SOFTWARE LOCALIZATION (L10N) QUALITY ASSURANCE FROM THE TESTER'S PERSPECTIVE

Carola F. Berger
CFB Scientific Translations

Abstract:

The presentation will give a very brief overview over the localization process for software and mobile apps before covering the quality assurance process in detail. The talk will not only discuss the fundamentals of the testing process, but will also help you become a better tester by presenting some tips, tricks, best practices, and pitfalls. The presentation is intended for beginning and intermediate localization testers and discusses the testing process from the tester's perspective. The talk will not cover other steps in the localization process such as file preparation, translation, etc., which have been covered at past ATA conferences.

1 Introduction

In the following, I will discuss the **localization** process for software applications, whereby software applications, or **apps**, refer to computer programs that carry out operations or tasks for a specific application. Traditionally, apps were executed on desktop computers, but lately more and more apps run on mobile devices or over a network (in the "cloud"). In the following, I will refer to all of these types of programs collectively as apps or alternatively simply as software.

According to a recent study, the worldwide software revenue totaled around 407 billion USD in 2013 (Gartner, 2014), whereby the mobile app market accounts for about 25 billion USD (Wall Street Journal, 2014). While the software market enjoys a yearly increase of a few percent, the mobile app market was expected to rise over 60% in 2013 compared to 2012, according to the above sources. These growing markets also mean a greater need for internationalization and localization of the software.

Below I will briefly outline the entire **internationalization** and localization process before discussing one crucial step in more detail: the QA testing process. QA testing is an important step, and many large software companies employ in-house linguistic testers or contract out the task to full-time linguistic testers. I will explain the entire process and present tips and tricks at the example of fictional software applications. After a brief summary, I include a glossary and references. **Glossary entries** are denoted in **bold** on their first occurrence.

2 Overview over the Internationalization and Localization Process for Software Apps

Figure 1 illustrates the localization process for software applications, from the linguistic and software design preparations to the final product. The first step in the process is the preparatory phase, which includes the so-called internationalization phase (abbreviated **i18n**), and the preparation of style guides and glossaries. Internationalization is the process of preparing the software such that it can be translated and adapted to a specific **locale** without design changes. Here, the term locale denotes a set of parameters such as currency symbols, number formats, date and time formats etc. for a specific country or region. Internationalization needs to take into account not only these various formats but also the ability to accommodate languages with different letter types (Roman versus Chinese characters, for example) and/or directionality such as left-to-right versus right-to-left. This step is the task of software engineers, for further technical details I refer to the list of references at the very end. The linguistic step of the preparatory phase includes the development of style guides and glossaries to ensure consistency in terminology and voice and the identification of potential challenges during the localization step. For details on this step I refer to the excellent summaries in (Bodeux, Whitty 2013), (Bodeux, McKay, Whitty 2014), and also (Esselink, 1998).

The next step is the translation and review of the **software strings**. The translation can be prepared in various forms, from a simple Excel table to online translation tools that display the string in its original context. Context can be very helpful for the translator, since software strings are generally very short and sometimes consist of only one word, whereby it can sometimes be hard to distinguish a verb from a noun without the necessary context. Here, adequate naming conventions of the **identifiers**, that is, the names of the string variables, can help tremendously,

if these are supplied to the translator along with the strings to be translated. It should also become clear now why the subsequent linguistic QA testing is so important, because during that step the translation will be reviewed in the full software context.

