VILNIUS UNIVERSITY
MATHEMATICS AND INFORMATICS FACULTY
SOFTWARE ENGINEERING STUDY PROGRAM

Programinės įrangos kultūrinių elementų neatitikimų lokalės normoms aptikimo automatizavimas

Automating detection of software cultural elements‘ inconsistencies with locale norms

Master’s degree thesis

Done by: Neda Zalieskaitė .....
(signature)

Supervisor: doc. Dr. Tatjana Jevsikova .....
(signature)

Reviewer: doc. Dr. Kristina Lapin .....
(signature)

Vilnius – 2023

Santrauka

Prasidėjus masiniam programinės įrangos eksportui į užsienio šalis ir didėjant kompiuterių bei interneto paklausai, lokalizavimas tapo itin svarbiu produktų pritaikymo konkrečiai vietai ar rinkai procesu. Lokalizavimas susijęs ne tik su kalbiniais klausimais, bet ir turiniu, kultūros normomis, bei jas pagrindžiančiomis technologijomis. Tačiau esami lokalizavimo modeliai ir įrankiai turi ribotą veiksmingumą sprendžiant pagrindinius iššūkius, todėl svarbiausi aspektai lieka neišspręsti ir lemia prastą vartotojo patirtį. Dažniausiai programinė įranga lokalizuojama rankiniu būdu, o tai užima daug laiko, brangiai kainuoja, lemia klaidų tikimybę ir priklauso nuo lokalizuotojų ir testuotojų kompetencijos. Be to, esami modeliai neatsižvelgia į lietuvių kalbos lokalizavimo kokybės užtikrinimo palaikymą.

Darbe analizuojamos mokslo publikacijos, standartai, kokybės užtikrinimo modeliai ir produktai, susiję su programinės įrangos lokalizavimo problemomis ir jų aptikimo galimybėmis lietuvių kalbos ištekliuose. Aprašomi ir sisteminami lokalizuotini programinei įrangai ir kultūrinei kalbai būdingi elementai. Atliekamas išsamus esamų lokalizavimo kokybės užtikrinimo priemonių ir modelių palyginimas.

Sukurtas naujas lokalizavimo kokybės užtikrinimo modelis su pasirinktais kultūriniais elementais ir lyginamų modelių aspektų deriniu. Suprojektuota modelį realizuojanti paslauga, skirta automatizuoti lokalizavimo klaidų paiešką programinės įrangos tekstiniuose lokalizuojamuose ištekliuose. Atliktas sukurtos paslaugos eksperimentinis vertinimas pritaikius ją daugiau kaip 100 projektų lokalizuotų tekstinių lokalizuojamųjų išteklių, atlikus ekspertų apklausą. Rezultatai parodė sukurto modelio reikalingumą ir pranašumus, palyginus su esamais lokalizavimo modeliais.

Raktažodžiai: lokalizavimas, lokalė, kokybės užtikrinimas, kokybės užtikrinimo modeliai, automatizavimas.

Summary

In response to massive software export to foreign countries and increasing computer and Internet usage, localization has become a crucial process of adapting products to a specific location or market. Localization is not just linguistic issues but also content, and cultural norms, and the underpinning technologies. However, existing localization models and tools have limited effectiveness in addressing the main challenges, leading to unresolved critical aspects and a poor user experience. Most software localization is done manually, which is time-consuming, expensive, error-prone, and depends on the localizer's and tester's competence. Moreover, support for Lithuanian language localization quality assurance is overlooked by the existing models.

The study analyses scientific publications, standards, QA models, and products related to software localization problems and their detection possibilities in Lithuanian language resources. Software and cultural language-specific elements to be localized are described and systemized. A comprehensive comparison of existing localization QA tools and models is carried out.

A new localization quality assurance model is created with selected cultural elements and a combination of aspects of compared models. A service implementing the model has been designed to automate the search for localization errors in localized software textual resources. An experimental evaluation of the developed service has been conducted by applying it to more than 100 projects' localized textual resources and through an expert survey. The results demonstrated the need and advantages of the developed model compared to existing localization models.

Keywords: localization, locale, quality assurance, QA models, automation.

TABLE OF CONTENTS

1. Introduction

During the recent decades there have been many examples of software expansion to the global market. As information technologies shorten geographical distances, businesses seek global market penetration [MH15]. It is no longer assumed that English is the only available software system language. The cost of software development can be considerably reduced by working internationally: the original product is developed once to be adaptable into many languages, i.e., localized to many languages. Moreover, while the product is being localized into many languages, the business gains more revenue from different geographical markets, new potential customer base, competitive advantage, improves the company's reputation, etc.

The need to localize software appeared when the software was massively exported to different countries. Today, while computer and Internet usage still grows, this need grows even more. Localization issues in software emerged 30 to 40 years ago when companies started creating customized software for clients beyond their initial target market [Hall02]. At first, specific software was developed in the language of the originators (usually, English). There were no standards on how to encode characters of non-Latin writing systems, thus software localization was ad-hoc. The situation changed with the appearance and further development of the Unicode standard. The Unicode gave the possibility for major writing systems to be handled adequately in software environments. Although Unicode has solved most of the major localization problems related to character use, other issues of local conventions still appear. Thus, software localization process and its quality improvement research are still relevant.

Localization is the process of adapting a product or content to a specific location or market. Localized software must properly produce and process documents, written in a particular language, use appropriate encoding, deal with local conventions [DGJ10]. While *internationalization* is the process of designing software and its data structures in a way that can be easily adapted to different languages and cultures [DGJ10]. Localization gave us the flexibility to adapt international software by translating text, adapting software elements according to specific language norms, processing dates, times, and other cultural elements to avoid operational problems. Software localization is one of the important tasks to ensure a successful software users' experience. It is important to provide a user with a software environment that does not contradict to his/her natural cultural environment [Sch02].

Every location (country, or part of it) has established its own cultural norms [DGJ10]. We learn how to comply with these norms when we talk, communicate, and write. Information technologies are an integral part of the modern world, so, there is supposed to be a possibility

to prepare and process documents that comply with cultural norms, and an interface between humans and computers in a particular cultural environment should be compliant with those cultural norms. That is important for software vendors to reach a global market, moreover, gain the highest business revenue.

Software adaptation for locale norms serves as a basis of the localization process. According to the international standard ISO/IEC 15897, the *locale* is "the definition of the subset of a user's information technology environment that depends on language, territory, or other cultural customs" [ISO99]. Not all cultural norms can be unambiguously and formally defined, but such norms are significant localizing software. Therefore, the locale definition is understood more broadly: "a group of aspects of specific location (country, or its part) and interface with the user" [DGJ10], often emphasizing the human-computer interaction complying with the cultural norms. For example, locale can be considered as the group of elements, specific for language, which is being used for the computer operating system, or software applications.

Localization Industries Standards Association (LISA) sees the objectives of localization not as just linguistic issues, but also content and cultural norms, and the underpinning technologies [Hall02]. Also, LISA sees the process of localization as "not a trivial task", the cost of localizing software in the early design phase is significantly less compared to market gain when the product is commercialized. "It is generally agreed that software should be designed so that subsequent localization is relatively cheap" [Hall02], meaning the early design phase. Often, localization issues are forgotten, and enterprises are not giving enough focus to a better user experience. This leads to worse product accessibility, usability, efficiency, and conflicts with profitability (business value) [OA13].

Monetary, paper, date and time, number, character, character case convention, character collation, and other cultural conventions are the most common locale issues. We still face issues when language-specific characters are broken if the software system does not recognize these characters [Bla21]. Many problems are caused by the semantic elements, e. g. usage of the correct grammatical forms, plural-singular forms, and composed strings [DJ09]. Furthermore, a widely distributed system needs to ensure 24/7 availability constraints. Since the system is widely distributed it means that every system component can work in a different region and time zone. Often this leads to global system time desynchronization because every component has its clock and it is running in a different granularity level of different locales [GFG20]. Therefore, simultaneous system operation sometimes is corrupted. Dates and times

are formatted incorrectly or not localized at all because systems cannot recognize various types of dates and times. Besides the issues mentioned before, many more problems occur during the localization process, which will be studied in this research work.

Software localization is an essential aspect of software development, but current localization models often fail to address the main challenges effectively. Despite the availability of various software localization quality assurance models and tools, many critical issues, such as alphabet and names, date and time formatting, measuring system, decimal fractions, thousands separation and others, remain unresolved. Consequently, most software localization is done manually by linguists or other specialists, which is time-consuming, expensive, error prone and depends on his/her competence. Manual localization testing can be a cumbersome task, so providing an automated technique for these processes has been the demand for a long time [Sin17]. Therefore, there is a need for more effective software localization solution that can help to solve these issues and improve the efficiency and quality of the localization process.

This study aims to address the main software localization issues encountered in the industry. The goal is to improve the user experience of Lithuanian-speaking users by combining several aspects of the existing quality assurance models. To this end, a new localization quality assurance solution has been developed that effectively deals with the selected localization issues. The literature review revealed that existing localization quality assurance (QA) products did not adequately address these issues, which often led to poor user experience and increased development costs. In contrast, a new a quality assurance model and it's implemented service in this study provides a comprehensive solution to the identified issues, resulting in improved software localization quality and increased efficiency of the development process. The service, implementing the model has been tested on several software applications, open-source projects, and compared to existing localization tools. It was found to improve the user experience by ensuring accurate and consistent localization across the Lithuanian language and culture. Overall, the research work identifies the gaps in software localization quality, demonstrates the importance of addressing software localization issues, the effectiveness of the developed solution in dealing with these challenges and the advantages over the existing localization models.

The study findings present that existing localization quality assurance products did not adequately address localization issues. Industry experts rely on manual localization testing, which is labour-intensive and error-prone, and the testing quality depends on the tester's competence. Additionally, the existing quality assurance models commonly overlook the Lithuanian language, resulting in poor user experience. The **aim** of this research work is to develop a model and its implementation to automate the detection and correction of software localization issues in textual localizable resources of the Lithuanian locale.

Objectives:

1. To analyze relevant research literature, standards, models and products on software localization issues and their detection possibilities.
2. To overview and systematize software cultural and language-specific elements and the scope of information and services that need to be localized.
3. To develop a model and its implementing service, automating the detection of the selected localization issues in the textual localizable software resources of the Lithuanian locale.
4. To experimentally apply and evaluate the developed service on the localized software resources of the Lithuanian locale and to assess service demand by the industry.

The development of a new service allowed us to achieve several significant objectives. We accomplished the goal to replace manual Lithuanian localization QA checking for essential localization aspects with an efficient automatic solution. Additionally, the service contributes to the reduction of high costs associated with existing models. Most significantly, the implementation of a new service enhances the overall user experience for Lithuanian-speaking users.

The rest of the research work is organized as follows. Section 2 describes the main concepts of this work. Section 3 presents the context, motivation, created model and its implementation of this master thesis. Section 4 displays the results obtained so far along with the developed model implementation. Finally, the results and conclusions are presented.

2. Software localization

2.1. Localization history literature overview

Localization research started when the need to expand to the global markets appeared 30 to 40 years ago [Hall02]. Its evolution has been marked by a move "from in-house localization to internationalization, along with marked changes in the nature of the tools used" [Ess02]. Thus, software development started to be internationalized ("is the process of generalizing a product so that it can process (maintain) different languages and cultural attitudes without redesigning it" [DGJ10]), which is related to the desire of software vendors to make more profit by expanding product market to the other foreign countries. *Microsoft* was one of the first companies to start its international operations in Tokyo, Japan in 1978, and one year later – expansion in Europe [Ess02].

From the 1980s computer hardware and software started to be used in a daily "normal" user life away from corporate or academic computing [Ess02] (see Figure 1 for further history timeline). As personalized computer usage was increasing exponentially, new software applications started to be built for the users to ease their life – turn their work more efficient, also software needed to reflect local standards, habits, and local language itself. Text editors needed to support input, process it, and output in some kind of format, user interface needed to be displayed in users' local language, etc. The very first drafts of some local language software versions were poorly designed. They contained encoding errors because initially there were no standards on encoding procedure, text input parsing, processing, and outputting were problematic, systems did not have proper tools to do it, or it was very limited. Therefore, the localization process was very ad-hoc [Hall02].

The demand for in-house translators and language engineers to support software vendor products started to increase in the 1980s after initial local language software versions were introduced. Decade later, software developers realized that localization should not be part of their day-to-day businesses and should be outsourced to external services. Software and hardware development teams started to outsource translation and localization tasks to focus on their main development competencies. So, outsourcing became a new trend for software localization. By the end of the 1990s, translation management systems (TMS) were introduced, to automate many parts of the human language translation process and maximize translator efficiency. It challenges many localization technologies because the translation of subsequent development updates of a software product can be simplified. Therefore, previously produced translations can be reused in many releases.

In the 1990s Ireland established itself as a leader in the localization industry. In the same decade, Lithuania also started to *lithuanianize* (case of localization, when the software is modified to suit the Lithuanian linguistic and cultural environment [DG06]) software by the Institute of Mathematics and Informatics. In 1995 first Lithuanian applications operating in the *MS-DOS* operating system was released. Moreover, different *Windows* and *MS-DOS* encodings used in e-mails were harmonized. Later, other applications, running on *Windows* and *Linux* OS started to be translated into the Lithuanian language as well.

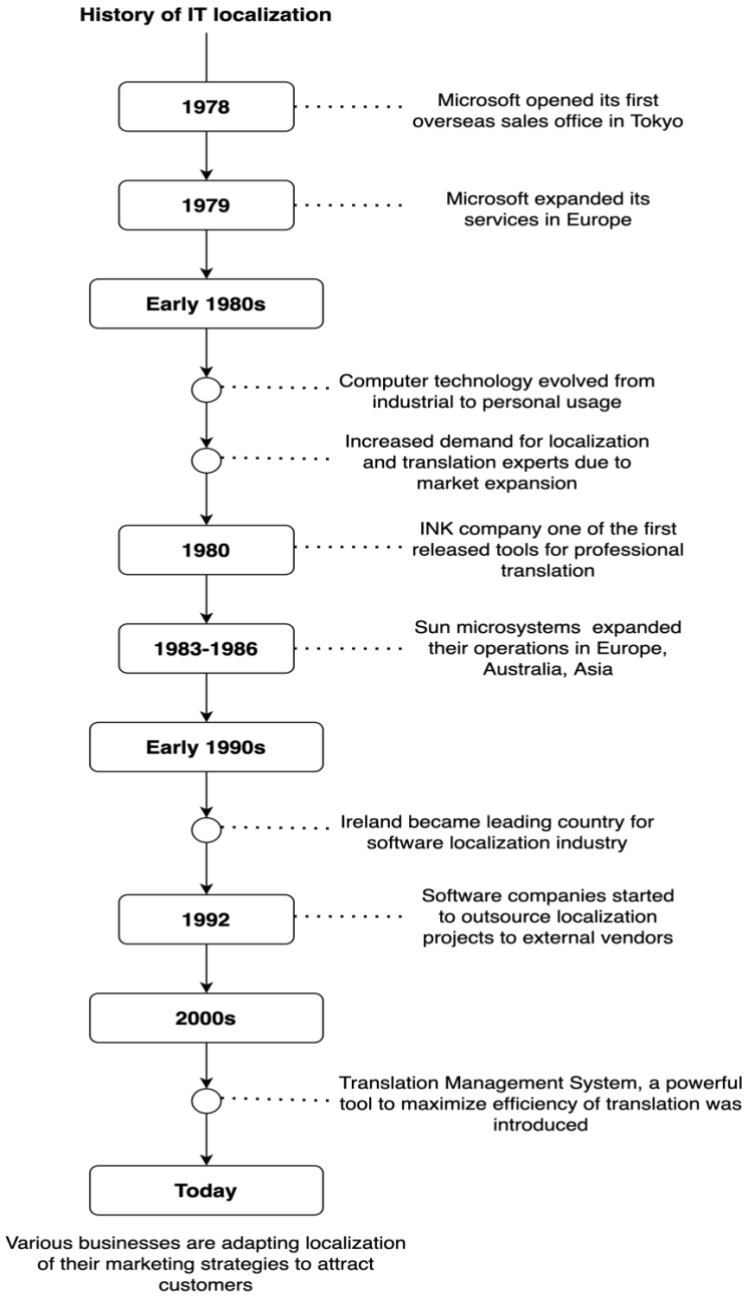


Figure 1. Localization history [Twa18]

Even though localization methods and practices were changing quite roughly compared with the 1990s, some of the old approaches are still being used today. Desktop and web applications are still using translated resources, which are compiled with the built environment. Development technologies like Java or .NET built their localization tools and libraries, so, most of the in-house translation at the end will be replaced. These technologies can process input during the runtime and output it in many locales. New standards to control the localization process were developed, such as ISO/IEC 15897, and Common locale data repository (CLDR) [Ess02]. Content management systems give a possibility to update translation resources without compiling and building a software application. Many new approaches to localization architecture were found, but the localization process has not reduced its complexity. We still can see many issues arising on language encoding, text element formatting for specific locales, character collations during the parsing, and others. Thus, the localization process still is one of the main concerns for the market expansion of software companies. And content localization simplification will be one of the main tasks for the software developers of global companies.

