits.

The perfect scenario is a full visualization of the UI during translation, but other elements can also be very helpful:

- String identifiers, if they are meaningful, can provide additional context information.
- Comments from developers can be a great source of information, especially for cryptic UI messages.
- Length limitation information; note that a "number of characters" limitation will work only with fixed-width fonts. When using proportional fonts, the linguist will need to know the actual width of a string in pixels compared with the length limitation.

In examples where localizability has not been built into the development process and where the Product Manager decides that the product should be sold globally, a developer is often quickly reassigned to produce "translatable" resources. In practice, these developers are very often tempted by easy looking formats like Excel files, where linguists are expected to insert translations into their language's column parallel to the English string. Once translation is completed, a developer then needs to copy and paste translations back into the code. It is worth mentioning that it is rare that hurried copy and paste operations in an unfamiliar language are done right first time!

The following is an example of such a translatable file in Excel format:



ID #	[redacted]	English string to translate	Debug Menu?	Translate?	Location Type	Max pixel width	Max # rows	Max CJK chars/row	Max Western chars/row	Max CJK string length**	Est. max Western string length	Translation	New string length	String OK?	Changes?	New?
45		At 60 mins			B2	148	2	6	11	12	22	Po 60 minutach	14	Yes		
46		Auto restart at			T1	284	1	11	21	11	21	Auto restart o	14	Yes		
47		Auto Watchdog PAT	Y	N	-	0	0	0	0	0	0	** Do not translate **	n/a	Yes		
48		Averaging On	Y	N	-	0	0	0	0	0	0	** Do not translate **	n/a	Yes		
49		Awaiting test	Y	N	-	0	0	0	0	0	0	** Do not translate **	n/a	Yes		
50		Barometer			B1	294	2	12	22	24	44	Barometr	8	Yes		
51		2400		N	B2	148	1	6	11	6	11	** Do not translate **	n/a	Yes		
52		300		N	B2	148	1	6	11	6	11	** Do not translate **	n/a	Yes		
53		4800		N	B2	148	1	6	11	6	11	** Do not translate **	n/a	Yes		
54		9600		N	B2	148	1	6	11	6	11	** Do not translate **	n/a	Yes		
55		Baud Rate			H1	636	1	26	48	26	48	Szybkość transmisji	19	Yes	Y	
56		Beeper			B1	294	2	12	22	24	44	Sygnał dźwiękowy	17	Yes	Y	

As you can see, the developer has put a great deal of effort into creating the file and has included useful information such as where the string is found and maximum string lengths. No doubt, the developer is very proud of the work carried out, and believes that the file will be a great help for translation teams.

Although the file might look cumbersome, the developer has actually done a decent job in this example; it is significantly better than many localization files received even 10 years ago!

In reality, though, these formats are not easy to process in a translation workflow, and it is worth getting developers and localization engineers talking together at the same table in order to discuss the available options. In most situations there are better ways to handle the UI files. These days, linguistic technologies are far more advanced and a little forethought can result in a much smoother localization process.

Where should you start? Usually, the best place to begin is the software development framework being used.