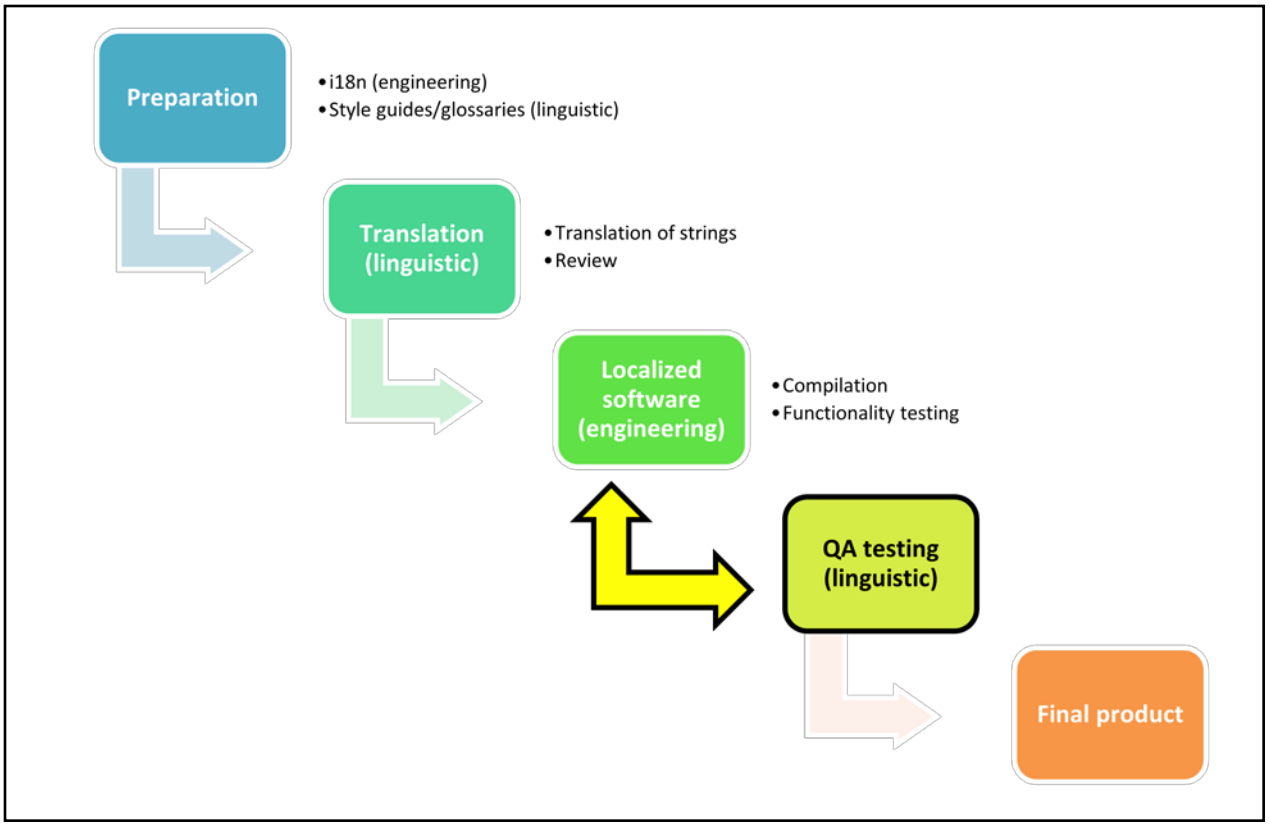

Figure 1: Software localization process flow (adapted from (Bodeux, Whitty 2014))

In the next step, the software engineers read the translated and reviewed strings into the software and produce a localized program version. They also test for functionality, that is, whether the imported software strings broke the code due to leftover special characters as artifacts of the translation process or due to leftover or missing code that was mistakenly added or left out of the translation. Furthermore, a QA test plan, or **script**, for the next step is usually developed during this step or earlier in the process. Occasionally, the QA testers develop the test script themselves as part of the next step.

The fourth step is the linguistic and functional testing by a language expert. This functional testing goes beyond the preliminary testing performed in the previous step and will be discussed in detail in the next section. Here, the QA testers follow the QA test plan developed earlier. The QA plan guides the linguistic QA testers through the entire app and ideally encompasses all scenarios that could possibly be encountered by an end user located in the target country or region. If errors, usually called **bugs**, are present, the testers file bug reports to inform the software engineers about the problem and, if applicable, submit corrected strings. Then the software engineers recompile the software, produce a new so-called build, and send the (hopefully corrected) programs back to the testers. This cycle repeats until no more bugs are found, and the final product is released. The testing is frequently done on site at the premises of

the software manufacturer, for confidentiality reasons or to test apps on new devices that are not yet publicly available and have been set up specifically for this purpose.

Concurrently, manuals, help files, marketing material and other associated documents also need to be translated and localized. Further information about this aspect can also be found in (Bodeux, McKay, Whitty, 2014) and (Esselink, 1998). The entire localization process for specific types of apps is also discussed in (Seeburg, 2012) and (Niedermair, Dietz, 2013).

In the following, we will look at the QA testing step and the interplay with the engineering stage in more detail.

3 Software Quality Assurance

The software QA process can be summarized as follows: QA testers try their best to "break" the localized software in order to locate bugs, which are then eliminated, such that the end users in the target region speaking the target language can enjoy an error-free product.