2.2. Software localization process

Adapting software to a particular language and culture environment is called *localization*. The localization process scope can be at different levels: starting with the customization of the program to process the characters of a particular language (partial localization) and to the completion of preparation of the software for the linguistic and cultural environment (complete localization) [DGJ10]. So that the person who is interacting with it feels as if the software was created in his cultural environment [Sch02].

The adaptation of the localized program starts with the preparatory localization tasks – the aim of which is to select necessary and the best programs for localization and prepare such a localization plan so that the quality of the localized program would be as high as possible [DGJ10]. The main steps that are taken before actual localization work are reviewed below (see Figure 2):

1. **Forecast of the number of future users**. Before localizing a software application, it is necessary to explore its number of future users. It is considered that it is worth localizing software applications in the Lithuanian language if there will be more than one thousand users.
2. **Acquaintance with a software license.** Localization is one way of modifying software. A software license is a legal document that must be strictly adhered

to. Proprietary programs can be modified and localized with the author's permission.

3. **Acquaintance with information about the author and his work**. From the author's works, it is possible to learn about his qualification, activity, and readiness to correct the internationalization errors observed during localization.
4. **Acquaintance with the software program.** The localizer must be well acquainted with the functions of the localized program and the terminology used in it. This can be achieved in the program directly or from the documentation.
5. **Analysis of localization possibility.** Types of software resources, types of localized resources and their analysis, the need to recompile the program, etc. should be analyzed before the actual localization work.
6. **Self-capability assessment.** Before starting localization, it is necessary to assess whether the strength, endurance, and resources will be enough to complete the localization work.
7. **Contact establishment with the author.** Useful even if the program resources can be freely modified.
8. **Internationalization analysis.** The complexity of localization depends on the level of internationalization of the program. It is easier to localize the program if the needs of future localization have been considered in its design phase: international standards are followed, and localized resources are separated.

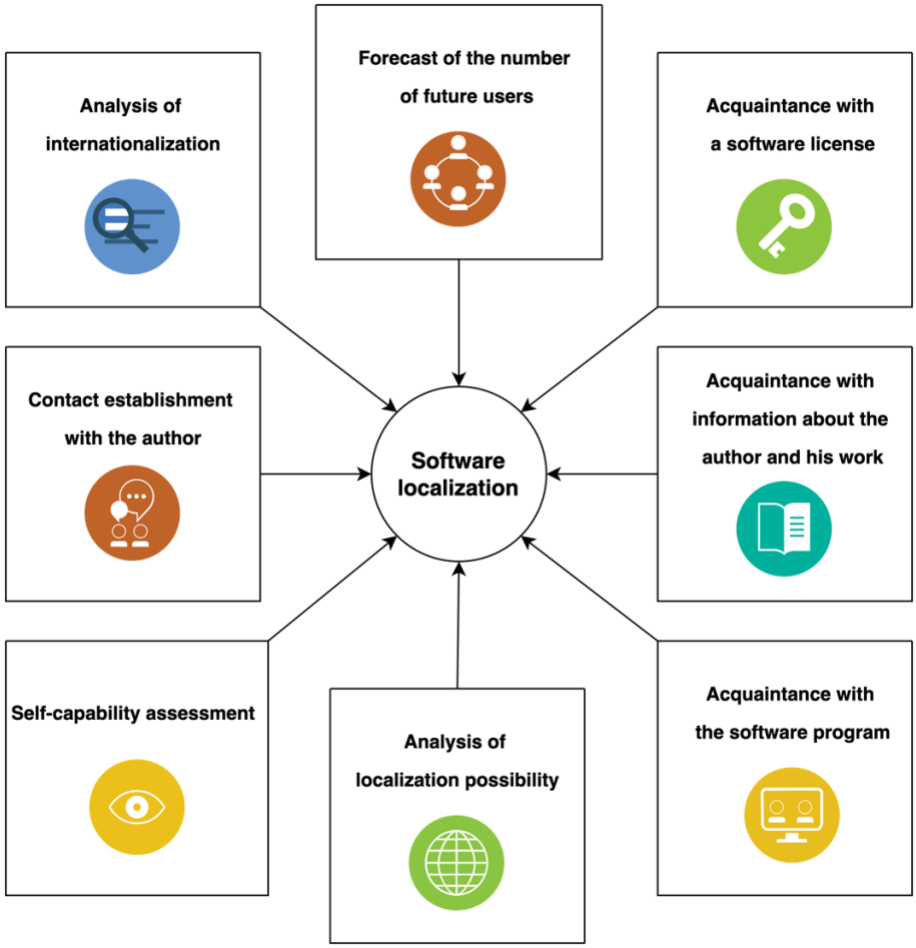


Figure 2. Preparatory work before localization [DGJ10]

Localization of software began gradually. The authors of the articles on the localization topic distinguish seven degrees of localization (see Figure 3). In this research work we will focus on three main localization degrees:

1. Local text processing.
2. Adaption to local norms.
3. Comprehensive localization.

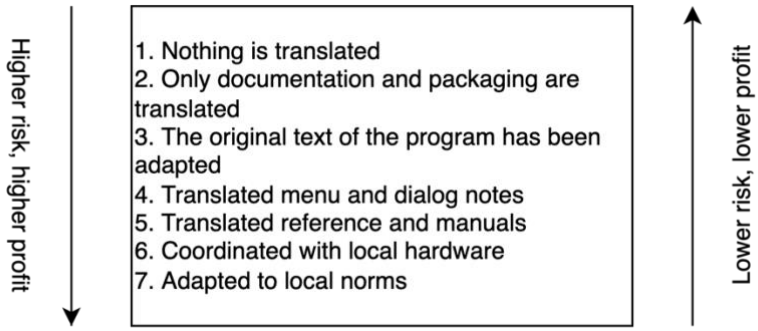


Figure 3. Seven degrees of localization [Car98]

The first degree is a possibility to process locale text documents, because software application, which is unable to process the characters of that language is worthless. When we enter any website on the Internet its content is parsed from resource files like HTML, RTF, or JSON. These files contain text, which is written in natural language (e.g., Greek, French, and Lithuanian), containing language-specific characters. When these resource files are being parsed to the source program and being rendered on the Web content is being checked by the parser. Parser evaluated the grammar of the text (syntax of text if it matches specific grammar rules, in our example could be Lithuanian language grammar rules), if no errors occurred then it is being tokenized into tokens, which are evaluated by the evaluator and then displayed on the Web. In Western Europe, this process was done as soon as computers were used to process text information, but in Eastern Europe, it was delayed of low economic development. The use of foreign characters to write texts in their language has long been tolerated (like ignoring language-specific characters in the Lithuanian language). Many computer users get used to that and no longer take the advantage of basic capabilities of modern technologies. So, society education is needed [DGJ10].

The second degree of localization is software adaptation to the various cultural and linguistic norms in a given language or territory (state, part of the state) [DGJ10]. Number and currency formats, date and time formats, the use of religious or other symbols, and symbolic meanings of specific symbols and colors, are the main cultural elements. Ignoring some of the cultural regulations can lead to serious mistakes (such as incorrect decimal fractions and separators of thousands). There are a lot of such requirements, which will be discussed later in a *locale concept* section.

And the last degree – is comprehensive software adaptation. Modern computers are intelligent devices that can perform service functions. A word processor replaces a pre-computer typewriter, an e-mail system replaces a mailer, etc. [DGJ10]. It is clear that the service must know and use the language of the client. A computer needs to “know” the language of a client when he changes a person – interacts with the client in his language. Thus, all texts displayed on the computer screen must be in client language.

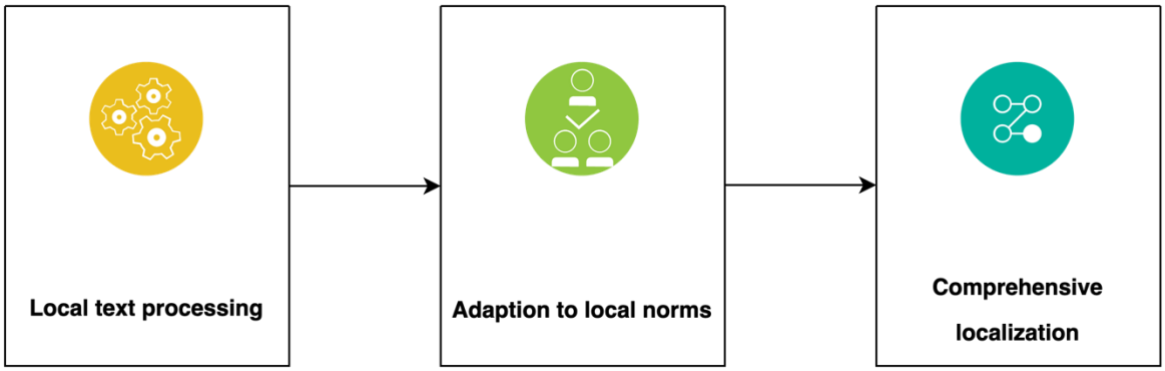


Figure 4. Three main localization degrees

Localizing web-based software applications, especially on the server-side, a third (complete) degree of localization can be pursued [DGJ10]. It is common for the localization of the website to localize only the content that is visible to the user – the view on the web page. Often, the website source text is forgotten, moreover, the universal addresses of the resources are displayed in the browser's address bar in a non-localized language. For example, Yijun et al. provide a method for localizing the content of XML documents and guidelines [Yij05]. Using the SGML standard (ISO 8879), which uses Unicode encoding of content and guidelines, creates guides for guideline translations. All these features allow the creation of appropriate add-ons for online software, such as a browser, and a web editor, that allow seeing the original text of a web page in the local language [DGJ10].

The information required to implement first, second, and partially third localization degrees of localization is the elements used in the software that are different for different languages and cultures [DGJ10]. These cultural elements will be analyzed further in the *software cultural elements* section.

2.3. Software locale

Locale is one of the key concepts in software applications. Locale is an explicit model and definition of a native-language environment [OC10]. International and Lithuanian cultural standard ISO/IEC 15897 locale is defined as "a subset of the user environment that depends on language or cultural norms" [DGJ10]. However, not all cultures can be unambiguously defined, thus, the locale is understood more broadly: the whole aspects of the dialogue with the user, depending on the location (whether it is the state or part of it) and the language-dependent aspects of the dialogue with the user.

It is impossible to formally define all these aspects. For example, it is impossible to define the cultural aspects of images, only recommendations on how to design images so that

they are acceptable in most cultures can be made. Thus, formally defining locales usually a limited generalized set of cultural aspects is used. The main parts covered by most locals are:

- **Language and country, their abbreviations.** Which language the user interacts with, country and abbreviation convention, and currency unit notation.
- **Character encoding, classification, and ordering.** Which characters are letters, numbers, what are language-specific characters, and charsets for a particular language?
- **Formatting**. How are numbers, currencies, dates, times, and addresses displayed?
- **Calendar, time zone.** Which calendar (Gregorian, Chinese, Islamic) is used, and which day is the start of the week?
- **Units for measurement.** Which units are used for measuring the length, mass, sound, and speed?

Locale is usually identified by the language, using a two-letter language code (ISO 639-1), and by territory, using a two-letter territory code (ISO 3166-1) [DJ09], for example, *en_US*, which identifies the English locale of United States. Nevertheless, there are locales, which are not dependent on territory (for instance, in Switzerland there are four official languages – German, French, Italian, and Romansh, thus, each of these languages has its language-territory locale combination) or language (locales of Australia, Great Britain, and the United States are different, even though the language is the same).

Different software vendors use different approaches to define cultural norms – some of them defined international standards (ISO/IEC 14882, 15897, IEEE Standard 1003), and some are considered de-facto standards, for example, Java locale models. Java localization capabilities will be analyzed later in the *Java localization capabilities* section.

The main elements identifying a locale are language and state (territory). For example, the United States, Great Britain, and Australia use the same language, however, they have different locales due to their different cultural attitudes and norms. The locales in the software are easiest to implement using the local services of the operating systems or another platform. However, this approach is not universal for two main reasons [DGJ10]:

1. A locale defines language alphabet, character encodings, alignments, date, and time formats, etc. Each language and state can have its own locale. One state may have different spoken languages and multiple locales (for example, Swiss-German, Swiss-Italian, and Swiss-French). Moreover, the same language can

be used in several states, and may also have separate locales (English in Great Britain and English in Australia).

2. Locale elements, which are mostly encountered in software, are accumulated, and formal ways to present them are sought. Of course, it is not possible to describe all the elements of a locale formally, such as software functions influenced by a culture-specific style of thinking, graphic elements, and use of colors, so only the main local elements are formalized.

One part of the software localization consists of the translation of dialog texts in a computer interface (notes and messages visible in the program, windows, and their elements - buttons, labels, menus, etc.) and electronic help, also other material related to the program into the local language. Another equally important part of software localization is an adaptation of other locale-related elements used in the program; the main elements are discussed in the *software cultural elements* section.

2.4. Software cultural elements

Cambridge dictionary defines *culture* "as the way of life, especially the general customs and beliefs, of a particular group of people at a particular time". The attitudes, beliefs, behavior, opinion, etc. of a particular group of people within society establishes a cultural environment for a location (country, or part of it). Vaske and Grantham (Vaske, Grantam, 1990) emphasize that culture is:

- Adaptable (can adapt to the physical and social environment).
- Integrated (characteristics, which make cultures generally compatible with each other).
- Constantly changing (due to adaption to certain cultural events or integrations with other cultures).

Many authors emphasize that there is a connection between culture and software for suitable use. In other words, software drives culture change. Thus, software engineer needs to understand cultural characteristics and cultural differences in purpose to create software that can be adapted to many cultures. The most important cultural elements which are used in the software will be analyzed further.

2.4.1. Alphabet and names

Names are used not only by humans but also by computers. Names written in the native language are easier to understand, memorize, manipulate, correct, and communicate (Dürst, 2003). In older software, there was a requirement to keep name length less than 26 letters,

which is very restrictive even for those languages that use the Latin alphabet. Some the languages have additional letters (ą, č, ę, ė, į, and others in Lithuanian language), and some letters are rarely used in names (q, w, and x).

Login names. The majority of software programs verify user identity. One part of such verification is user login name authenticity logging in to the system. Unfortunately, many systems only allow using Latin letters, numbers, and underscore (_) characters to create login names. Therefore, it is clear that the user should have an opportunity to choose a name made up of the letters of his / her native alphabet, not just ASCII characters [DGJ10]. Although there are no technical barriers to the use of multilingual characters in login names, software designers tend to skip this feature in their products.

Personal names. Many applications have a user profile area where personal data is recorded and stored when creating an account in any system. All letters of a person's real name and surname should be accepted, the system should not change the spelling of the user name and surname to register. "Observations show that the number of correctly written names (using letters with diacritics) of Skype users varies from 15% to 96% depending on the language" [DJ09] (German, Czech, Danish, Estonian, Icelandic, Latvian, Polish, and Lithuanian languages were used for the survey). So, this high level of "illiteracy" can be explained by a habit inherited from other programs that still have restrictions on login name signs [DGJ10].

Passwords. Password is another part of the verification process for a user's login into the system. Password is usually composed of letters, numbers, and special characters for a strong password requirement. Many systems limit the set of letters to only ASCII characters, which is not natural for those locales whose languages use another alphabet. Moreover, it reduces password security requirements.

File names. Operating systems allow files to be called by real, natural names. Letters, numbers, and some other characters in the text, such as a dot (.), and a dash (-), are used. Only characters that have a special purpose in this context, such as slashes, which are used to indicate the path to the file, and commas, etc., are not allowed [DGJ10]. A person can choose file names in their language so that the name could describe the contents of the file. The same goes for directory names.

Domain names. User, who works with Internet software often operates with domain names. For a long time, only 26 letters of the English alphabet could be used to compile domain names [DGJ10]. The latest versions of the most popular browsers have tools for working with internationalized domain names. However, many companies still avoid using this feature. One possible reason could be the risk of fraudulent use of homographic signs from different

alphabets when the method is implemented internationally. Another - is that not all older programs work correctly with internationalized domain names used as hyperlinks.

2.4.2. Personal name formatting

A person's first and last names are represented differently in different cultures (e.g. "First Last", "Last First", "Last, First") [DJ09]. In a software applications personal names are used to greet logged-in users or when the session starts. Thus, personal name formatting should be included in the localized resources as a separate parameter or by using different parameter names for the first and last name so that they can be easily swapped.

2.4.3. Date and time formats

A date can be represented in a long (for example, 1st of January 2022) or a short (01/01/2022) form. A longer form is more complex because the year and month representing numbers should be expressed along with text lines, which form can differ. Shorter date form also may differ between the cultures. For example, in Lithuania, the short date form consists of three parts. The first on the left is the 4 digits for the year, the second is the two digits for the month, and the third is the two digits, meaning day [DGJ10] and separated with dashes (for instance, 2022-01-01). Time can be represented in 12- or 24-hour format with the different hour and minute separators (e.g., dot, colon, space, dash). For example, in Great Britain, the 12-hour format is used (e.g., 10:00 AM, 10:00 PM) and many other European locales e.g., Lithuania use the 24-hour format (e.g., 10:00, or 22:00).