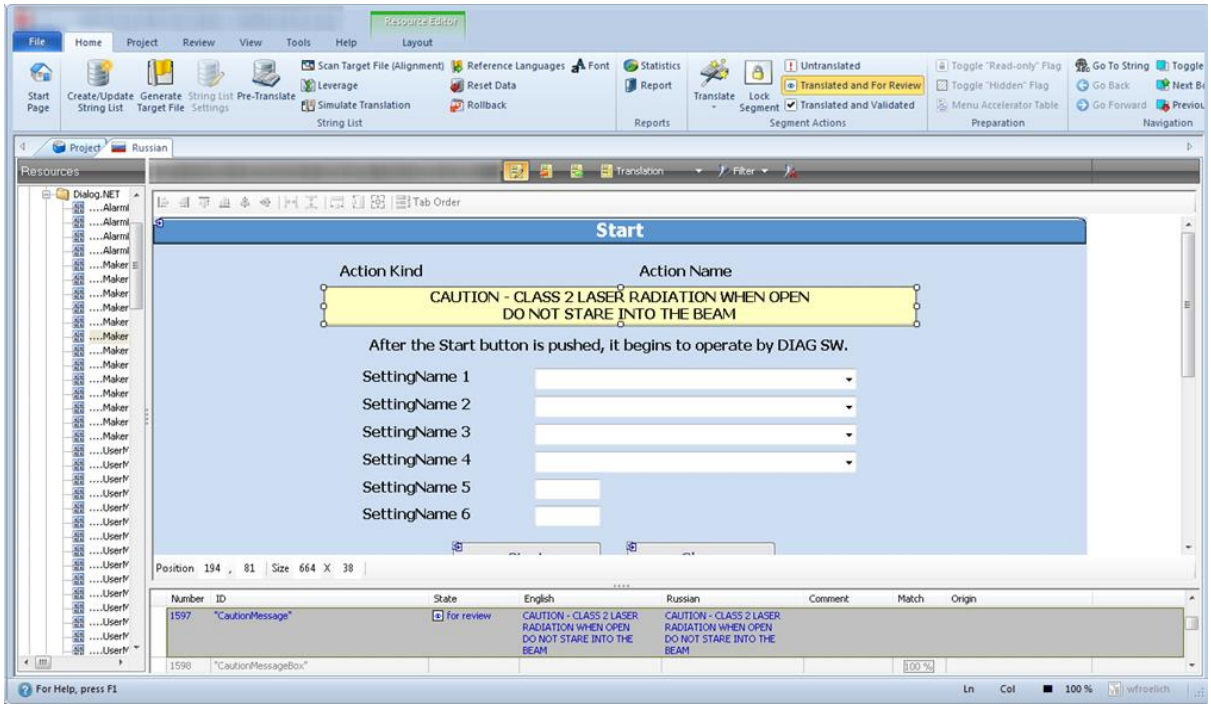
Most modern software development frameworks provide tools to extract UI content from code and guidelines on how to create code that can be easily processed for localization. These tools produce files in formats supported by translation tools like .resx, .properties, .po and .ts files.

Processing UI content with translation tools gives linguists access to many advanced features like translation memory, terminology databases and automated quality checks right at the translator's fingertips as part of the translation environment. These features help speed up the production of high quality translations.

Some frameworks even enable linguists to preview UI while translating. This is usually the best possible scenario – translators see the translation immediately in the right context, they see what type of element is being translated (button, menu item, text field label), they can also see the potential space limitations.

Unified formats for UI translation are a key element to automation of hand-off/hand-back processes in agile workflows. Without automation, agile localization processes can't integrate with agile development processes, and there is no automation if formats are not clearly defined.

The following is a preview generated from a .resx Windows Form by a popular translation tool. As you can see, there is lots of useful information available for linguists, including the string ID, development comments as well as a preview of the dialog, which will help linguists to understand how a dialog is used, and whether there are any string length limitations.



Even custom XML files can be previewed. Because XML files can be easily processed for translation, additional info such as comments about usage or context can also be included for linguists, giving them the best possible chance of getting translations right first time.

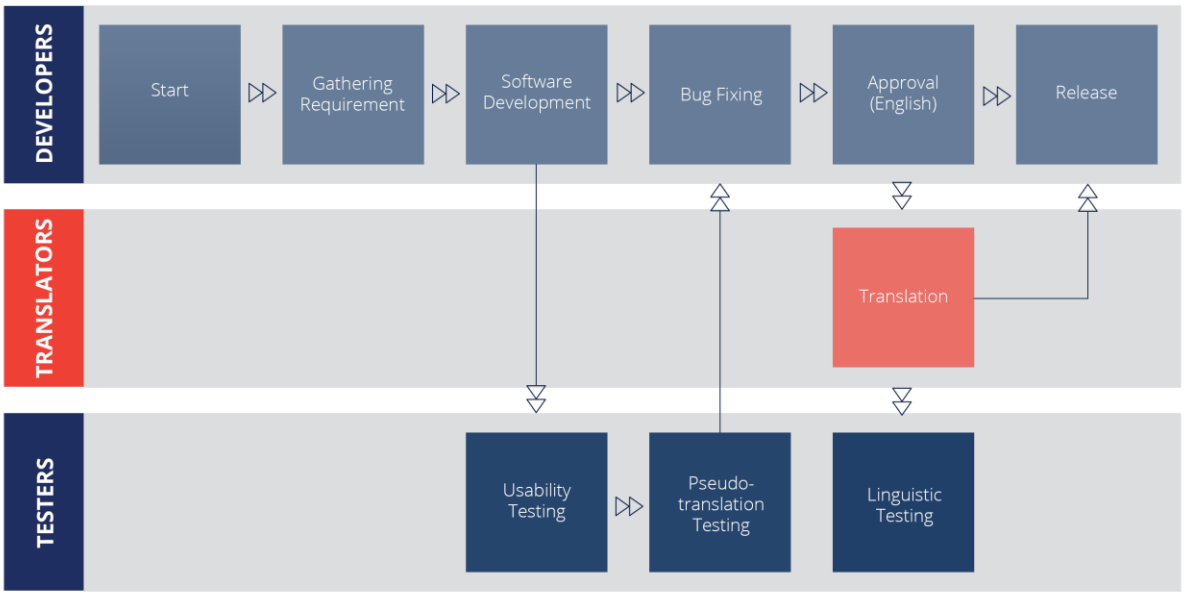
A Proactive Localizability Development Model