3.1 The QA testing process

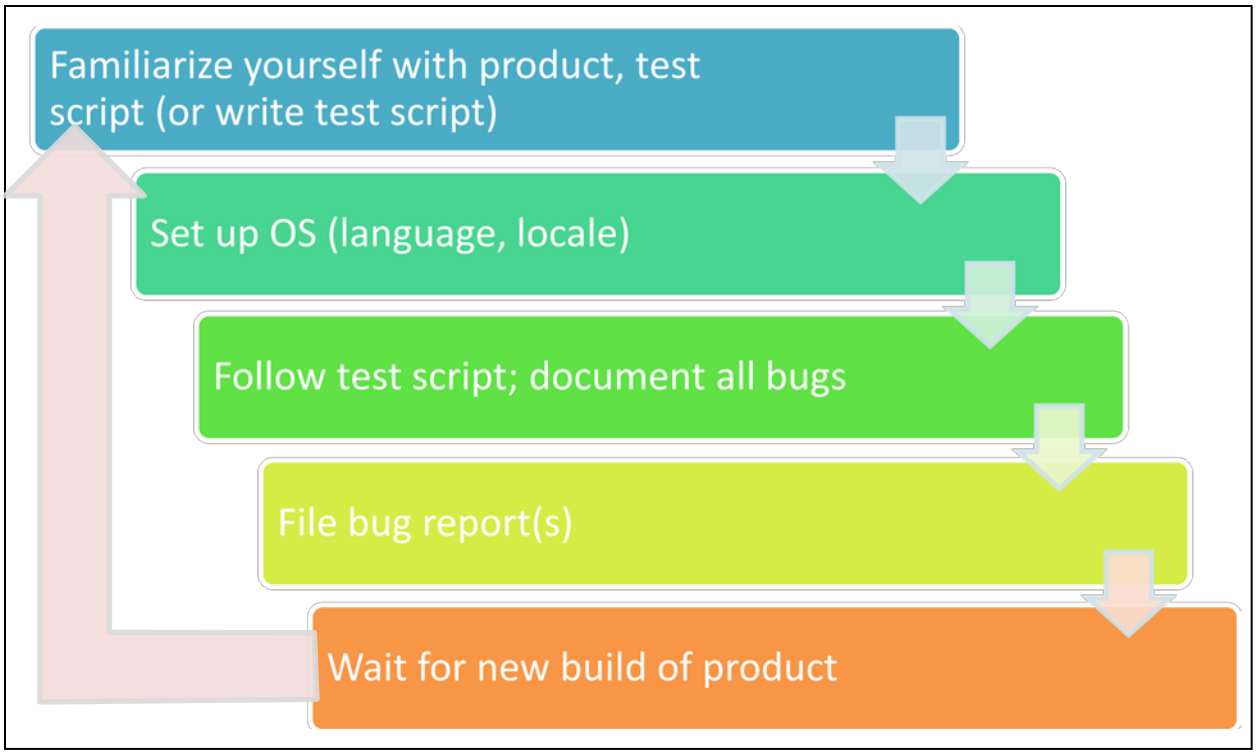

Figure 2: QA testing process flow

In brief, a typical QA testing process looks as follows, as illustrated in Figure 2 above:

1. Familiarize yourself with the product in the source language and read the test script, if the engineers have provided one. If there is no test script, go very carefully through all the screens, buttons, menu items, and error messages of the product in the source language and write a test script in order to test every single screen, button, menu item, message, and error scenario.
2. Set up the **operating system** for testing – this usually requires changing the locale, the language, and other settings as appropriate for the target region. Occasionally, a second device is available for comparison in the source language, but often the comparison is done with the screenshots in the test script.
3. Go very carefully through every single screen and button, following the test script. However, at every step, question whether there is a setting, input, or touch of a button not mentioned in the test script that might cause an error.
   Document all bugs that were found, the precise version of the software and the OS, and the precise steps to reproduce the bugs without referring to the test script. This is important, because test scripts and build versions change. Ideally, every step of the process illustrated in Figure 1 is concluded before the next step is begun, but more often than not the entire process is in flux, and software engineers are already adding functionality to the product while the QA testing of the current build is not yet complete.
4. Send the bug reports to the engineers, following the precise requirements and instructions. Many companies have their own proprietary bug reporting system and it is important to use this system as instructed, whereby it helps to remember that the person who receives the report likely does not speak/read the target language.
5. Wait for a new build of the software without the bugs reported in step 5. Go back to step 1[1] and repeat.

3.2 Problem types

There are several types of problems (see also (Globalme/Golota 2013) that software QA testers look for. (Esselink, 1998) mentions that these problem types are identified in separate rounds of testing. In my experience, however, testers generally look for all problem types in the same round of testing at once, although the problems are clearly classified separately in the respective bug reports.

- **Linguistic problems**
  These include:
  - *Grammatical, spelling, and punctuation errors*
  - *Translation errors due to context*
    The translator sometimes does not have the full context of a single word on a button, and it is sometimes hard to guess whether the source word is a noun or a verb.
  - *Linguistic consistency*
    Is the terminology consistent throughout the product and in accordance with the glossary?

[1] Occasionally, new functionality or new test steps are added as well, and the script changes.

- *Missing translations*
  Is the source language or even the identifier displayed instead of the translation in any part of the interface including buttons, pop-ups, tool-tips, and obscure error messages?
- *Cultural aspects*
  Does the translation respect local sensitivities and will the end users understand all cultural references?