2.4.4. Measuring system

Indicates, which system (or systems) of measurement is used in the locale [DJ09]. For example, in Europe metric system units, in Great Britain and the United States – inch, miles, and pounds are used. In Lithuania metric measuring system is used.

2.4.5. Decimal fractions and thousands separators

International Standard ISO 31-1 defines two types of decimal separators: a comma and a period [DGJ10]. United States, Great Britain, Australia, and other countries that use the English language, the decimal fraction is a dot (.). In Lithuania and other European and South American countries decimal fraction is a comma (,). The wrong decimal fraction in a given locale can be confused with the thousands separator and cause incorrect number values.

In the U.S., to separate thousands commas are used, in many European countries, a dot or space is used. An incorrect thousands separator can be confused with the decimal fraction and cause incorrect numeric values in the system.

2.4.6. Currency formatting

Each country usually has a unique currency and its symbol. This includes national rules and symbols that are used to represent monetary amounts and currency symbols [DJ09]. They appear in online shops, web forms, etc. In Lithuania, and most European Union countries Euro (€) is used as a national currency.

2.4.7. Postal address and telephone number format

Usually appears in internet applications, e.g., in user profile data, forms, and form filler options [DJ09]. A postal address is composed of address line one (where street, house number, and the flat number are filled), and two, if necessary, state, and postal code (normally it is a five-digit number, but in some countries, it can be a four or seven-digit number).

The telephone number format differs per country. As a rule, it consists of country code, area code, and the rest. If the form for dialing a telephone number consists of several cells, there should be a possibility to change the number and/or order of the cells during localization.

2.5. Software localization-related standards

Locales define cultural elements and tools and how to formally specify them. In this section, we are going to analyze the most important documents and models used to specify locale data. Most of these documents are international standards.

2.5.1. POSIX family standards

The **Portable Operating System Interface** (POSIX) is a family of standards specified by IEEE for maintaining compatibility among operating systems [BC21a]. Thus, any software that is compliant with POSIX standards should be compatible with another operating system, which conforms to the POSIX standards.

Early in software development start programmers used to write software for an operating system in different environments and specifications. Thus, shifting to another operating system would require a lot of cost and effort resources. So, to overcome this problem POSIX was introduced. *POSIX is the standardization of the original UNIX, which came back in 1988 to resolve issues not only between different UNIX variants but also Non-UNIX operating systems as well* [BC21a].

POSIX development started in the 1980s, first standard version was introduced in 1988. Two parts of the standard were associated with the software locale: POSIX.1, which became the accepted standard of ISO/IEC 9945-1, and POSIX.2 (ISO/IEC 9945-2), which was internationally accepted as IEEE Std 1003.2-1992.

POSIX locale model defines its standards base on the C programming language. The model formally specifies how to define each category element (see Table 1). POSIX locale model is used in *Unix* operating systems, the latest releases of *macOS* are completely POSIX compliant, while *Microsoft Windows* OS does not conform to the standard at all because its whole design is completely different from UNIX-like operating systems [BC21a].

Table 1. POSIX locale categories [BC21a]

Category	Description
LC_TYPE	Used for character classification e.g., lowercase, and uppercase letters, decimal, and hexadecimal digits.
LC_COLLATE	Defines the order of characters
LC_MONETARY	Used for monetary formatting, e.g., currency symbols, and decimal fractions used for money calculations.
LC_NUMERIC	Used for formatting numbers, e.g., thousand separators, digit grouping requirements.
LC_TIME	Used for date and time formatting
LC_MESSAGES	Used for program messages such as information messages and logs.

POSIX locale model advantages [DGJ10]:

1. It is an international standard that was the first to formally define the key elements of a locale.
2. Portability: independence from the operating system or platform.

POSIX locale model disadvantages [DGJ10]:

1. Categories described in Table 1 cannot be expanded.
2. Possible incompatibility with encodings.
3. Too few categories. They do not cover basic cultural norms related to language and territory.

2.5.2. ISO/IEC 15897 standard

ISO/IEC 15897 (procedures for the registration of cultural elements) is a standard, which sets out the procedures for registering cultural elements, both as narrative text and more formally, using the techniques of ISO/IEC 9945-2 POSIX (POSIX.2 version) Shell and Utilities [ISO99]. In 2001, this standard was adopted in Lithuania as well (LST ISO/IEC 15897). Standard is compatible with POSIX.2 defined categories; however, it allows for

registering more locale elements. Registration results are freely available: software developers are free to use them. Four types of cultural specifications can be registered under this standard [DGJ10]:

1. Descriptive cultural specification.
2. POSIX locale.
3. Encoding of POSIX characters.
4. A set of POSIX characters.

The descriptive cultural specification defines cultural norms in narrative English and may include equivalent definitions in other languages. Types 2, 3, and 4 are described using the POSIX specifications for cultural elements, defined in ISO/IEC 9945-2 standard [DGJ10].

The first six chapters of the descriptive culture specification overlap with POSIX locale categories, other most important chapters for a more detailed description of the cultural aspects are listed in Table 2.

Table 2. Elements of the descriptive culture specification [ISO99].

Specification chapter	Description
National or cultural Information Technology terminology	Terminology for a language or culture can be listed, e.g., translations of ISO terminology for Information Technologies.
National or cultural profiles of standards	Profiles of standards can be listed, e.g., profiles of the POSIX standards.
Character set considerations	Describes how characters are used in the culture, for instance: which characters are necessary to write in a particular language, for further precision or used in newspapers and books.
Sorting and searching rules	Used for further description of how to split a record into sorting fields, which fields are ignored when comparing or sorting.
Transformation of characters	Describes transliterations and transformations of characters, e.g., transliteration requirements between Latin and Greek languages.
Use of special characters	The use of special characters such as quotation, abbreviation, and punctuation marks are described. Moreover, what should be avoided is specified in this chapter, e.g., number signs.
Character inputting	The chapter describes keyboard inputting rules and input alternatives.
Personal names rules	What is considered a family name, how different titles are used, and fathers' and mothers' family name inheritance are described.

Hyphenation	Hyphenation rules can be described in this chapter.
Spelling	Specification of spelling rules and lists.
Numbering, ordinals, and measuring system	A measuring system can be described.
Monetary amounts	Old currencies can be described.
Date and time	Extension of POSIX time chapter. Time zone names and day length saving rules can be described.
Coding of national entities	Coding of different entities, such as postal codes, and police districts can be described.
Telephone numbers	Formatting of telephone numbers nationally and internationally can be described.
Mail addresses	Formatting of post addresses is described, how sender and receiver addresses are written.
Electronic mail addresses	Cultural conversions of electronic addresses can be described.
Identification of persons and organizations	Culture may have numbering formats for person and organization identifications, for example, personal identification numbers, and tax numbers for companies.
Payment account numbers	Conventions for bank account numbers for culture can be described.
Keyboard layout	Conversion of keyboard layout can be described.
Paper formats	Conventions of paper size and the use of window envelopes can be described.

Even though the standard is compatible with the POSIX locale model, which allows the operating system or platform independency, ISO/IEC 15897 locale data is not actively recorded. Thus, software developers cannot take advantage of them.

2.5.3. CLDR standard

Unicode **Common Locale Data Repository** (CLDR) provides key building blocks for software to support the world's languages, with the largest and most extensive standard repository of locale data available [UCLDR22]. CLDR data is widely used across companies for software internationalization and localization, adapting software of multiple languages support. CLDR includes:

1. **Locale-specific patterns for formatting and parsing:** dates, times, timezones, numbers and currency values, their symbols, and measurement units.
2. **Translations of names:** languages, scripts, countries, regions, currencies, eras, months, weekdays, day periods, time zones, cities, etc.

3. **Language & script information:** characters used, plural cases, gender of lists, capitalization, rules for sorting and searching, writing direction, etc.
4. **Country information:** language usage, currency information, calendar preference.
5. **Validity:** Definitions, aliases, and validity information for Unicode locales, and languages.

The project was started in 2003 continuing the *OpenI18N* group project of creating an XML local data store. The first CLDR 1.0 version was released in 2004, and the latest – CLDR 41.0 and is continued by the Unicode consortium.

The goal for the common locale data is to make it as consistent as possible with existing locale data, and acceptable to users in that locale [UCLDR22] (see Table 3 for more data types). Moreover, accumulate as much locale data as possible (currently, CLDR covers more than 400 languages).

Table 3. LDML data types [UO22]

LDML data category	Description
Locale display names	1. *languages:* provides localized names for all languages in *Language-List.* 2. *scripts:* provides localized names for all scripts in *Script-List.* 3. *territories:* provides localized names for all territories in *Territory-List.* 4. *variants*, *keys*, *types:* provides localized names for any in use in *Target-Territories*; for example, a translation for a phonebook in a Lithuanian locale.
dates	1. calendars: localized names 2. month names, day names, era names, and quarter names. 3. week: minDays, firstDay, weekendStart, weekendEnd. 4. am, pm, eraNames, eraAbbr. 5. dateFormat, timeFormat: full, long, medium, short. 6. intervalFormatFallback.
numbers	symbols, decimalFormats, scientificFormats, percentFormats, currencyFormats for each number system in *Number-System-List.*
currencies	displayNames and symbols for all currencies in *Currency-List*, for all plural forms.

transforms	Transliteration between Latin and each other script in *Target-Scripts.*
collation	The sorting order of strings of characters, also language-sensitive searching and grouping.

At the moment it is the greatest locale data repository (in 2022 with the 41.0 release there are 414 basic locale specifications). In the 40.0 CLDR version grammatical features, such as grammatical gender and case were introduced to form grammatical phrases. For example, it could sound as bad as "on top of 3 hours" instead of "in 3 hours" [UCLDR21]. The main goal of CLDR was to introduce grammatical gender and case so that advanced message formatting could be handled. Moreover, this feature was introduced in the Lithuanian locale within the second release phase.

CLDR also provides tools to export data to POSIX compatible format and Java resource sets. Some well-known companies such as *Apple* (*macOS, iOS, tvOS*), *Google* (*Web Search, Chrome, Android, Google Maps*), *IBM* (*DB2, Lotus, Websphere*), and *Microsoft* (*Windows, Office, Visual Studio*) use CLDR for their products.

2.6. Java localization capabilities

Java is a high-level, class-based, and fully object-oriented programming language used for many kinds of software development. TIOBE index (software quality indicator company) results in May 2022 show that Java is 3rd most popular programming language used for server-side applications and it was the 1st for a long time.

Native Java runtime is rich with its capabilities to internationalize and localize software. For example, Java 8 *Locale* class, is used to quickly differentiate between different locales and format content appropriately, which is vital for the internationalization process – "preparing an application to support various linguistic, regional, cultural or political-specific data" [BC21b]. The *locale* class consists of more than 150 available locales and most importantly that all required locale-specific formatting can be accessed without recompilation [BC21b]. Applications can handle multiple locales at the same time, and new language support is straightforward.

Locales usually are indicated by language, country, and variant abbreviation, which is separated with an underscore (for instance, *da* – Danish, *fr_CH* – French, Switzerland, or *en_US_UNIX* – English, United States, Unix platform). There are two additional fields, which can be set up – script and extensions. General rules to set these fields [BC21b]:

- **Language** can be an *ISO 639 alpha-2 or alpha-3* code or registered language subtag.
- **Region** (Country) is *ISO 3166 alpha-2* country code or *UN numeric-3* area code.
- **Variant** is a case-sensitive value or set of values specifying a variation of a *Locale*.
- **Script** must be a valid *ISO 15924 alpha-4* code.
- **Extensions** is a map that consists of single-character keys and *String* values.

The locale class model consists of the following local elements: number formats, date and time formats, currency symbols, calendar items, time zones, sorting and other operations with text, and the layout of graphical components, local resources separation. The classes of the listed locale categories are given in Table 4.

Table 4. Java Locale class categories

Category	Description
Numbers and currencies	Class: *NumberFormat*. A particular locale is passed as an argument to initialize this class. Numbers are formatted into locale-specific decimal and thousands separators. Currency formatting involves currency symbol append and rounding decimal part into two digits.
Date and time	Class: DateTimeFormatter. Class initialization consists of including one date and time pattern, and locale. Short and long date and time formatting.
Calendar	Classes: *GregorianCalendar, Calendar*. Date formatting of particular calendar and locale.
Text ordering	Classes: *Collator, RuleBasedCollator, BreakIterator*. Consists of language-specific rules of how letters and strings are ordered. BreakIterator class finds the location of boundaries in text.
The layout of graphic elements	Class: ComponentOrientation. Contains all methods required for creating user interfaces, painting images, and their language-sensitive orientation.
Separation of locale resources	Class: *ResourceBundle*. The vital part of internationalization is to provide localized messages and descriptions that can be externalized to separate files.

The Java locale model has more methods compared with the POSIX model. Unicode from the beginning was introduced in the Java locale model. Although Java has some localization limitations (rules of personal name writing), those can be achieved through parameterized statements.

2.7. Overview of existing localization (internationalization) quality assessment products

Oracle Java documentation identifies that internationalized application has the following characteristics [OC22]:

- Executable applications with their localized data can run worldwide.
- Graphical user interface elements which have textual content are not hardcoded in the program. Instead, they are stored outside the source code (e.g., the content management system) and retrieved dynamically.
- Support for new languages does not require recompilation of the program.
- Culturally dependent data, such as dates, currencies, and numbers, occur in formats that conform to the end user's cultural environment.

Based on the listed characteristics the quality of software localization (internationalization) can measured. In the global market, there are several existing products, for measuring localizability (ability of a computer software program that can be converted to any specific or general local language according to its local culture with local look and feel [BS13]) quality measure. In recent years, several localization QA models and tools have been developed to automate the process of identifying and correcting localization issues. In this section, we will overview four such products: *LocalyzerQA, Localise, ErrorSpy,* and *QA Distiller*.

Lingoport company created the *LocalyzerQA* product to provide linguistic quality reviews of a running application after the translation process. *LocalyzerQA* is a cloud-based localization QA tool that utilizes a combination of rule-based and machine-learning techniques to identify and correct localization errors, from inconsistent translations and missing translations to mistranslations and incorrect grammar - spelling, grammar, punctuation, and formatting [Lin23]. Its focus on localization testing enables it to fix identified issues, thereby cutting down the linguistic QA process. *LocalyzerQA* helps companies to test their translations for accuracy and consistency. Product allows users to upload their translations and test them against a set of rules[Lin22]. The first version of *LocalyzerQA* was released in March 2021, thus it is still in its early development stage and provides a small set of localization QA features.

While a tool can find translation issues and replace them with the help of native language linguist from the *Lingoport* company team, it does not address critical localization errors and does not support the Lithuanian language.

Lokalise is a cloud-based translation management system developed by *Localise* company that provides a machine learning-based localization QA to automate, organize, control, review localization work, and provide quality assurance checks on the project level. The tool uses a combination of natural language processing (NLP) and machine learning techniques to identify and correct localization errors in multilingual content. It also has a built-in visual editor that allows users to edit translations in context [Loc22]. *Localise* focuses on localization process workflow: organization and control of localization work, and it monitors identified issues throughout the development process. The tool has been trained on a large dataset of multilingual content and can detect some errors in spelling, grammar, and formatting [Loc23b]. However, this relates to the linguistic (translation) errors and does not provide for other cultural elements formatting support analyzed in the *software cultural elements* section.

ErrorSpy, a localization quality assurance product, is designed for error monitoring and uses a combination of rule-based and machine-learning techniques to identify and correct some of translation errors. *ErrorSpy* scans localized resources and generates a report of potential translation issues, such as grammatical, consistency, typography, and number formatting errors [DOG22]. It also uses machine learning to identify more complex errors, such as mistranslations and incorrect grammar [DOG23]. Identified issues are reviewed manually by linguists, making tool only partially automated. Despite being developed over twenty years, *ErrorSpy* only supports a small set of languages, and Lithuanian is not among them. Additionally, it focus on terminology consistency and linguistic issues and lacks support for other basic cultural elements such as date and time formatting, personal name, postal and telephone number formatting, different measurement systems, and currency support.

QA Distiller is a standalone tool that aims to find translation mistakes in bilingual files [Yam23]. The tool's functionality includes the automatic detection of common errors (omissions, inconsistencies, formatting, terminology), which can provide a direct link to the errors by highlighting them. The tool's features are fully customizable, and the Lithuanian language is supported, allowing testers to create profiles, use regular expressions when searching for mistakes, or customize the errors they want to detect. After project evaluation, testers can generate a report that highlights the issues. Nonetheless, this tool supports the Lithuanian language, but it still lacks support for other cultural elements formatting analyzed in the software cultural elements section.

LocalyzerQA, *Lokalise*, *ErrorSpy* and *QA Distiller* products will be analyzed further to understand their pros and cons more in-depth (Table 5).