The problem with incorporating localization late in the development process is that it complicates the process of resolving language-related issues that are discovered during or after translation, and at localization testing. These issues are likely to require some development re-work and are likely to happen at a very critical moment in the timeline – right before the release. A study by Systems Science Institute at IBM found that the "cost to fix an error found after product release was four to five times as much as one uncovered during design".

For localized products, these costs have the potential to be multiplied by the number of target languages.

If localizability testing isn't carried out before English approval, each testing team (for each target language) will spend time discovering and reporting the same issues, and will then each need to run regression-testing on the same bugs. This can be avoided if localizability testing is brought forward in the product development cycle.

If we modify the earlier process a little - by running localizability testing in sync with usability testing- we can avoid these issues altogether. The revised process flow below shows a pseudo-translation testing phase in orange.



This means developers will exchange localizable files with the translation team even before source content is approved.

Introducing pseudo-translation testing helps to identify language-related issues early in the process, so there is still enough time to go back to developers and fix any language-related issues.

Once the source content is approved, it can be translated and linguistically tested safe in the knowledge that the number of issues requiring developer intervention has been minimized because they were already discovered and fixed much earlier in the process.

Pseudo-localization

So what is pseudo-localization? It is a very rapid and cost-effective method of mimicking the effects of the translation process without the costs and scheduling impact of a real translation.

Pseudo-localization simulates the translation of all resource strings, and usually includes:

- Replacing some English characters with similar non-English characters.
- Adding extra characters to simulate text expansion.
- Adding prefixes and suffixes to mark the boundaries of a string.
- Using Unicode characters for all resources.

All these changes can be automated, so all UI resources can be modified, using similar patterns making issues quick to identify in newly built resources.

Pseudo-translated resources can be used to rebuild an application and test it. The way it functions should be the same as the original version.

Thanks to the patterns used while pseudo-translating resources, it is easy to detect the following issues:

- Strings that were hard-coded and were not moved to the localizable resources.

- Strings that were built from other strings by concatenation or replacing some parts of the string – these strings will often not work in languages different from the source language.
- Any Unicode-incompatible functions that process or display text.
- Any other display issues like incompatible fonts, truncated strings or unexpected space limitations.

Code reviews, the future and learning from the past

Just as complex projects should be followed by a post-project review, it is worth closing the development loop by creating and maintaining code development guidelines that include localization-related guidelines. These guidelines should reflect results of code audits and localization process issues that were fixed in past. For projects with large volumes of code, there are third-party tools that can automate the code review process. What's more, localization providers will normally be happy to participate in this process too, as it will help them to perform great work first time around in the future.

In a nutshell, world-readiness is perfectly compatible with the development process as long as you take steps to bring your localization partners on board early, perform pseudo-translations to test localizability before translations begin and consider the best format to allow linguists to perform their job.

All this will mean that everyone can enjoy a peaceful night's sleep when your Product Manager asks to sell into multiple locales. You will be able to answer with confidence:

> "Sure. No problem, let's do it!"