- **Layout and formatting issues**
  - *Truncations, text expansion in the entire interface including error messages, pop-up windows, text boxes, etc.*
  - *Wrong alignment – especially in left-to-right versus right-to-left languages*
  - *Character corruption issues, wrongly displayed (non-ASCII or accented) characters*
  - *Line breaks*
  - *Number, date/time, currency formats, addresses, zip-codes, phone numbers*
    Some of these problems with formats may also be functionality issues, as discussed next.
  - *Symbols and icons*
    Do symbols and icons respect the cultural sensitivities of the target population?
- **Functionality problems**
  - *Link, button, menu functionality*
    Do all links, buttons, menu items function as intended in the localized environment?
  - *Input/output validation*
    Do the input fields allow proper entries in the units/format as customary in the target region? For example, in Austria postal codes only have four digits, whereas in Germany they have five, in other parts of the world up to 10. Does the software allow proper entry of these postal codes without error messages or give the correct error message as appropriate for the country/region?
  - *Number, date/time, currency formats, addresses, postal codes, phone numbers, paper sizes*
    Can the software handle these number formats properly? Are the calendars adjusted appropriately? Is there interference between the OS settings and the software settings?
  - *Problems with character display, sorting, or input*
    Are all (non-ASCII) characters of the target language properly handled upon input, displayed, saved, and sorted? Some characters such as & or ~ label keyboard shortcuts for quick access to menu items, others, such as \n, \t, denote line breaks and tab stops. Can the software handle it, if these or other non-ASCII characters are entered into an input field by the user?
  - *Keyboards and shortcuts*
    Can all keyboard shortcuts and control functions be accessed with international keyboard layouts?
  - *Interaction with OS functions such as copy/paste and locale and language settings in the OS*
    Does the software properly handle interactions with the OS, such as

copying/pasting to/from the clipboard? Do the settings for locale and language in the OS interfere with the app?

- **Installation/delivery problems**
  This type of problem is in fact a functionality problem, possibly the most fundamental of all functionality problems, since the end result of a failed delivery is that the app cannot be run on the target system due to an incompatibility. The details that are to be tested depend on whether the software is a desktop app (delivered via installation discs or other media), a mobile app (downloaded through an app store), or a web app (accessed via web browser AKA the "cloud"). Sometimes, this functionality is tested separately from the overall testing process; sometimes it is included as the very first step (after setting up the OS) in the testing process.

3.3 A few concrete examples

Below, I will show some concrete examples to illustrate the problem types discussed in the previous section.

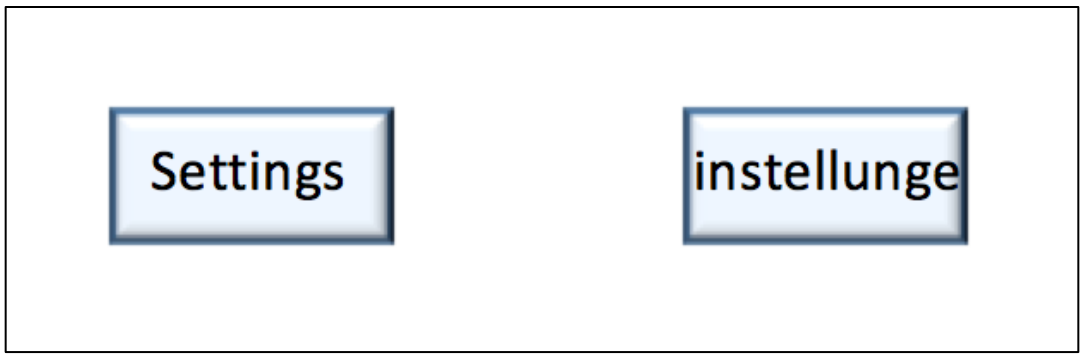

Figure 3: A classic example of a layout problem English > German.

Figure 3 illustrates a classic example of a *layout issue* when the label "Settings" of a button is translated from English into German. The German word "Einstellungen" clearly does not fit on the button, but there is no shorter word with the same meaning. In general, Western languages can take up to 30% more space than English (see for example (Bratu, 2014)). **GUI** designers usually take this into account, but some problems can still arise. In this case, if at all possible, the quickest resolution is sometimes an alternative, shorter translation that conveys the same meaning. If this is not possible, as in the above example, a smaller font may be a possible solution; otherwise, the interface may need to be redesigned.

Another example of a possible problem source is the use of *different number formats* in various regions of the world, as shown in Figure 4. In about 60% of the world, including most English-speaking countries, the decimal point is used as the decimal mark. These countries are colored in blue in the figure. The green colored countries, about 24% of the world's population, use a comma as decimal delimiter. The red colored regions employ Eastern Arabic numerals, and for the gray colored regions there is no data available. These different number formats can lead to functionality problems, especially when the app has its own settings for number formats or expects a certain number format, which interferes with the locale settings in the OS (usually separate from the language settings). It is therefore important to thoroughly test the functionality

of apps that include operations on numbers entered by the user and to try all available combinations of locale, OS, and app settings.