Table 5. Comparison of existing localization quality assurance products

	LocalyzerQA	**Lokalise**	**ErrorSpy**	**QA Distller**
Number of supported file formats	15	28	20	5
Source file preprocessing	N/A	Advanced	Advanced	N/A
Number of languages supported	N/A	More than 400	N/A	More than 90
Lithuanian language support	Not available	Available	Not available	Available
Translation memory	Available	Available	Available	Available
API integrations	Available, the exact number is not known	More than 40	Not available	Not available
Mobile integrations	Not available	Available	Not available	Not available
Mobile platform support	Available	Available	Not available	Not available
Activity logs and statistics	Advanced	Advanced	Not available	Advanced
Content management system support	Not available	Available	Not available	Not available
Installation	Required, cloud-based (additional hosting pricing)	Required, cloud-based (additional hosting pricing)	Required only to the local workspace	Required only to the local workspace (*Windows*

				operating system)
Pricing	Starting from 29$ per month	Starting from 120$ per month	Starting from 249€, a one-time purchase	Free

The four localization quality assurance tools have different areas of focus. *Localise* manages the entire localization process, *ErrorSpy* emphasizes error monitoring, and *LocalyzerQA* together with *QA Ditiller* focuses on localization testing. *ErrorSpy* adheres to the ISO 17100 standard on requirements for translation services and SAE J2425 translation quality metric model to capture error types and quantities of translation errors [Woy01] as important evidence of the QA process [DOG23]. *Localise* employs ISO 9000 quality management standards [Loc23b] to establish, implement, maintain, and continuously improve their quality management system [ISO15]. The quality assurance model of *LocalyzerQA* and *QA Distiller* is not explicitly stated in the documentation, but it is assumed the use a combination of localization industry standard association (LISA) QA and multidimensional quality metrics (MQM) models due to its rule-based (metric-based) approach to assure quality. The comparison of these models will be presented in the later sections.

Four localization QA tools discussed in this section provide capabilities for identifying and correcting selected localization and translation errors. Each tool can detect some errors in spelling, grammar, punctuation, formatting, and terminology.

Despite effectiveness, existing localization QA tools and their models mostly rely on manual checking, which is time-consuming, expensive, and often results in errors due to human factor. Although some aspects of software localization, such as functional testing, language spelling and grammar checking, can be automated, practically all aspects of resource translation and localization testing still require manual processes. This approach depends heavily on the competence of the localizer/tester, which can lead to inconsistencies in the translation and a poor user experience. Moreover, the tools discussed have certain limitations, such as limited language support, high costs, partial automation, and critical localization aspect checking for Lithuanian language. Furthermore, *LocalyzerQA* and *QA Distiller* LISA QA and MQM models provide locale violation, punctuation, and other, mostly translation-related metrics, the critical localization aspects remain unresolved. Although these features can be

offered natively by Java programming language, they are often overlooked in existing localization QA tools.

Due to the reasons discussed above, a new localization QA service using Java, *Gettext PO* technologies and its model is proposed in this study for small projects with limited budgets and open source projects. This service aims to fix the limitations of existing QA tools and their models by providing comprehensive and automated verification of critical aspects of software localization for Lithuanian locale. By leveraging the built-in features of Java programming language, this service can provide an efficient and cost-effective solution for ensuring consistent and accurate localization for Lithuanian language and culture. Its development represents a vital contribution to the software localization field and Lithuanian language localization QA with potential to improve the efficiency and accuracy of the localization process, particularly for smaller / open source projects. Moreover, service also aims to improve the user experience of Lithuanian-speaking users and provide a competitive advantage to competitive advantage to companies seeking to expand their reach across different cultures and languages, including Lithuanian.

3. Localization quality assurance

3.1. Overview of objectives for a new localization quality assurance service

After conducting an analysis in the previous section, we analyzed software cultural elements, which we will be incorporated into the definition of a new QA model. The newly developed model will combine certain aspects of ISO 9000, ISO 17100, LISA QA, and MQM models. The implementation of a new QA model is realized in the development of a new service. In this section, we will discuss the objectives of a new localization quality assurance service developed to address some of the challenges of existing localization QA models.

Objective 1: automate localization QA checking.

Manual checking of localization quality is a labour intensive and time-consuming process to complete [OA13] that involves human testers checking the localized product or service for issues. However, manual checking of localization QA is not reliable as it is subjective and depends on the competence of the localizer/tester [BOK20]. Automated localization testing can significantly reduce time and effort required for localization QA. The reasoning underpinning this is driven by the reduction in workload that can be brought about through the automation of such techniques [OA13] and overall improve localized content and user experience. To address this, the new localization quality assurance model aims to replace manual checking with automated processes that can detect and report errors quickly and accurately.

Objective 2: provide support for the Lithuanian language localization QA.

Lithuanian is a Baltic language spoken by more than 3 million people. It is a complex language with a unique grammar and a complex set of linguistic rules. However, many localization QA models, including products review before, do not support Lithuanian language testing. To solve this problem, new localization QA model aims to provide comprehensive localization testing and ensure the quality of localized content for Lithuanian-speaking users.

Objective 3: reduce high costs of existing localization QA models.

Localization QA is a critical aspect of globalization, but it can also be costly (up to 20% of the total localization budget [PGD14]). Many businesses struggle to keep up with the high costs of existing localization QA models. Therefore, another objective of the localization QA model is to help reduce the high costs by providing affordable, automated solution.

Objective 4: provide QA for critical localization aspects.

Localization involves more than just translating the content from one language to another. It also involves adapting the content to meet the cultural and linguistic requirements of the target market [DGJ10]. This includes aspects like measurement system, date and time, and currency formatting, alphabet and names, decimal fraction and thousand separations. To overcome this, new model aims to provide comprehensive QA for these critical localization aspects.

Objective 5: improve the user experience of Lithuanian-speaking users.

Quality is a measurement based on end user expectations [OA13]. The final and ultimate goal of new model is to improve the user experience of Lithuanian-speaking users. This includes providing comprehensive localization QA to meet the linguistic and cultural requirements of the target Lithuanian market.

In conclusion, the new localization quality assurance service has a several objectives aimed at improving the localization QA process. It aims to address the challenges of existing QA models by providing automates and cost-effective approach.

3.2. Existing localization QA models

In recent years, several QA models have been developed to assure the quality and improve the process of identifying and correcting quality issues. This section provides an overview of five such models mentioned previously in *overview of localization (internationalization) quality assessment products* section: ISO 17100 standard, ISO 9000 standard family, SAE J2450 metric, LISA QA and MQM models.

ISO 17100 standard, ISO 9000 standard family, SAE J2450 metric, LISA QA, and MQM models are all related to localization QA. ISO 17100 standard and SAE J2450 metric mainly focus on translation quality assurance, which is only a part of localization. ISO 9000 standard family is oriented on the overall quality of various products and services. And the last two, LISA QA and MQM, models are more directly related to localization.

ISO 17100 standard specifies the requirements for the entire translation process, including the translator's qualifications and experience, translation project management, quality control, and the use of technology. The standard is designed to ensure that translation services meet a high level of quality, and that the translation process is carried out consistently and efficiently [ISO15b]. ISO 17100 assures clients that the translation service they receive is of a high standard and that the translation process has been carried out according to recognized best practices. Therefore, it is particularly relevant for organizations that require translation services for important documents (legislation, medical reports).

The ISO 9000 family of standards consists of several different standards. ISO 9001 is the most well-known and widely used standard in the family, and it provides a framework for organizations to establish and maintain a quality management system that meets customer and regulatory requirements. ISO 9004, on the other hand, guides how to improve the overall performance of an organization [ISO15a]. By implementing the ISO 9000 standards, organizations can improve their quality management processes, increase customer satisfaction, enhance their overall performance, and demonstrate a commitment to providing quality products and services.

SAE J2450 metric enables evaluators to capture error types and qualities of translation errors. To evaluate translation quality, each error identified by the evaluator is marked in two ways. First, is it categorized into seven predefined categories. Then, the evaluator determines if it is a serious or a minor error based on their perception of the error's severity. Finally, each error is multiplied by its weight and counted. This score is then divided by the total number of words measured, resulting in a translation quality score (TQS) [Woy01].

LISA developed a quality assurance model to evaluate all components of a localized product in terms of its functionality, documentation and language issues [BS13]. LISA QA model is assumed to assess translation quality by identifying errors and checking if the target meets the requirements. Quality model also suggests rating errors based on their severity levels (minor, major, and critical) and provides data recording and reporting capabilities.

MQM is a comprehensive framework for evaluating the quality of translated source texts in the context of translation. MQM is based on an analysis of over 20 existing translation quality assessment metrics and provides an extensive set of issue types that can be checked in both human and machine translation processes [LBU13]. The list of Issue types comprises over 100 issue types and covers most translation quality assessment metrics. Issues are arranged in a hierarchical structure with higher-level categories and their respective subtypes. The five MQM issue categories are fluency, accuracy, verify, design, and internationalization. Using this framework, users can evaluate translation quality across a wide range of criteria in a systematic and customizable manner.

Localization QA standards and models discussed in this section are effective in identifying and addressing localization and translation issues. They describe standardized guidelines on how to evaluate quality. Unfortunately, most of these guidelines remain unimplemented in today's tools, nevertheless, standards provide procedures on how to realize efficient quality assurance. ISO standards and SAE J2450 metric model rely on manual quality assurance. While LISA QA and MQM models provide a comprehensive set of quality metrics

that are not adapted to the Lithuanian language, therefore, critical Lithuanian locale aspects remain unresolved and rely on manual checking. Our ultimate goal is to improve the user experience of Lithuanian-speaking users by combining several aspects of the ISO 9000 family standard, SAE J2450 metric, LISA QA, and MQM models to offer a promising solution to overcome Lithuanian language localization issues.

3.3. Localization quality assurance process

Testing is the most crucial task in the software development life cycle to assure the quality of globalized software since its common mistakes are easily noticeable on the user interface for the final users [CM22]. Internationalization and localization testing ensures that software can be used in different geographic areas this could mean considering language, local conventions etc. [Pat06]. The localization testing process includes the content of the software and user interface. To ensure quality and reinforce that the software meets the conventions and requirements of a specific culture, it is necessary to perform internationalization and localization testing on the software [CM22]. QA testers try to "break" the localized software to locate bugs, which can be eliminated so that end-users in a target region can enjoy the error-free product [Tom20].

However, this type of testing is usually put aside or performed only at late development stages or not carried at all [CM22]. Typically, software localization quality assurance process is performed before releasing to an expensive production environment to avoid undetected errors. It can be evaluated as:

User satisfaction = compliant product + good quality + delivery within budget and schedule [BS13].

Localization quality assurance takes part in every task [Ber21]:

- Properly defining the project requirements.
- Preparing a complete localization kit.
- Accurately translating and correctly editing the text, which will be appended in product dialogue boxes.
- Performing accurate dialog box resizing and professional application publishing.

Figure 5. Localization QA process [Ber21]

Typical QA process looks as follows, as illustrated in Figure 5:

1. Familiarize yourself with the product in the source language. Carefully going through all the screens, buttons, menu items, and error messages. Writing test script (manual or automated script that contains the instructions for implementing a test case [IBM16]) to test each screen, button, menu item, etc.
2. Setting up the operating system for testing. Usually requires changing locale, language, and other settings for the specific target region.
3. Going through every screen and button, following the test script. Document all bugs that were found, also, providing specifics, like: operating system, software version, steps how to reproduce the bug.
4. Send the report to the developers. May also include storing the report in a company repository, or bug reporting system.
5. Waiting for a new build of a software product with eliminated bugs found in previous steps.
6. Going back to step one and repeating.

Localization QA involves technical and linguistic expertise. Localization engineers focus on graphical user interface (UI), compatibility (cultural norms) and functionality of software product, while linguists are evaluating the linguistic and visual parts of the software. More specifically, different types of localization QA exist [Mie22]:

- **Linguistic testing**. Translations are verified: grammar, spelling, style. Includes reviewing every dialog box, menu and as many texts as possible. Considers:
    - Spelling, grammar, and punctuation.

    - Translation being appropriate to the context. Sometimes translations do not have a full context of a word on a button, menu item, sometimes hard to guess if it is a noun or verb.
    - Consistency in translation. Does terminology is consistent thought the software product?
    - Correct display of strings, variables, and national characters (e g., letters with diacritics).
    - Truncations.
    - Cultural aspects. Does the translation respect all local sensitivities and will the end users understand all cultural references?
- **Visual / cosmetic testing**. Focuses on all visual aspects of a localized application (menus, dialog, boxes, messages, reports). Considers:
    - Text layout, alignment (left-to-right, right-to-left).
    - Line breaks.
    - Number, date / time, currency formats, addresses, zip-codes, phone numbers.
    - The number of options, menus, or commands in the localized version (must be the same as in the original one).
    - Application aesthetics (sizing and alignment of buttons, control boxes).
    - Dialog boxes (must be able to be resized without truncations).
    - Extended characters display.
    - Menu items, help balloons, dialog boxes, status bar messages (must fit on the screen in all resolutions).
    - Regional settings in dialog boxes (decimal separators, date, and time formats, etc. must be correct).
- **Localization testing**. Performed on each language version of a localized software product. Focuses on linguistic and cosmetic verification of localized application. Considers:
    - Link, button, menu functionality. Do all of them function as intended in the localized software product?
    - Input/output validation. Do the input fields allow proper entries in the units/formats for the target region? Do the output fields are formatted according to the target region cultural norms?

- Number, date/time, currency formats, addresses, postal codes, phone numbers.
- Character display, sorting, filtering, or input. Is all (non-ASCII) character are properly handled upon input, displayed, saved, sorted?
- Keyboards and shortcuts. Can all keyboard shortcuts and control functions be accessed with international keyboard layouts?

In this work, we focus on localization testing quality assurance, automated issue detection, and correction for the Lithuanian locale. Alphabet and names, date and time formatting, measuring system, decimal fractions and thousands separator cultural elements discussed in *software cultural elements* section will be used for issue detection. Based on a literature review locale and localization issues' analysis, a new model and its implementated service is developed to check key-value property parsing of textual resources (localized textual resources and source code) and detect and fix issues to comply with Lithuanian locales cultural norms.

3.4. High-level overview of a new service developed

Defect localizer is a quality assurance service for detecting localization inadequacies, which operates via an application programming interface (API). The developed model combines certain aspects of models used in *Localise*, *LocalyzerQA*, *ErrorSpy* and *QA Distiller* (Figure 6) to address several critical localization challenges previously discussed in *overview of existing localization QA models* section. Aspects included from other models are marked in bold in Figure 6. The service aims to improve Lithuanian language support, reduce high manual localization defect detection costs, increase automation level, and enhance the critical localization aspect checking objectives.

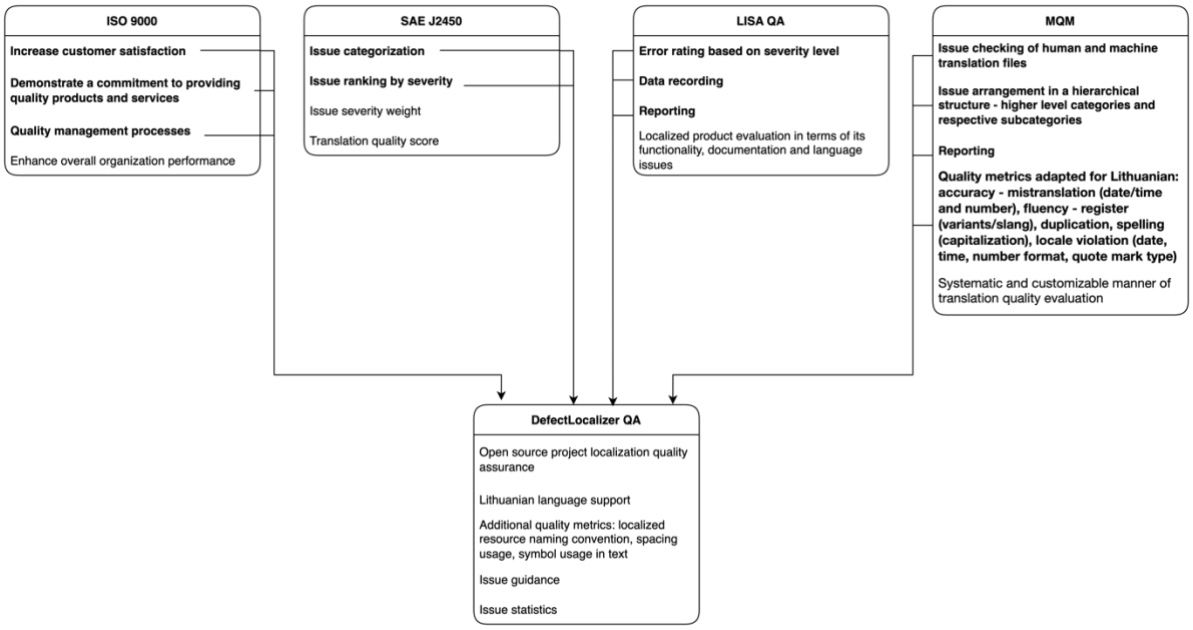


Figure 6. Existing QA model summary and new model overview

Service application programming interface (API) based approach provides numerous advantages in detecting localization inadequacies. Utilizing APIs, service can smoothly integrate with existing software systems and enable automatic defect detection. Combining the strengths of the models discussed previously, the new service will offer users a comprehensive and effective solution. API approach allows for rapid data processing and analysis, enabling users to quickly identify and correct localization inadequacies.