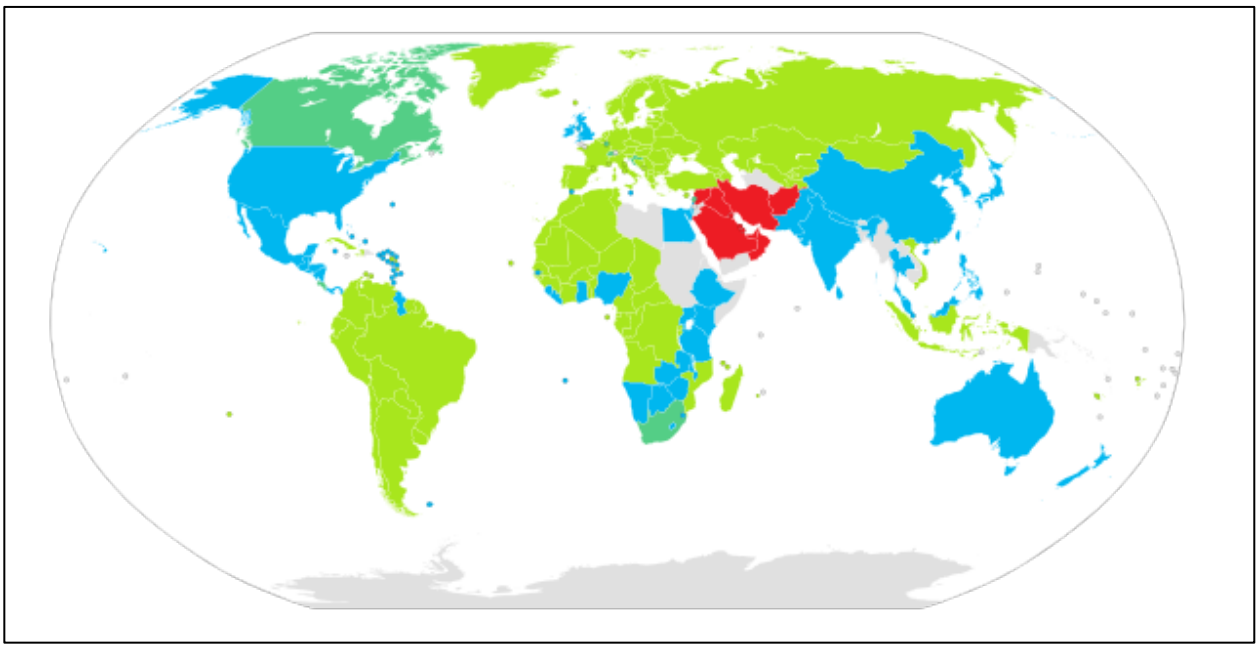

Figure 4: Worldwide map of decimal delimiter usage
(Image source: "DecimalSeparator" by NuclearVacuum - http://commons.wikimedia.org/wiki/File:DecimalSeparator.svg#mediaviewer/File:DecimalSeparator.svg)

Many apps have *built-in functions to validate the data* entered by the user. While this can be tremendously helpful, it can also render an app completely useless. For example, ZIP codes have 5 digits in the US (plus optionally 4 digits in extended ZIP codes). Many programs have therefore a check built in that requires exactly 5 (plus 4) numbers when entering a ZIP code, otherwise an error message will be shown until the user corrects their entry. For an app that specifically targets the US market, the ZIP code check is great. Suppose this app is then subsequently translated into German, to be used in Germany, Austria, and the German-speaking part of Switzerland. In Germany, the “Postleitzahl” (the German equivalent of the ZIP code) has 5 digits, so no problem there. In Austria and in Switzerland, however, postal codes have 4 digits, so there would be a problem with the functionality of the app, since the user would not be able to enter a perfectly valid postal code without receiving an error message. Figure 5 shows a global map of the postal code formats currently in use all around the world. QA testers need to know and test for these and other formats (phone numbers, paper sizes, etc.) that are in use in the target region(s). If a certain language is used in various parts of the world, the testers need to be aware of all these nuances. Sometimes, several testers are available to test for various regional variants, but in some instances, one tester is expected to be able to handle several of these variants, which requires a bit of preparation by the tester.

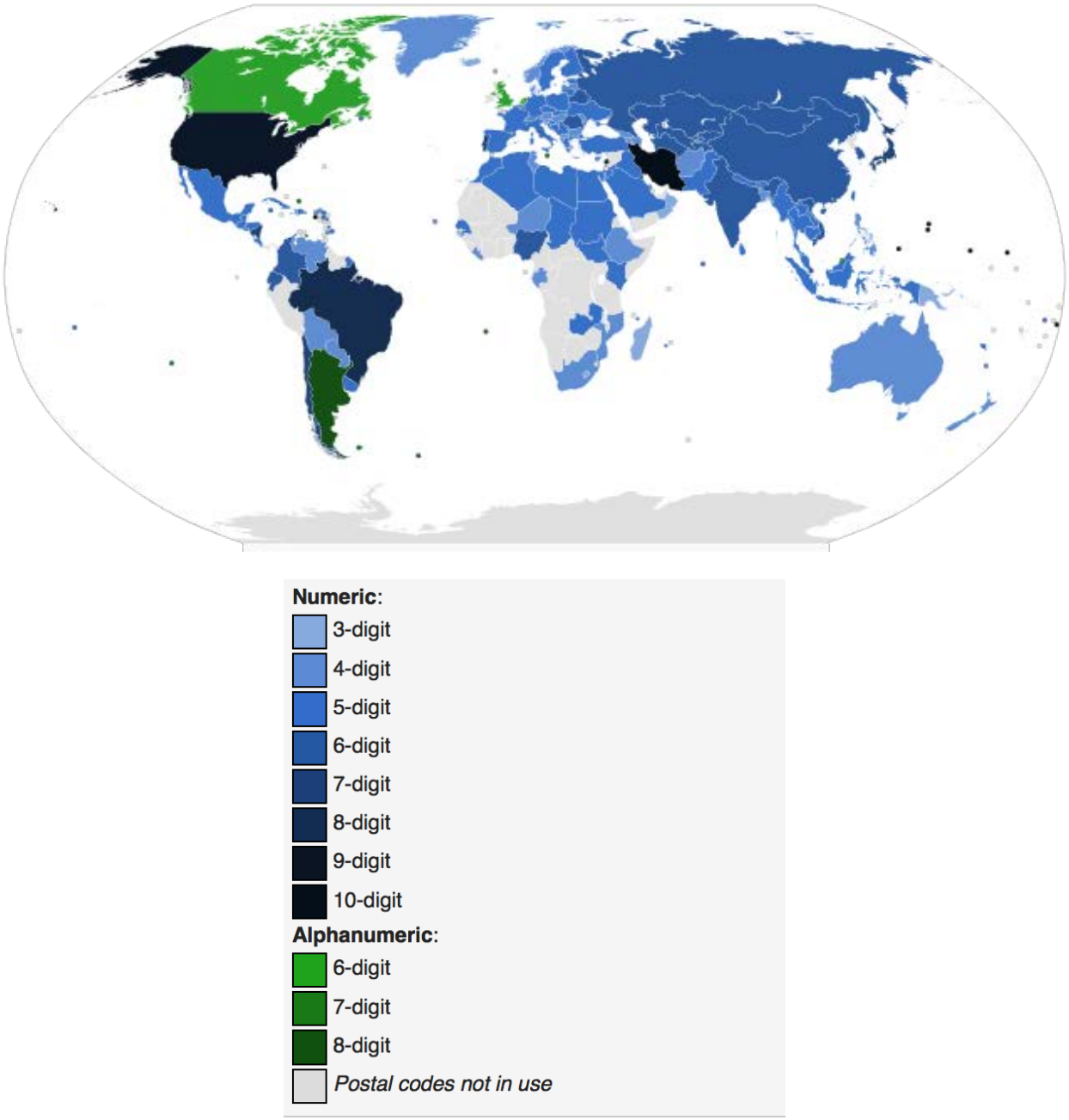

Figure 5: Postal code format by country (Image source: "Postal codes by country" by Stadscykel - Own work. http://commons.wikimedia.org/wiki/File:Postal_codes_by_country.svg#mediaviewer/File:Postal_codes_by_country.svg)

Finally, I want to illustrate a *layout issue that borders on a functionality issue*, which is not trivial to spot. Figure 6 shows the display of a fictitious (and quite rudimentary) app for cycling, displaying the current speed and the distance ridden since the start of the current ride, in English and in German. The German equivalent of the speed display nicely illustrates how a potentially problematic length increase has been dealt with, by decreasing the font size such that the word "Geschwindigkeit" (German for "Speed") is still readable and fits into the allotted space. The

decimal mark as discussed above is also correct, as are the units, kilometers per hour instead of miles per hour for the German speaking market.