The *Defect localizer* identification and correction process automates the stage of the localization QA process discussed previously "following the test script, documenting all bugs" (Figure 5). Testers are not required to go through every screen and button of the application and document all bugs manually. The service automatically scans localized resources to detect and report all bugs.

Service offers two distinct use cases, both of which are visualized in Figure 7. The first use case involves directly uploading of localized resources to the *Defect localizer* service through client request, as demonstrated in Figure 8. The second use case involves filtering resources by downloading remote repository source files via client request and providing HTTPS or SSH link of the remote open-source repository in the request query parameter, as illustrated in Figure 9.

To accomplish its goal of detecting localization inadequacies, the *Defect Localizer* service first parses localized textual resources to identify key-value pairs - localized property and its value. These key-values are then subjected to rigorous analysis to check for any localization errors that may be present. This analysis involves examining each key-value for

potential issues reviewed in the *localization property grammar checks* section that may affect the user experience. By carefully analyzing each key value, the *Defect Localizer* service can identify and flag any localization inadequacies, providing developers with the necessary information to make appropriate corrections and ensure the quality of their localized products.

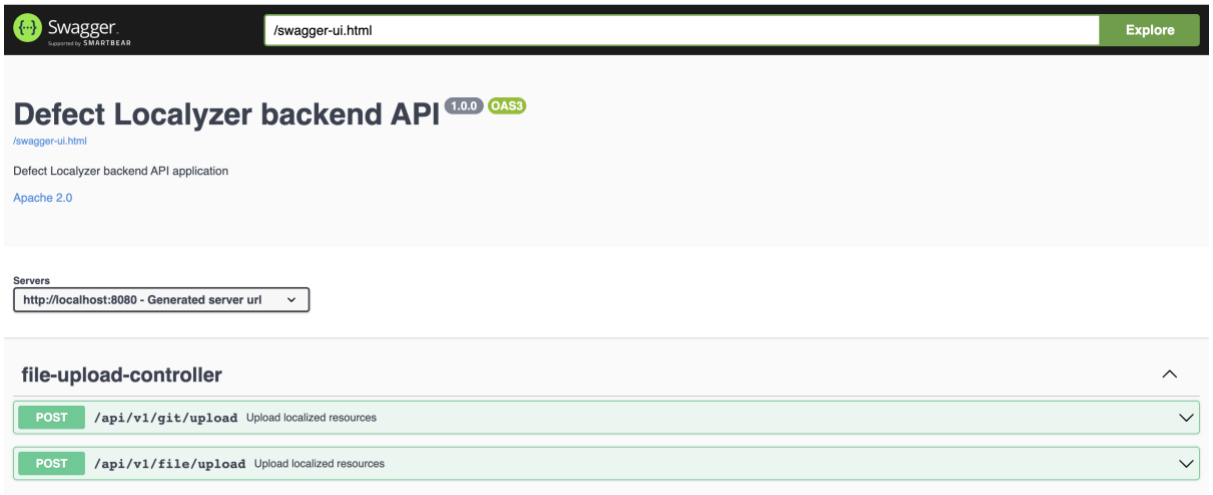


Figure 7. OpenAPI Defect localizer service API documentation

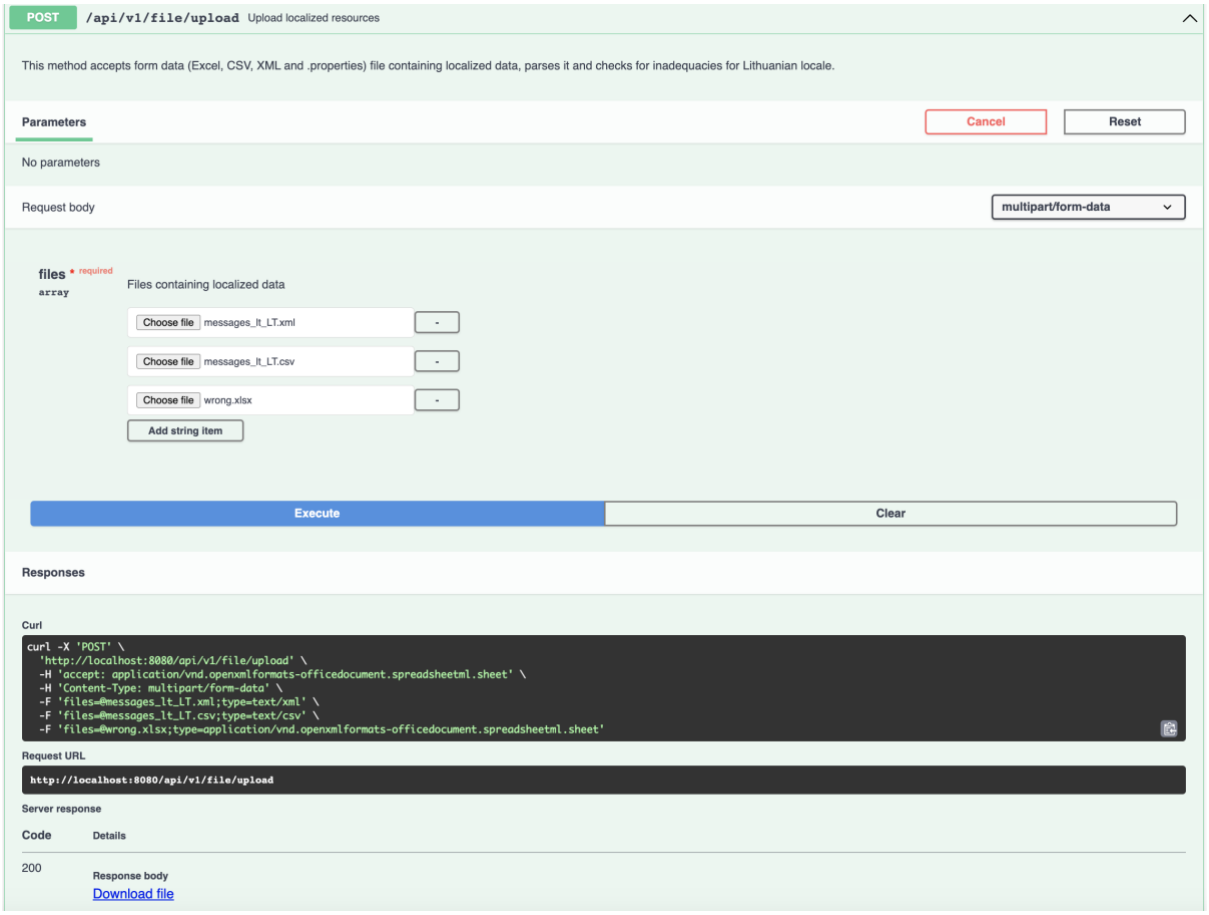


Figure 8. Direct localized resources upload inadequacy use case example

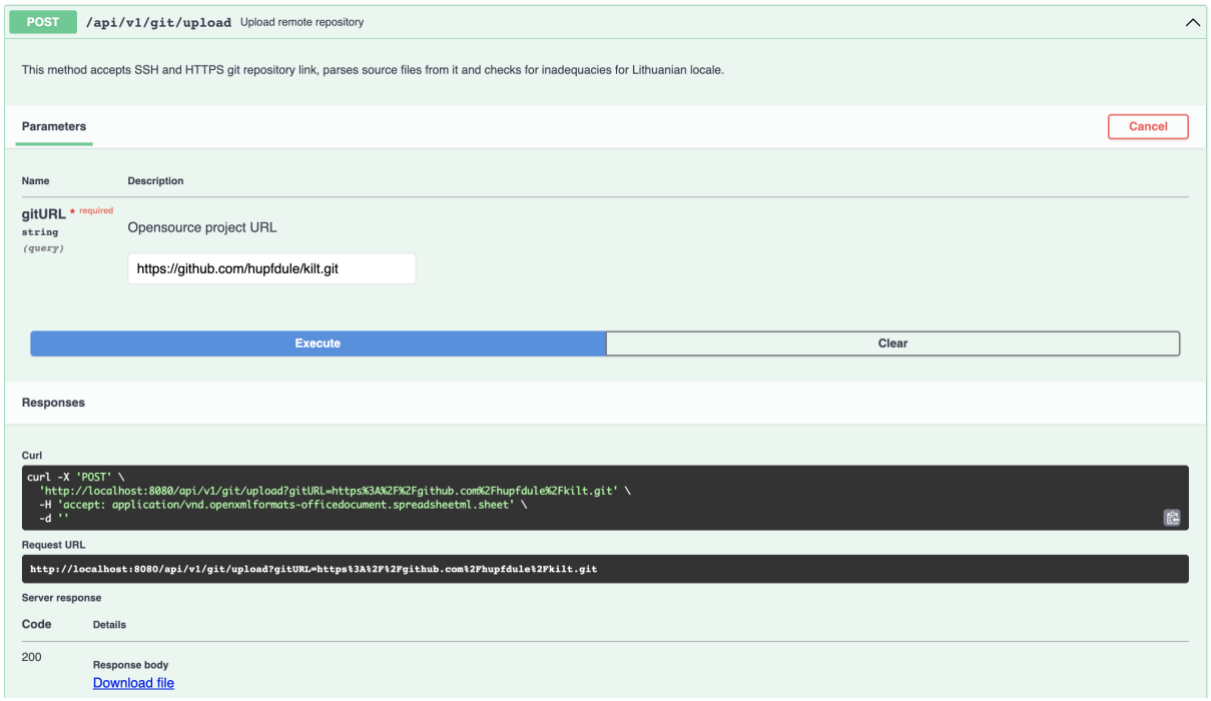


Figure 9. Open-source repository resource inadequacy check use case example

Defect localizer is composed of five services, each with a specific responsibility (see Figure 10):

- **Locale detection service**. Responsible for identifying the locale of the localized resource. The service verifies the presence of Lithuanian locales as the *Defect localizer* is targeted for Lithuanian locale support. The textual presence of the resource locale is verified through two steps. First, the service for the existence of the locale code in file naming (by base locale name, language code, or Java locale name) convention. Second, it checks for the existence of the locale code in the file columns in Excel and CSV file formats. If no locale code is found in both cases, new violations are created, and the resource locale is counted as unidentified.
- **Git service**. Responsible for verifying if the remote repository in the client request query parameter exists and can be cloned into the service file system. Service can establish a connection to the remote repository using HTTPS or SSH protocols. Once files are downloaded and filtered, the connection is closed, and the repository is removed from the service file system.
- **File parsing service**. File parsing service filters out unsupported file formats (any, except xls, xlsx, xml, csv, properties, and PO) and parses contents of the resources into *Property* Java objects. The *Property* Java objects contain property key, value, filename, and property line number. The object values are

be used to generate an issue report in an Excel file for better tracking. Later, Java objects are analyzed by the inadequacy check service.

- **Inadequacy check service**. The service applies all grammar rules discussed in the *localization property grammar checks* section to the *Property* Java objects. Service analyses property value for inadequacies and generates a report of found issues and suggestions on how to fix them.
- **Excel report generation service**. Service collects found violations from inadequacy check service and locale counter from locale detection service. Data is used to generate an analysis report with found issues, suggestions on how to fix them, and the location where the issue was found. Additionally, service builds a locale count report and column chart with issue statistics.

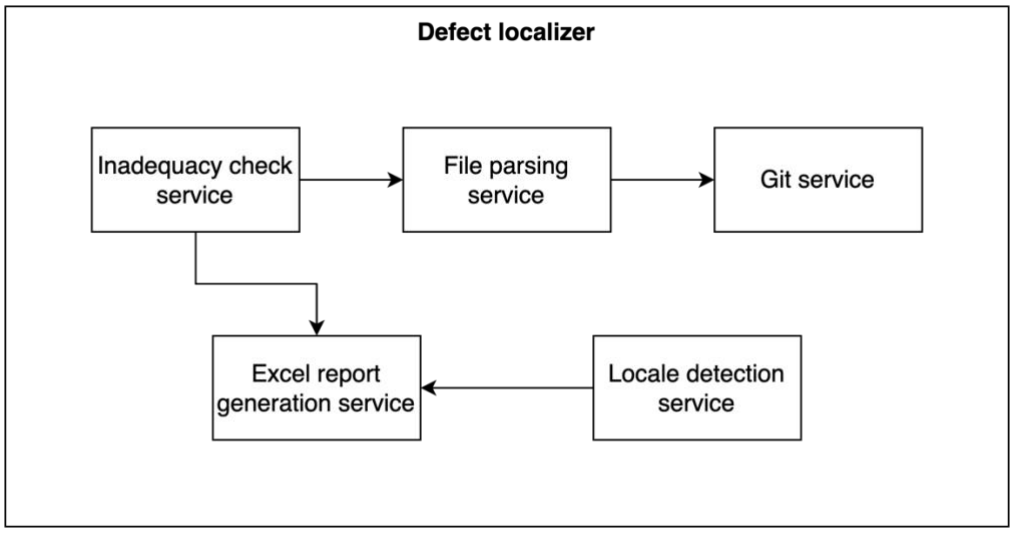


Figure 10. Defect localizer service component diagram

Supported *Gettext* PO (portable object) file formats are mainly used for the translation of software and have a unique structure compared with others. They are human-readable plain text files that contain translations of strings for different languages. PO files consist of a header section and message sections, where each string contains a source text and its corresponding translation. These files are typically created using a text editor or specialized translation software. When translating a PO file, translators typically use a message ID (msgid) to identify the source text and the message string (msgstr) to enter the translations for the target language [GNU23]. When parsing PO files, the message ID serves as a *Property* key, while the message string is the associated value. In contrast to other file formats, line numbers are not included in PO files as translations are typically carried out in text editors without specific reference to the location of the text being edited within a file.

In addition to *Gettext* PO files, other file formats (xls, xlsx, xml, properties and csv) are commonly used for the translation purposes due to their simple structure and integration. Translators typically use text editors to specify the translatable property, which serves as a *Property* key and translation values (*Property* value). Although these file formats may contain numerous translatable properties, line numbers are included in *Property* Java objects to facilitate issue tracking. This is important as translators may encounter difficulties when working with these formats due to the large number of translatable properties.

The high-level localization inadequacy QA process flow for two distinct use cases is visualized in Figure 11. The first use case involves direct uploading of resources via client request, while the second use case involves filtering resources by downloading remote repository source files through a client request. Regardless of the use case, the localization inadequacy QA process flow involves parsing localized textual resources and analyzing key values for potential errors. Any identified localization issues are then flagged and presented to the user for appropriate corrective action in the Excel report.

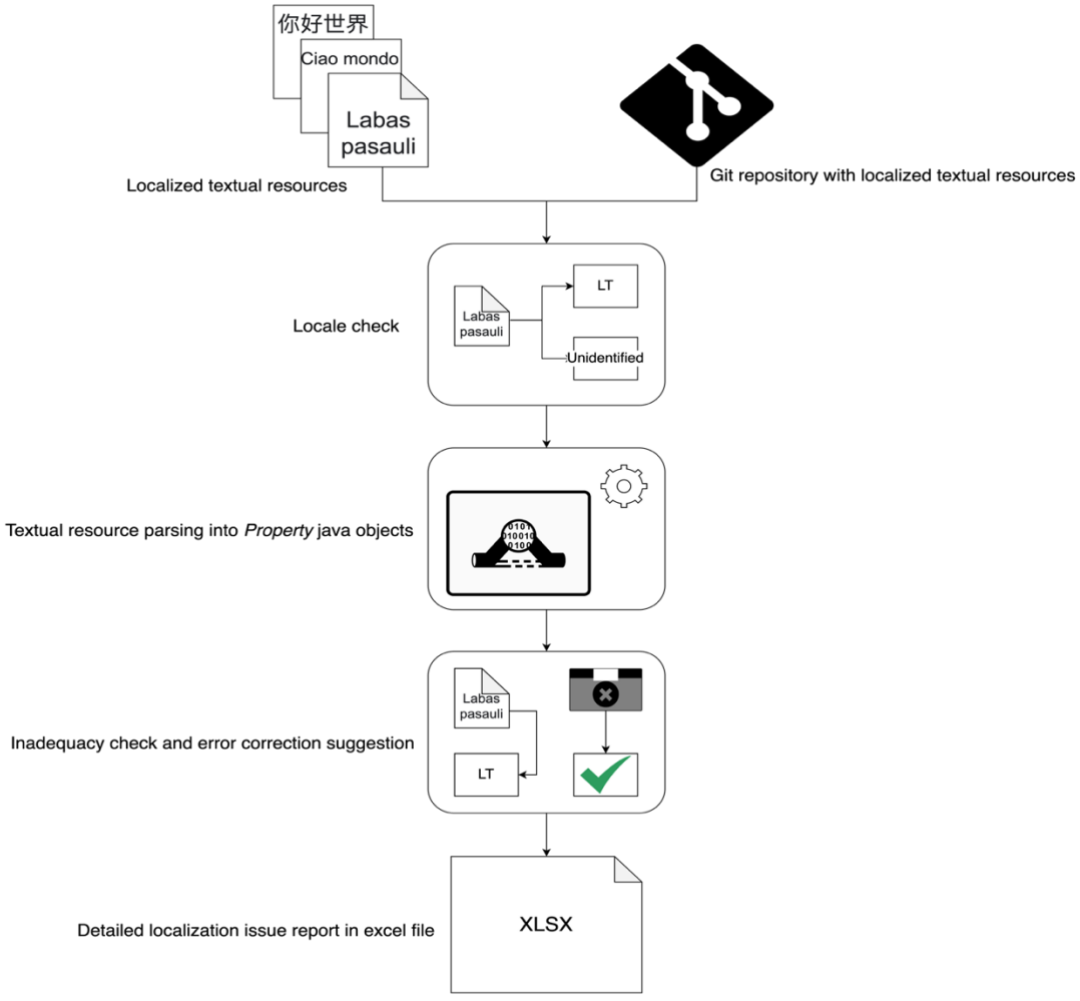


Figure 11. High-level process flow of the Defect localizer service

3.4.1. Direct resource upload inadequacy check use case

One of the two distinct use cases involve the direct uploading of supported file formats into the system and get a detailed spreadsheet report with detected Lithuanian locale inadequacies and suggestions on how these mistakes could be solved. The green rectangles identify the unique *Defect localizer* process over existing localization QA products. The detailed process for this use case is clearly illustrated in Figure 12.