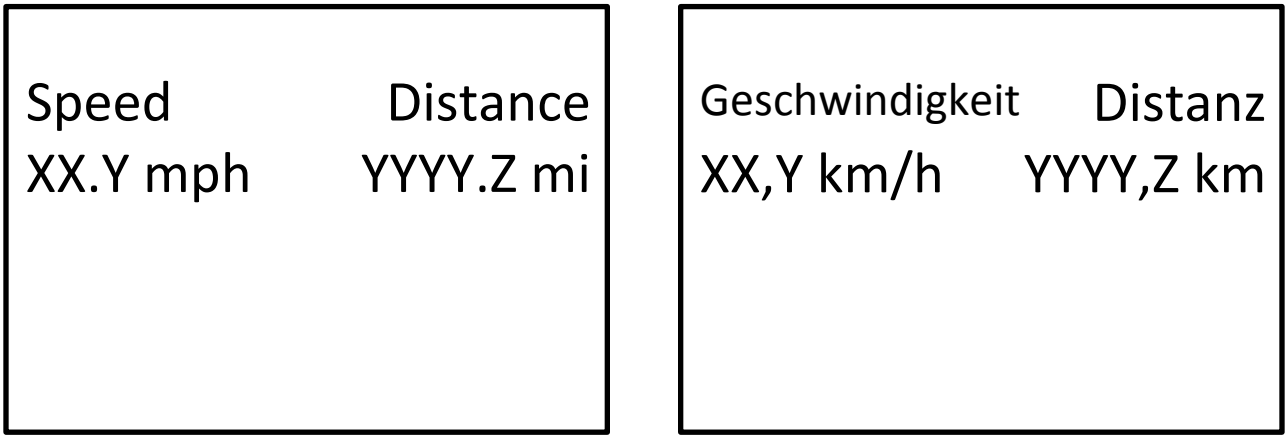

Figure 6: Display of a fictitious app for cycling in English (left) and in German (right) (Any similarity to actual cycling apps is purely coincidental.)

It therefore seems that everything looks great. Not so fast! The digits allotted for the speed, xx.y mph, allow for a maximal speed of 99.9 mph, which is usually plenty for a bicycle.[2] However, the same is not true for the speed in kilometers per hour. Professional cyclists routinely exceed 100 kph (about 62 mph) on downhills, and even I, as an amateur, have reached top speeds of over 100 kph on descents. In this case, the display cannot accommodate the data, which results in an error. The solution is to allot another digit for the speed display, which is quite feasible here, since there is enough space on the screen for another digit.

These examples, and especially the last example, nicely illustrate that QA testers do not only have to be linguistic experts, ideally they are also natives (or extremely familiar) with the target culture and very familiar with the app or the type of app they are testing. Of course, that does not mean that only golf pros should test golf apps, for example. However, clients usually expect familiarity with the terminology and the functionality of the apps that are being tested, and this means a certain amount of preparation for the testers.

I have now discussed the general QA testing process, the problem types that can arise, and a few specific examples. These bugs, when found during the QA testing process, have to be properly reported to the software engineers, such that they can be remedied.

3.4 Software problem reports

The project manager or localization engineer generally maintains a repository that tracks all problems and bugs that arose, along with the date of occurrence and their resolution. This

[2]The current world record on a bicycle exceeds 100 mph by far, and I have friends who have exceeded 100 mph with their HPV (human powered vehicle = a special, very aerodynamic construction that could be classified as a faired recumbent), but for the everyday cyclist who might be using this fictitious app, 99.9 mph should generally be sufficient.

repository serves as the interface between programmers and testers and is essentially a summary of all problem or bug reports submitted by the QA testers.

In general, QA testers are asked to submit separate reports for every single problem that they encounter, though this depends somewhat on the client. A typical bug report usually contains the following information (see also (Esselink, 1998)):

- Number or ID of the problem to allow for easy identification and tracking
- Product, version, platform/OS, platform version, hardware/firmware version (if applicable)
- Date of occurrence and (internal) build version
- Language and locale, usually identified by an alphanumerical code (see (Wikipedia, 2014) and references therein). In case of several language variants, these details are important – e.g., Swiss German differs from German German or Austrian German, just as American English differs from British English.
- Precise location in the file plus file name or precise location and name of the application where the problem occurred
- Precise description of the steps needed to reproduce the problem, avoiding extraneous information, but containing enough information such that a software engineer, who does not speak the target language and who does not have access to the test script, can reproduce the problem
- Precise description of the problem, taking into account that the software engineers most likely don't speak or read the target language, possibly including screenshots
- Solution to the problem or linguistic correction, preferably in paste-and-copyable form because the software engineers are most likely not able to enter special/non-ASCII characters on their keyboard. An annotated screenshot alone is therefore not sufficient and possibly confusing to the software engineers.

The precise format of the bug report and the submission procedure depends on the client's preferences. In general, it is always a good idea to keep in mind that the report needs to be detailed enough such that a person who is familiar with the product but who cannot decipher the specific target language can reproduce the problem. This is especially important if the source and target languages use different character sets.

It also helps the programmers if precise computing terminology is used in the bug report, e.g. if a checkbox is referred to as such instead of, say, a choice selector, or if the identifier of a variable is referred to as such instead of being called a placeholder, for example. While QA testers do not have to have a programming background, I have found that being at least able to use the correct terminology is very helpful when communicating with the people who need to implement the bug fixes. It also helps to know how to deal with remnant tags or other code fragments in the translated text. In short, some rudimentary programming knowledge is certainly an asset for software QA testers.

4 Summary

Above, I have given a brief overview over the internationalization and localization process for software programs or apps. I have then discussed the localization QA testing phase in more detail,

outlining and explaining the individual steps. There are several classes of problems or bugs that QA testers look for, and I have described these in detail along with a few concrete examples. Finally, I have listed the information that should be contained in a complete bug report such that the bug can be eliminated as quickly as possible. The following sections contain a brief glossary with computing terminology as well as a list of references and further reading.

5 Glossary

App: A software application, or app, is a computer program that carries out operations for a specific application. It cannot run by itself but requires an operating system to execute. An app can run on desktop computers (traditional software applications), on mobile devices (mobile apps), or over a network (in the "cloud").

Bug: A software bug is an error or flaw in a computer program that causes it to malfunction or produce an incorrect or unexpected result. The term "bug" was coined in 1946 by Admiral Grace Hopper in a report about a malfunction of a computer that was caused by a moth trapped in a relay.

Build: An unreleased version of a software program for testing, a "sub-version" if you will. Usually, apps go through many test builds before they are publicly released.

G11N: Numeronym for **globalization**.

Globalization: The entire process of **internationalization** and **localization**.

GUI: A **G**raphical **U**ser **I**nterface allows the user to interact with a program via graphical design elements such as buttons and icons instead of a command line.

i18n: Numeronym for **internationalization**.

Identifier: see **Variable**.

Internationalization: In computing, internationalization refers to the process of designing and preparing a software application such that it can be translated and localized without engineering (i.e. programming) changes.

L10n: Numeronym for **localization**. Here, the uppercase L is used to avoid a possible confusion between a lowercase l (L) and an uppercase I (i).

Localization: In computing, this is the process of adapting a product that has been previously translated into different languages to a specific country, group, or region. It is the second step in the process of translation and cultural adaptation of a product, the first step being internationalization.

Locale: In computing, a locale is a set of parameters that defines the user's language, country or region, and other specifics. This includes number, currency, date formats, paper sizes, etc. For example, an app in English still needs to be adapted to the various English-speaking locales with different currencies (US-Dollars, British Pounds, etc.), different spelling, and so forth.

Operating system (OS): An operating system is the software that enables other programs (**apps**) to run on a specific hardware and manages the interaction between apps, the hardware, peripherals such as printers etc., and the user.

Script: Here, a test plan that outlines the precise steps to navigate through an app and test all of its functionality.

Software string: A string is a sequence of characters. Software strings that are to be translated are not hard-coded but occur in the form of **variables**.

Variable: A variable is a symbolic name, the so-called **identifier**, associated with a value. The value can be changed. For example:

> My_name = "Carola"
> Here, "My_name" is the identifier of the variable, and "Carola" is its value.

6 References and Further Reading

Bratu, F. (2014). *Guidelines for Designing Documents Intended for Translation*. Retrieved from http://desktoppub.about.com/od/layout/a/translation_2.htm

Bodeux, E. & Whitty, T. (2013). *Two Sides of a Coin: Software Localization from Both the Translator's and Project Manager's Perspectives*. Presentation at ATA54, T-6.

Bodeux, E., McKay, C., & Whitty, T. (2014). *Software localization: insights from the translator and PM perspectives*. Speaking of Translation audio recording, retrieved from http://speakingoftranslation.com/listen/software-localization-insights-from-the-translator-and-pm-perspectives-new/

Esselink, B. (1998). *A Practical Guide to Software Localization*. Amsterdam/Philadelphia, John Benjamins Publishing Company.[3]

Gartner (2014). Press release: *Gartner Says Worldwide Software Market Grew 4.8 Percent in 2013*. Retrieved from http://www.gartner.com/newsroom/id/269631

Globalme/Golota H. (2013). *The Localization Testing Checklist*. Retrieved from http://www.globalme.net/blog/the-localization-testing-checklist

Niedermair, Ch. & Dietz, F. (2013). *More than a Coffee Break Amusement: Localizing Casual and Social Media Games*. Presentation at ATA54, T-5.

Seeburg, D. (2012). *Localization For iPhone and Other iOS Devices*. Presentation at ATA53, T-7.

[3] While the references to specific software in this book are somewhat outdated (publication date 1998), the overall strategy and detailed procedural information are still quite valid.

Wall Street Journal (March 2013). *Apps Rocket Toward $25 Billion in Sales*. Retrieved from http://online.wsj.com/news/articles/SB10001424127887323293704578334401534217878

Wikipedia (2014), *Language codes*. Retrieved from http://en.wikipedia.org/wiki/Language_code