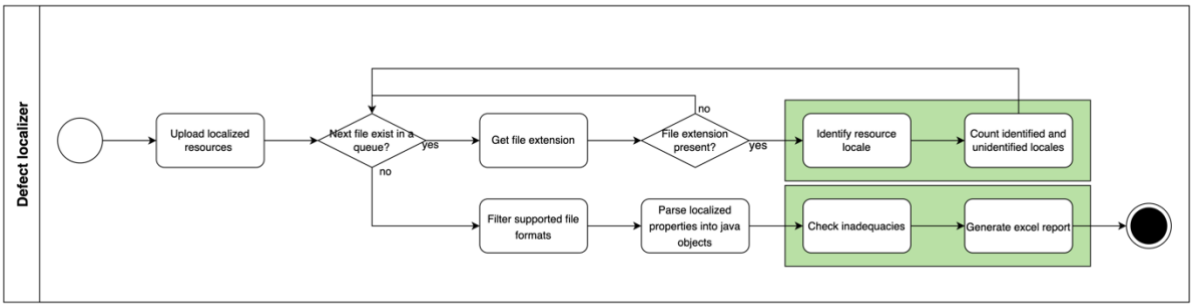


Figure 12. Localized resources upload inadequacy check flowchart

The process flows as follows:

1. User uploads localized textual resources into the *Defect localizer* system. File-by-file resources are iterated and passed in the locale detection service, where Lithuanian or unidentified locales are counted.
2. Locale detection service checks for file extension. If the extension is present, then locale presence is checked, otherwise, the service iterates the next file. Then locale presence in file naming and file column in Excel and CSV file formats is checked.
3. Locale counter counts Lithuanian and unidentified locales.
4. Files are parsed in the parsing service. The parsing service then additionally filters out unsupported localized textual resource formats.
5. Resource content is parsed into *Property* java objects.
6. *Property* java objects are analyzed in inadequacy service where all checks are applied to ensure smooth localization QA.
7. Excel report generation service generates the report with a list of found issues and other reports.

3.4.2. Open-source repository resource inadequacy check use case

The service also offers a second use case for users to download localized resources from an open source remote repository by providing an HTTPS or SSH link. Once downloaded, service checks the resources for Lithuanian locale inadequacies, generating a comprehensive

report that included detailed suggestions for resolving any identified issues. The green rectangles present the processes which are unique in the service compared with existing localization QA products. The detailed process is illustrated in Figure 13.

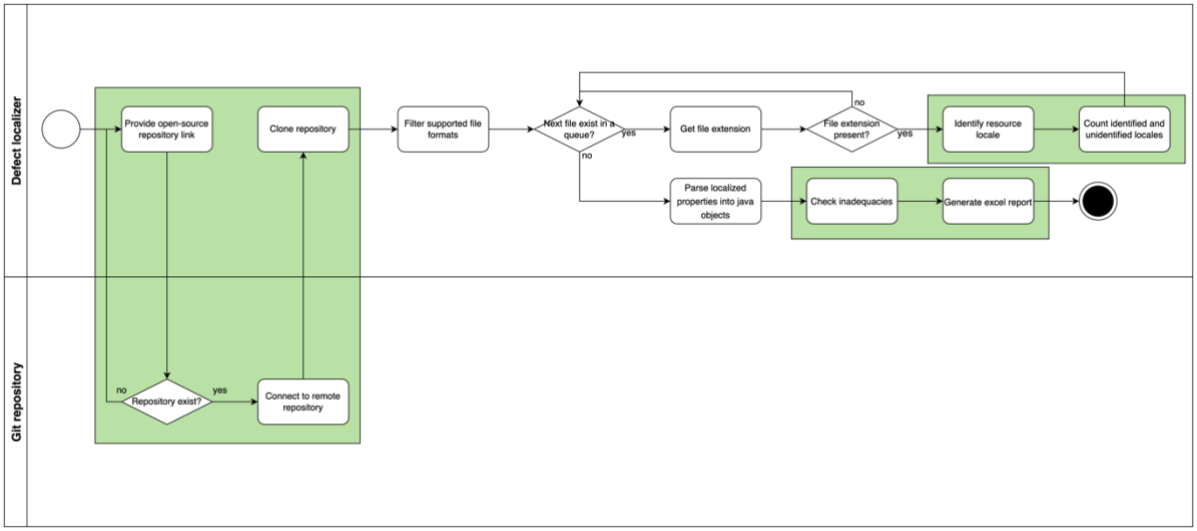


Figure 13. Open source repository inadequacy check flowchart

The steps involved in this process flow are as follows:

1. User provides a repository link via client request in the request query parameter. Service then checks if the repository exists. In case it does not exist – the user will get an error message that the repository with a provided link does not exist.
2. If the repository exists, service connects to it.
3. Git service clones repository into the file system.
4. Supported format repository files, which were mentioned before, are filtered, and passed to the locale detection service.
5. Locale detection service checks file extension and the presence of the locale in file naming and file column in Excel and CSV file formats.
6. Locale counter counts Lithuanian and unidentified locales.
7. Files are parsed in the parsing service. The parsing service then additionally filters out supported localized textual resource formats.
8. File content is parsed into *Property* Java objects.
9. *Property* Java objects are analyzed in inadequacy service where all checks are applied to ensure smooth localization QA.
10. Repository is then removed from the service file system.
11. Excel report generation service generates a report with all found issues and other reports.

3.5. Localization property grammar checks

The responsibility of inadequacy check service is to detect localization issues. As mentioned earlier, service analyses Java *Property* objects by applying all grammar rules to the property value. Inadequacy check service is comprised of multiple checks, and their detailed descriptions are presented in Table 2.

The inclusion of grammar checks in the service was based on an analysis of localization inadequacies that can be verified from the textual localization resources automatically by the service. These checks contain a set of several QA metrics of the MQM model and several (localized resource naming convention, spacing, and symbol usage) additionally added rules, which were illustrated in Figure 6. Grammar rules consist of the check name, severity if the rule is violated (Table 6), issue category, and explanation. Table 8 presents detailed insights into the manner and execution of how these rules are performed on localized properties.

Table 6. Localization properties grammar checks

Check	Severity	Category	Explanation
Localized resource file naming	Minor	Localized resource file naming	Applications, which are localized into many languages contain multiple localized resources. For better application source code structure maintainability, it is agreed to include locale code into the file name.
Specifying locale in the file column	Minor	Locale existence in file column	The rule is applied only for Excel and CSV file formats. Localized resources with specified file formats often contain localized *property* translations in multiple languages. It is agreed to specify locale code in the file column for better structure maintainability and localization tracking.
Property duplication	Minor	Localized properties duplication	Localized resources should not contain duplicated properties to avoid possible application errors.
Quotation	Minor	Use of non-Lithuanian quotation marks	Lithuanian quotation marks („ “) are often confused with English (single or double quotes).
Spacing usage after number (parameter)	Minor	Space after number (parameter) use	No space between the number and the following text. The number in

			the resource can be written directly or in the form of a parameter.
Spacing after period	Minor	Use of space after period	No space after a period, before the next word.
Multiplication sign	Major	Multiplication sign use	x is used instead of × as a multiplication sign.
Hyphen usage	Minor	Hyphen usage in number ranges	A hyphen (0-1) is used instead of a long hyphen (0–1) in the number ranges.
Number sign	Major	Use of number plate	The number uses a grid symbol or something else that is not suitable for marking the number. Correct either to "No" e.g., "Student No" or omit or replace the number and word e.g., "Requirement 5".
Thousands separator	Minor	Use of thousand separator	In Lithuanian language the thousand's grouping tab is a dot ".". Numbers are grouped in threes.
Decimal separator	Minor	Number decimal fraction part	The decimal separator for the decimal fractional part of numbers is a comma ",".
Symbol code in a text	Major	HTML symbol codes in a text	The sign is provided in HTML code, e.g. *&* or text contains HTML tags. The sign may need to be localized, which means to replace it with another word.
Non-normative lexis	Major	Jargon / non-normative term use in text	Use of jargon words / non-normative computing terms usage in the text. Change the term, which is used and approved in the Lithuanian language.
Capital letter use	Info	Capital letters in the middle of text	A capital letter within a segment (phrase, sentence). Capitalized words that are written outside of quotation marks, and outside of punctuation marks, are not distinguished by markup language tools. The exception is real nouns.
Date	Critical	Lithuanian locale date	Lithuanian locale has four date formats [Loc23a]: • Short. Pattern: "y-MM-dd". • Long. Pattern: "y 'm'. MMMM d 'd'.".

			• Full. Pattern: “y ‘m’. MMMM d ‘d’., EEEE”. Long and full format date months and weekdays should be translated.
Time	Critical	Lithuanian locale time	Lithuanian time formats are: • Short “HH:mm”. • Long “HH:mm:ss”.

Table 7. Localization properties grammar check severity explanation

Severity	Explanation
Info	The analyzed property was internationalized, double checking to ensure QA is needed.
Minor	The analyzed property was internationalized, but the issue was introduced by the translator.
Major	The analyzed property was not internationalized or left intentionally as it is.
Critical	Analyzed property issue was introduced by automated property creation without configuring to support Lithuanian locale specifics.

Table 8. Localized properties grammar checks explained

Check	Explanation
Localized resource file naming	To follow the localized resource naming convention, the rule checks if the file name (without extension) contains locale language code (lt), international component for Unicode (ICU) code (lt_LT), or Java locale code (lt-LT).
Specifying locale in the file column	To follow the localized resource maintainability quality property, Excel and CSV localized resources should contain locale language code, ICU code, or Java locale code in the first row as a header of localized properties.
Property duplication	Property Java object key and its corresponding value are checked for occurrences of duplication.
Quotation	A regular expression ((“|’)s*(“|’)), non-Lithuanian quotation marks, matched with a given text value.
Spacing usage after number (parameter)	A regular expression (\d+(?!$|\\.|\\s|[0-9]|'|\"|“|,|\\)|!|\\?|-|–|:|;)), no space between the number and the following text, is matched with a given text value.
Spacing after period	A regular expression (\.(?!$|\\.|\\s|[0-9]|'|\"|“|,|\\)|!|\\?|-|–|:|;)), no space after period, is matched with a given text value.
Multiplication sign	A regular expression ((d+)\s*[xX]\s*(d+)), wrong multiplication usage, is matched with a given text value.

Hyphen usage	A regular expression ((d+)\s*[-]\s*(d+)), wrong hyphen usage in a number range, is matched with a given text value.
Number sign	A regular expression (#(d+)), wrong number sign, is matched with a given text value.
Thousands separator	A regular expression (,[0-9]{3}), wrong thousand separation, is matched with a given text value.
Decimal separator	A regular expression ((\.\d+)+), wrong decimal separation, is matched with a given text value.
Symbol code in a text	Property value contains HTML tags and symbols.
Non-normative lexis	Property value contains non-normative or jargon Lithuanian word roots.
Capital letter use	Property value text is split into sentences. Each sentence is checked to contain multiple capital-case words.
Date	Localized applications into Lithuanian locale often tend to have English locale date formats. To locate such mistakes regular expressions are matched with a given text value: • Short. Pattern: d{4}/d{2}/d{2} or d{2}/d{2}/d{4}. • Long. Pattern: \b[A-Za-z]+ d{1.2}, d{4}\b. • Full. Pattern: \b[A-Za-z]+day [A-Za-z]+ d{1.2}, d{4}\b.
Time	Localized applications into Lithuanian locale often tend to have English locale time formats. To locate such mistakes regular expressions are matched with a given text value: • Short. Pattern and long. Pattern: \bd{1,2}:d{2}(:d{2})?\s*(am|pm)\b.

3.6. Localization issues analysis report

To provide a comprehensive analysis of issues related to the localization process Excel report generation service forms a comprehensive issue analysis report for both use cases. The report will be generated for both service use cases. The report consists of two sheets: the first containing insights obtained from locale detection service (see Figure 14) and a report highlighting issues in the localized resources (see Figure 15). The second sheet presents a column chart providing statistics on the issues detected (see Figure 16).

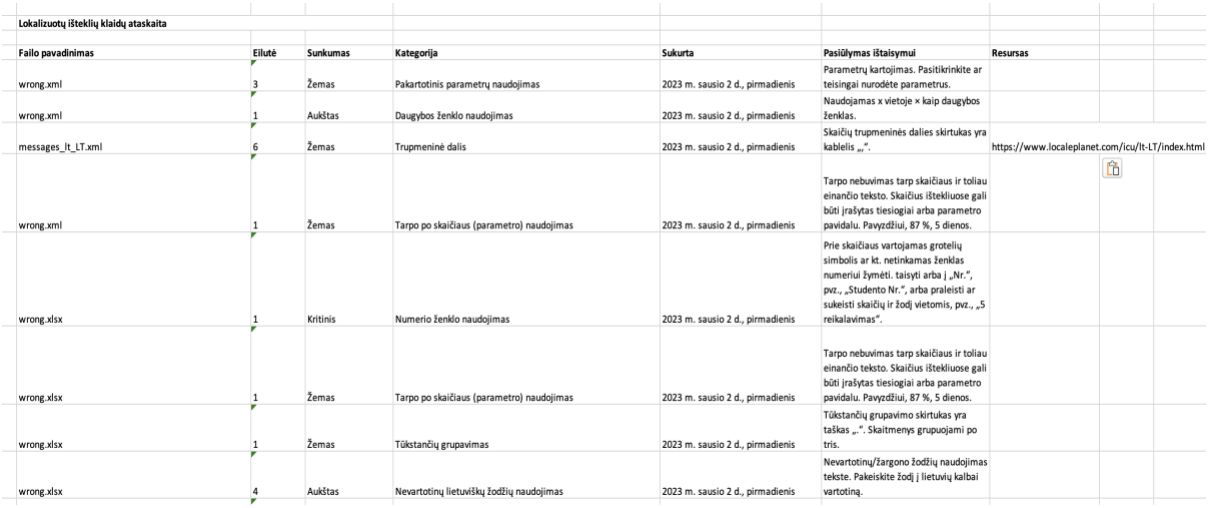
The locale detection report provides valuable information on the number of Lithuanian and unidentified localized resources locale that were identified during the process. This report presents the number of discovered Lithuanian and unidentified locales out of all examined files. Lithuanian or unidentified locales are counted for each resource, incrementing an appropriate counter by one for each one identified. The report presents a clear overview of the structure of

the localized resource naming convention, for user to understand localization process more effectively.

Lokalių atpažinimo ataskaita		
	Lietuviškų lokalių:	2/7
	Neatpažintų lokalių:	5/7

Figure 14. Locale detection report

A localized resources issue report is a comprehensive report of detected issues. The report provides a detailed explanation of the exact file and line number where the localization issue was discovered. Additionally, it categorizes the severity of the problem is and to which category it belongs. As a hand to overcome localization issues, the report provides recommendations for correction. For further assistance in the resolution of localization issues, the report also includes online resources that accommodate a deeper understanding of the problem at hand.



Lokalizuotų išteklių klaidų ataskaita

Failo pavadinimas	Eilutė	Sunkumas	Kategorija	Sukurta	Pasiūlymas ištaisymui	Resursas
wrong.xml	3	Žemas	Pakartotinis parametrų naudojimas	2023 m. sausio 2 d., pirmadienis	Parametrų kartojimas. Pasitikrinkite ar teisingai nurodėte parametrus.	
wrong.xml	1	Aukštas	Daugybos ženklo naudojimas	2023 m. sausio 2 d., pirmadienis	Naudojamas x vietoje × kaip daugybos ženklas.	
messages_lt_LT.xml	6	Žemas	Trupmeninė dalis	2023 m. sausio 2 d., pirmadienis	Skaičių trupmeninės dalies skirtukas yra kablelis „,“.	https://www.localeplanet.com/icu/lt-LT/index.html
wrong.xml	1	Žemas	Tarpo po skaičiaus (parametro) naudojimas	2023 m. sausio 2 d., pirmadienis	Tarpo nebuvimas tarp skaičiaus ir toliau einančio teksto. Skaičius išteklyje gali būti įrašytas tiesiogiai arba parametro pavidalu. Pavyzdžiui, 87 %, 5 dienos.	
wrong.xlsx	1	Kritinis	Numerio ženklo naudojimas	2023 m. sausio 2 d., pirmadienis	Prie skaičiaus vartojamas grotelių simbolis ar kt. netinkamas ženklas numeriui žymėti. taisyti arba į „Nr.“, pvz., „Studento Nr.“, arba praleisti ar sukeisti skaičių ir žodį vietomis, pvz., „5 reikalavimas“.	
wrong.xlsx	1	Žemas	Tarpo po skaičiaus (parametro) naudojimas	2023 m. sausio 2 d., pirmadienis	Tarpo nebuvimas tarp skaičiaus ir toliau einančio teksto. Skaičius išteklyje gali būti įrašytas tiesiogiai arba parametro pavidalu. Pavyzdžiui, 87 %, 5 dienos.	
wrong.xlsx	1	Žemas	Tūkstančių grupavimas	2023 m. sausio 2 d., pirmadienis	Tūkstančių grupavimo skirtukas yra taškas „.“. Skaitmenys grupuojami po tris.	
wrong.xlsx	4	Aukštas	Nevartotinų lietuviškų žodžių naudojimas	2023 m. sausio 2 d., pirmadienis	Nevartotinų/žargono žodžių naudojimas tekste. Pakeiskite žodį į lietuvių kalbai vartotiną.	

Figure 15. Localized resources issue report [in Lithuanian]

The issue statistics column chart is an effective means of gaining insight into the inadequacies found within various issue categories, as well as their frequency of recurrence. It should be noted, however, that any inadequacy categories that were not discovered during inadequacy check would not be represented in the column chart.

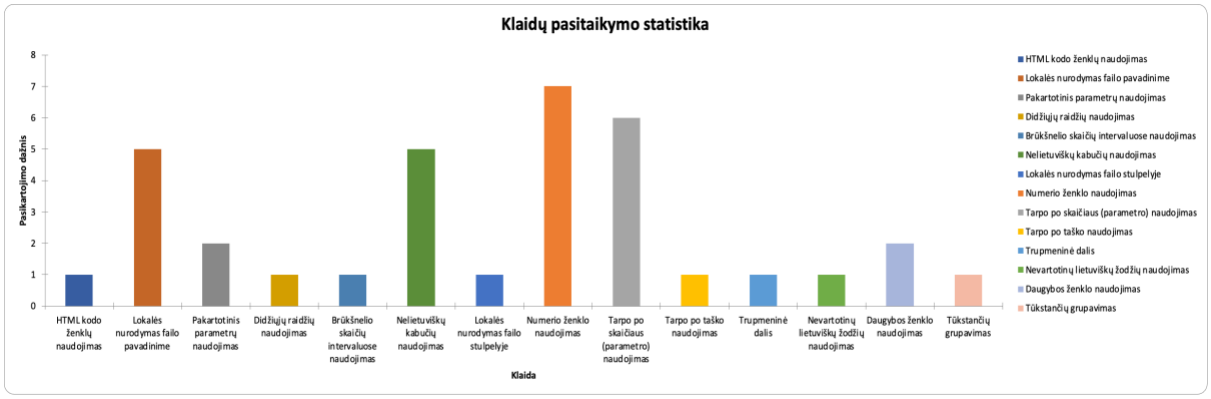


Figure 16. Issue statistics column chart [in Lithuanian]

Generated Excel reports are both comprehensive and user-friendly, even for non-technical users. Withing the file, there are three distinct report types that provide valuable insight and guidance to users, helping to better understand and avoid potential Lithuanian locale issues.

As described in the *high-level overview of a new service developed* section, the *Defect localizer* service is designed to address critical localization challenges by incorporating certain aspects of existing tools such as *Localise*, *LocalyzerQA* and *ErrorSpy*. More specifically, *Defect localizer* combines the localization process management (structuring, organizing, measuring, monitoring, and optimizing the process of adapting product for different target markets [Phr23]) features of *Localise*, error monitoring capabilities of *ErrorSpy*, and localization testing functionality of *LocalyzerQA*. Three distinct generated excel reports, which aid in the localization process management by structuring, organizing, measuring, monitoring, and optimizing detected issues. Additionally, the service accomplishes error monitoring goal through the generation of localized resources issue and issue statistics column chart reports. Lastly, *Defect localizer* verifies software behaviour, accuracy, and suitability for targeted Lithuanian locale (localization testing) through the generation of all reports. By combining the strongest aspects of the overviewed existing localization QA models and addressing previously discussed challenges, the *Defect localizer* service provides QA features, which are not offered by the existing tools.

3.7. Defect localizer technologies

To develop new *Defect localizer* service Java programming language technologies were used. Java long-time support (LTS) version 17 programming language used to implement and connect different frameworks: *Spring Boot*, *Gettext* PO, *Spring Open API*, *Logstash*, and *Apache POI* library. *REpresentational State Transfer* (REST) architectural style and *model-view-controller* (MVC) architectural pattern were applied to develop a well-organized service

source code structure. Service will be deployed to Google cloud computing provider's infrastructure *Kubernetes* orchestration service.

Java is one of the most popular programming languages in today's information technology industry. Designed to minimise dependencies, Java is a high-level, class-based, object-oriented programming language. It is a general-purpose programming language allows programmers to adhere to the "write once, run anywhere" (WORA) principle, eliminating the need for recompilation. Java code can be executed on various platforms that support Java. Typically, Java applications are compiled to bytecode that can be executed on any Java virtual machine (JVM) regardless of the underlying computer architecture. Therefore, these features help to implement new service using architecture best practises and to run it anywhere [Ora13].

Spring Boot is an open-source Java-based framework used to create micro service – standalone and production-ready applications. It became so popular of its known advantages [Tut22]:

- Easy to understand and develop java applications.
- Increases productivity.
- Reduces the development time.

Thus, *Spring Boot* framework was a choice to start developing a new *Defect localizer* high-performant service.

The *Gettext* PO toolset is utilized for the production, maintenance, and deployment of translation files by programmers and translators. The *Gettext* toolset provides many features, including support for developer comments, a context for multiple cases, and the ability to specify the location of a string in the source code [GNU23]. The simple structure and flexibility of the toolset have made it one of the most widely used translation file formats.

Spring Open API is a library, which help to automate generation of application programming interface (API – a way for two or more computer systems to communicate with each other) documentation using *Spring Boot* projects (see example in Figure 2). Library works by examining an application at runtime to infer API semantics based on spring configurations, class structure and various annotations, which helps for documentation and service user to understand API communication specifics [Lah22].

Logstash is another open source, light-weight data processing pipeline to ingest data from multiple sources. It transforms data and sends to the desired destination [Ela22]. The

library is used for better application log writing and tracking in deployed cloud computing service *Kubernetes*.

Apache POI Java library used for working with Microsoft Office formats – such as Excel, Word, and PowerPoint [ASF22]. New service uses this library and its features to read, write and format the Excel template report with detected issues and other reports.

JGit is pure Java library implementing Git version control system [Ecl22]:

- Repository access routines.
- Network protocols.
- Core version control algorithms.

JGit has very few dependencies, making it suitable for embedding in any Java application, whether the application is taking advantage of other *Eclipse* technologies [Ecl22]. The library was used to check if the user-provided open-source repository exists. In case it exists, the library helps to clone source code into the service file system.

Kubernetes are open-source framework for automating application deployment, scaling (resources), and management [Kub22]. *Kubernetes* framework was used to deploy *Defect localizer* service using *Google* cloud computing provider. The system helps to monitor product states, to change the scale freely – to automate the increase of computer resources during high user traffic. At the same time, it allowed manual work to be automated and thus save time and labour resources.

Service adheres to the representational state transfer application programming interface (REST API) communication pattern, which is an architectural style employed in distributed hypermedia systems. An API is a collection of definitions and protocols that facilitate the development and integration of application software. It is often referred to as a contract between an information provider and an information user – establishing the content required from the consumer (the call) and the content required by the producer (the response) [RH20].

MVC stands for model-view-controller. MVC design pattern specifies that an application consist of a data model, presentation information, and control information (see Figure 17) [Her21]:

- **Model**. The backend that contains all the data logic.
- **View**. The frontend or graphical user interface (GUI).
- **Controller**. The brains of the application that controls how data is displayed.

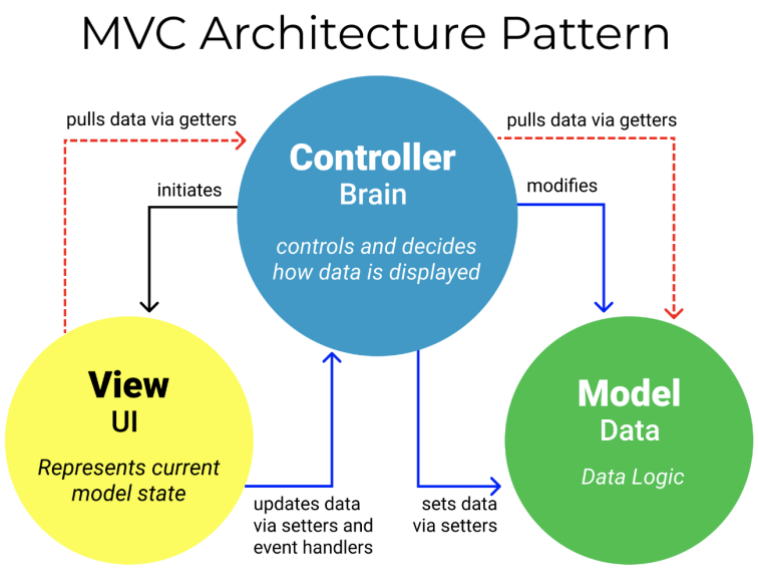


Figure 17. MVC pattern high-level architecture [Her21]

MVC architectural pattern was used to maintain a well-organized service source code structure and separation of concerns. The pattern has other advantages for e.g.: applications following this pattern are easily modifiable, and maintainable and the development process of MVC applications is faster.

3.8. Scientific and practical novelty

Defect localizer service offers novel features that distinguish it from existing localization (internationalization) quality assessment products analyzed in the literature review section of this study. The service is designed to be lightweight and automatically detect and correct critical localization issues while complying with Lithuanian locale cultural norms.

The following scientific novelties were developed during the work:

- New QA model, combining ISO 9000, SAE J2450, LISA QA, MQM model, and several additionally included aspects, was developed.
- Lithuanian language localization quality evaluation.
- Issue reporting with three different sub-reports and solution guidance.
- Issue statistics.

The development of the new model implementing service attained practical innovations:

- Automated Lithuanian language localization testing.
- Open source project localization testing.
- Public free-of-charge solution.
- Jargon / non-normative Lithuanian lexis detection.

The development of *Defect localizer* service has enabled to achieve of several significant goals discussed at the beginning of the *localization quality assurance* chapter. First, it replaces the need for manual localization QA checking. By automating the localization QA process users are provided with a faster and more efficient way to identify and fix localization issues. Additionally, the service offers localization QA support specifically for the Lithuanian language, which has been underestimated in existing localization QA models. This ensures that Lithuanian users have access to a reliable service that can help to address critical localization issues. Furthermore, using a *Defect localizer* can help reduce the high costs associated with existing models. And finally, the service provides QA for critical localization aspects, which ensures that localized resources are accurate and culturally appropriate for the intended audience.

4. Experimental evaluation of *Defect localizer*

Overviewed new model and *Defect localizer* service in the *localization quality assurance* section aims to replace manual localization testing and improve the user experience of Lithuanian-speaking end users. This study will provide an evaluation and comparison of automated localization testing results with *Defect localizer* performed on open source software projects localized into Lithuanian with manual localization. The study will assess existing localization model translations with the new service. Moreover, the outcomes of the study will be compared with existing localization QA product to determine the necessity of a new service to improve the localization process. In addition, the findings of the study will be shared with the focus group (localization experts) to collect valuable insights into their practices and the demand for such a service. Finally, this study will identify the advantages of using the *Defect localizer* service over existing localization QA models.

4.1. Localization testing with *Defect localizer*

Localization testing is the process that ensures that a software application or website is usable and adapted to the local region to which it was localized [Loc21]. In this study, we aim to validate and verify the effectiveness of a new localization testing service through a thorough set of steps. Firstly, we will analyze and test Lithuanian open source projects from a popular *GitHub* source control management system (SCMS) with a newly developed service. Secondly, we will compare automated localization QA with traditional manual localization testing. Lastly, we will assess translations for open source projects using existing *QA Distiller* localization QA tool and compare them with *Defect localizer* testing results.

To validate and verify the effectiveness of the *Defect localizer* service, we will start by selecting and analyzing Lithuanian open source projects from *GitHub* SCMS. This step will involve the automatic detection and correction of localization issues and the generation of comprehensive Excel reports assisting users in overcoming and avoiding Lithuanian language localization errors. We will compare manual versus automated localization QA to identify the strengths and limitations of the new service.

Finally, we will evaluate translations from open source projects made by existing *QA Distiller* localization tool. This step will involve an in-depth analysis of the translation quality and accuracy generated by this service and its suitability for Lithuanian localization. Analysis outcomes will be shared with localization experts and compared with *Defect localizer* testing results.

Following these steps, we aim to provide a comprehensive evaluation of the effectiveness of our newly developed service in detecting and correcting localization issues, ensuring user experience improvement for Lithuanian-speaking users. The results of our study will present valuable insights into the strengths and weaknesses of automated localization testing to help to identify areas for improvement in the newly developed localization QA model.

4.2. *GitHub* open source projects localization testing with *Defect localizer*

For this study, *Linux KDE* and several randomly selected *GitHub* open source projects that had been localized in the Lithuanian language were chosen. Over one hundred projects were tested and analyzed, involving the examination of over half a million localized text strings. To collect and analyze testing results comprehensive Excel reports were used, including examples of localization issues and statistics. To determine the total number of errors, an issue statistics chart from each Excel project report was used, as illustrated in Figure 18.

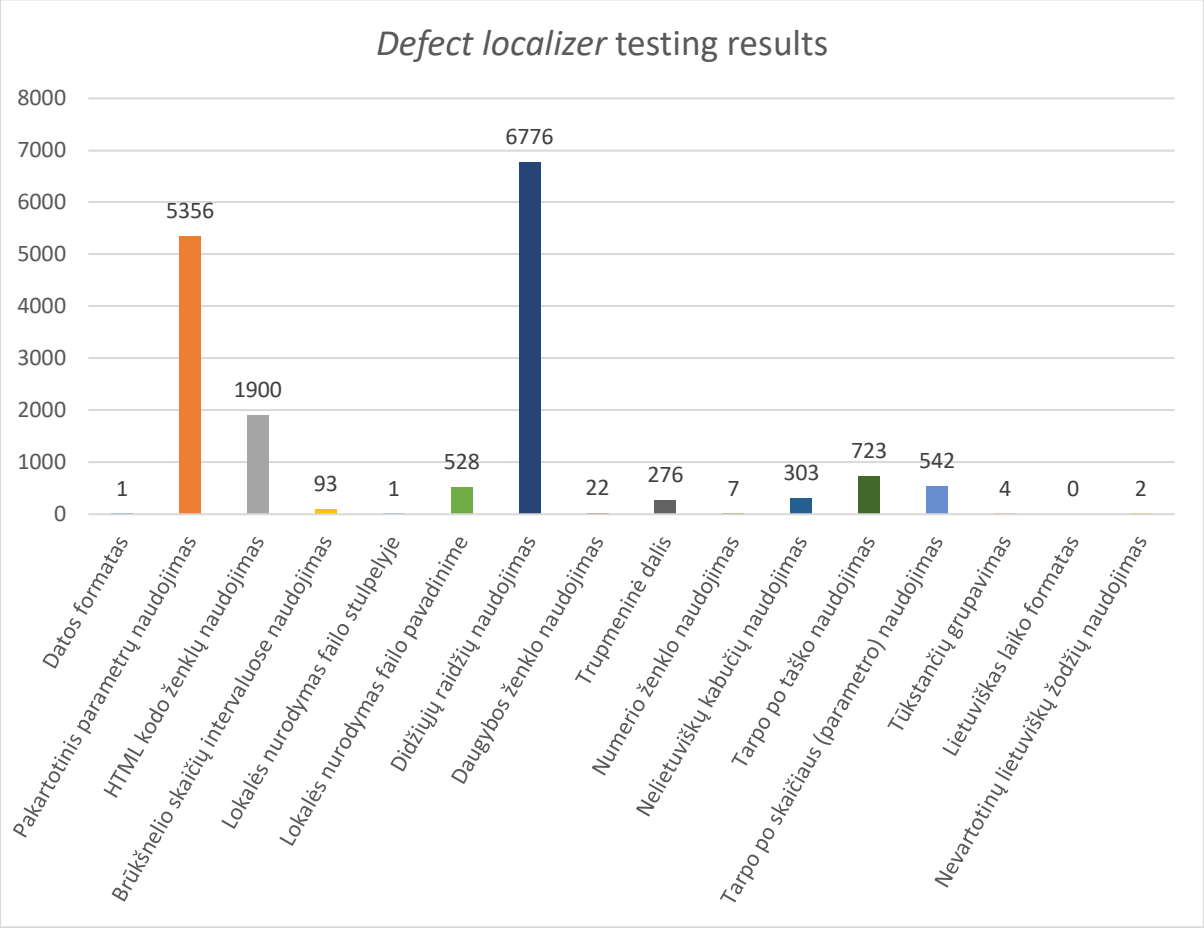


Figure 18. Defect localizer testing results (in Lithuanian)

Linux KDE projects are developed and maintained by the open source *KDE* community, which is responsible for sustaining over 200 various applications. Localization of the majority

of these applications into the Lithuanian language is carried out by translators, developers, or community volunteers. In the context of this study, we focused on assessing the applications that have undergone nearly complete or full localization in Lithuanian using the *Defect localizer* service.

To evaluate *Linux KDE* and several other projects, we used the *Defect localizer* open source repository use case request. This involved submitting the HTTPS link of each project in the request query parameter, after which the service downloaded the localized textual resources in Lithuanian and parsed them into *Property* Java objects for further analysis. The majority of the textual resources were localized using PO file format, with a few using CSV. During the parsing process, the service removed all comments, HTML code, headers, and empty property translations. While some applications were relatively small, containing less than a thousand localized strings, the majority were considerably large, with more than 1000 strings and some even exceeding 10000. A total of 533715 localized text lines were examined.

Upon analyzing open source projects *Property* Java objects all rule violations were collected to generate Excel reports. An issue report was then generated to analyze the collected errors and their respective locations, with examples of such issues presented in Table 9. In total, 16534 issues were identified through all possible *Defect localizer* service QA checks.

Table 9. Defect localizer identified issues examples (in Lithuanian)

Check category	Issue examples
Localized resource file naming	kio_jabberdisco.po, kopete.po, systemdgenie.po
Specifying locale in the file column	No headers
Property duplication	Ne, Lygis, Užmigdyti, Nepavyko, Nėra, Adresas, Ištrinti, Nežinomas dydis, Taip, Grupė, Tipas.
Quotation	Automatiškai įjungti būseną "pasitraukęs", Ar seniau naudojote "%1" kaip slapyvardį?, Derinti Python modulį iškviečiant jį su '-m', Serveris pranešė: "%1".
Spacing usage after number (parameter)	%3v %2m %1s. Ilgą laiką išlaikant akumuliatorių 100% įkrautą, akumuliatorius gali būti greičiau sugadintas. Paskirties failų sistema palaiko tik failus iki 4GiB dydžio, %1mėn, %1sav. False positives: Mp3tunes, Klaida: %1%2.
Spacing after period	Šis tekstas rodomas Vietų skydelyje.Aprašymą turėtų… Ši piktograma bus rodoma Vietų skydelyje.Spustelėkite ant mygtuko. False positives: „.part“, įveskite .kde.org, andrius@stikonas.eu, liudas@akmc.lt, Garso failai (*.ogg *.wav).

Multiplication sign	Įterpti 3x3 lentelę, 2x2, 1x1, 1x1 (mažiausias failas), Minimali (640x480), Maža (800x600), Įpasta (1024x768), Didelė (1280x1024), Labai didelė (1600x1200).
Hyphen usage	2010-2018, (c) 1998-2000, 1996 - 2001. False positives: ISO-8859-5 Kirilica, ISO-8859-14, ISO-8859-7 Graikų.
Number sign	False positives: &#177, &#8226, &#946, PKCS#12.
Thousands separator	9,999+, False positives: 2005,2009
Decimal separator	0.0000017, 1.6749286, 0.0000010, PI() lygu 3.141592654. False positives: Amarok 1.4, RSS 1.0 versijos kanalas.
Symbol code in a text	Visada uždraudžia vertikalią slinkimo juostą. <qt>Turite įvesti teisingą kompiuterio vardą.</qt>. <html><body><I>Naudotojo informacija nepateikta</I></body></html>. Sukurta:
Non-normative lexis	CD
Capital letter use	Antras žingsnis: Paskyros informacija. Eksportuoti į Adresų knygelę. Pagrindinis Langas. Jingle Video skambutis. Python: Dabartinis failas. False positives: David Faure, Stephan Kulow, Bernhard Rosenkraenzer.
Date	Atpažįstamų datų pavyzdys: 05/31/2008 15:24:30.

To analyze the occurrence of issues, a column chart with statistics was utilized after issue identification analysis. It provided statistics on the frequency of various issue categories for each project. The results of the study revealed 6776 issues related to capital letter use, 5356 instances of property duplication, 1900 occurrences of symbol code in a text, 723 spacing errors after a period, 542 spacing errors after a number, and 528 issues related to localized resource file naming, 303 quotation errors, 276 problems related to decimal separation, 93 hyphen usage errors, 22 issues related to multiplication sign usage, and only a few errors related to dates, specifying locale in the file column, number sign, thousands separation, and non-normative lexis.

Following this assessment step, we analyzed various open source *Linux KDE* and several other projects from *GitHub* SCMS. Over 100 projects were examined, with more than half a million of localized lines analyzed to collect and analyze results. During the analysis of these projects, we presented examples of identified issues, along with the frequency of various issue categories.

4.3. Manual versus automated testing with *Defect localizer*

In the *localization quality assurance process* section, we analyzed the traditional manual localization testing process. Manual localization testing is labour-intensive and time-consuming [OA13], and automation has been the demand for a long time [Sin17]. In this section, we will compare traditional manual and automated localization testing with the *Defect localizer*.

One of the major differences between the automated localization testing approach of *Defect localizer* and manual testing is time. To perform automated localization testing, we noticed that the *Defect localizer* can provide bug reports within minutes, depending on the size of the project, while manual testing of each button, window, or modal may take hours or days. To test more than 100 projects containing over half a million lines, it took one day using *Defect localizer*, while manually reading and testing the same amount of text reading could take up to 100 times longer. Automated testing increases efficiency, especially in regression testing [Kar11], as it can quickly verify existing translations, which may be missed in manual testing due to the human factor. Consequently, less labor time also leads to faster time to market.

Automation decreases costs by relieving the manual workload: provides the possibility to execute more tests in less time and fluent reuse of testing scripts [Kar11]. Testing open source projects with *Defect localizer* allowed us to re-execute and reuse testing scripts multiple times, without human intervention, enabling us to save time and effort. In the long run, it can lead to a reduction in software development costs for organizations.

In general, we see that the literature agrees that test automation is plausible for saving labour time, enhancing quality, and reducing costs in the long run [Kar11]. The development of a new QA model helped to achieve multiple of our objectives: automate localization QA checking, localization QA support for the Lithuanian language, and help to reduce high localization costs.

4.4. *Github* open source projects localization testing with *QA Distiller*

To compare localization testing results with the Defect localizer for the translation assessment step, we chose *QA Distiller*. *Linux KDE* and several other open source projects were utilized in this step. In total, we analyzed over a hundred projects, along with more than half a million localized lines, using this QA tool. To analyze the testing results and determine the total number of each error category, we used *QA Distiller* HTML reports with identified issues and issue occurrence rate table, as illustrated in Figure 19.

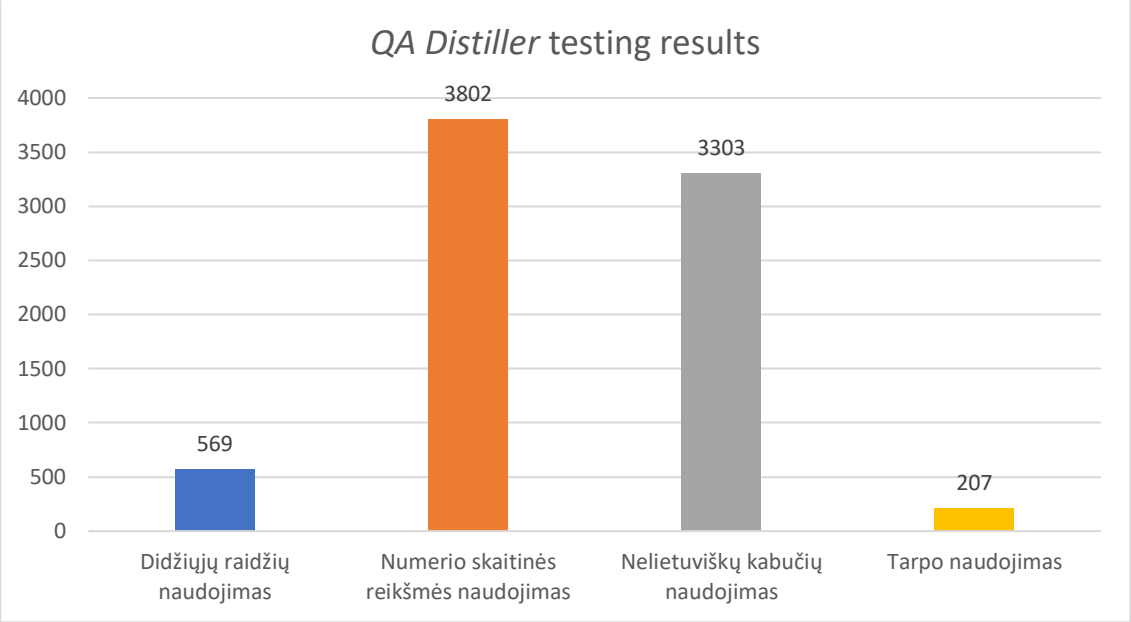


Figure 19. QA Distiller testing results (in Lithuanian)

In the *overview of localization (internationalization) quality assessment products* section, four different localization QA tools were reviewed. In order to compare the testing results gathered during the analysis of open source projects localized in the Lithuanian language with a *Defect localizer*, *QA Distiller* was selected as a choice. This decision was made based on several reasons. Firstly, it was noted that *LocalyzerQA* and *ErrorSpy* do not support localization testing with the Lithuanian language. Secondly, the *Localise* tool is mainly focused on managing the localization process rather than providing QA checks. Therefore, it was not worth comparing results with these products. Consequently, it was determined that the QA model of the *QA Distiller* tool was more similar to the *Defect localizer* due to its rule-based QA approach.

In Table 10, we compared the QA checks offered by *Defect localizer* and *QA Distiller*. We observed that the majority of the checks available in the *Defect localizer* are not supported by *QA Distiller*. Furthermore, some checks are more comprehensive and combine multiple categories, such as spacing (spacing usage after number and period in *Defect localizer*) or numerical (thousand and decimal separator in *Defect localizer*) category checks. Additionally, *QA Distiller* allows users to upload a set of non-normative or terminology for matching and identifying non-normative / jargon issues with a given text.

Table 10. Defect localizer QA check comparison with QA Distiller

Check category	QA Distiller
Localized resource file naming	-

Specifying locale in the file column	-
Property duplication	-
Quotation	+
Spacing usage after number (parameter)	+
Spacing after period	+
Multiplication sign	-
Hyphen usage	-
Number sign	-
Thousands separator	+
Decimal separator	+
Symbol code in a text	-
Non-normative lexis	+
Capital letter use	+
Date	+
Time	-

It is important to note that the QA check process of the *QA Distiller* differs from that of the *Defect localizer*. During the text reading phase, *QA Distiller* tests not only the actual localized text but also all comments, HTML code, headers, and empty property translations are also being tested. Consequently, the results of identified issues vary, and identified issues are not as accurate as compared to those of the *Defect localizer*.

To analyze the occurrence of issues, an issue occurrence rate table in the HTML report (Figure 20) was utilized after the issue identification step, with examples of such issues presented in Table 11. It provided statistics on the frequency of various issue categories for each project. The results of the study revealed 3802 numerical, 3303 instances of spacing errors, 569 occurrences of multiple capital letter use in a text, and 207 spacing issues.

Table 11. QA Distiller identified issues examples

Check category	Issue examples
Quotation	Persijungti į "Insider" kasdienį kanalą, Neveiklus - '%1', Albumas "%1" iš %2, Nepavyko inicializuoti iPod'o, Serveris pranešė: "%1". False positives: <h1 align='center' style='font-size: large; text-decoration: underline'>Legend:</h1><ul type='square'>, <emphasis strong='true'>
Spacing	42°C, 2m, Tipas 1C.
Numerical	MIME-Version: 1.0, 2022-11-17 23:34+0200, X-Generator: Lokalize 1.5, PI() lygu 3.1415926.
Capital letter use	%1 OA (RAM), I,II,III, VCD & AHD, GMT+12:00, N/A, DEV BUILD.

Error summary		
	Discarded	**Genuine**
Capitalisation	**0**	**4**
Corrupt character	**0**	**9**
Information	**1**	**0**
Measurement unit	**0**	**9**
Number formatting	**0**	**43**
Number value	**0**	**337**
Quotes	**0**	**195**
Regular expression	**0**	**70**
Terminology	**0**	**93**
Word spacing	**0**	**10**
Total	**1**	**770**

Figure 20. QA Distiller HTML issue occurrence rate table example

In the context of this evaluation phase, where we analyzed various open source *Linux KDE* and several other projects. While examining over 100 projects, which contained more than half a million localized lines, we discovered numerous quotation mark, spacing, numerical and capital letter use issues. These errors were identified not only in the HTML code in a given text but also in the actual text itself, therefore, results are not as precise as compared with the *Defect localizer*. A detailed comparison of results between *QA Distiller* and *Defect localizer* will be presented in the next section.

4.5. Defect localizer versus *QA Distiller* localization testing

Both *Defect localizer* and *QA Distiller* employ LISA QA and MQM QA models. To validate and verify the results, we used both tools. However, as the QA check set and localization evaluation process differ between the two tools, we obtained different results for each. In this section, we will review the testing outcomes.

In Table 11, a comparison was made between the QA checks offered by the *QA Distiller* and the *Defect localizer*, highlighting the differences in their functionalities. While both tools support quotation checks, *QA Distiller* performs this check not only in the actual text but also in HTML code and file headers, leading to more of issues in these areas. Similarly, *QA Distiller* performs a spacing check. However, it is important to note that the *Defect localizer* also identified some false positive issues, e.g., spacing after period in emails, web page hyperlinks, file extensions, spacing after number in application names, and multiple parameter bindings.

Defect localizer checks for property duplications and symbol code in text usage. While performing the validation and verification phase, our analysis revealed a total of 5356

occurrences of duplication and 1900 instances of symbol code in a text. Identifying and resolving instances of duplication and symbol usage in a text could result in savings in space, as well as improve the organization and maintainability of file structure. Modern content management systems can format text within a source code, making it unnecessary to store formatting information in localized resources. Consequently, duplication and symbol in a text check is an important aspect of localization QA.

In our experiment phase using *Defect localizer* to perform multiplication sign, hyphen usage, we observed that when localizing applications into Lithuanian language, translators often use the English version of hyphen and "x" letter instead of the correct Lithuanian multiplication sign. We also found that there were false positive mistakes in standard, language ISO codes when performing hyphen check.

Number sign QA check yielded false positive issues in our experiment. We discovered that these issues were detected in HTML symbols within the actual text. Additionally, the thousand and decimal separator check results also indicated that translators tend to use the English version of thousand and decimal separators. We also identified some false positive issues in thousand separator mistakes between two dates, which are separated by a comma without a space and decimal separation in application version numbers. While the majority of *QA Distiller* numerical checks identified false positive issues in file headers, header dates, and application versions.

In the validation and verification phase, while performing a non-normative check with a *Defect localizer*, 2 issue occurrences were identified. However, *QA Distiller* could not find any. Both tools support date check, but with the *Defect localizer*, only one instance was found, while *QA Distiller* did not find any.

Additionally, both tools support capital letter use check. The *Defect localizer* checks if multiple words are starting with a capital letter within a sentence, while *QA Distiller* only finds fully capitalized words. As a result, the *Defect localizer* identified more occurrences of this issue, although some false positives were identified with names and surnames.

Based on the examination of over 100 projects containing more than half a million lines of text, it was observed that *QA Distiller* is not yet mature enough and suitable for performing localization quality assessment for the Lithuanian language. The study discovered numerous false positive issues identified in the HTML code and the actual text itself. Additionally, some critical aspects, such as thousand and decimal separation, spacing after a period, and number (parameter), were missed by *QA Distiller*. As a result, *Defect localizer's* quality assurance is

essential in ensuring that localized software products meet the needs and expectations of global audiences.

4.6. Sharing findings with localization experts: insights into practices and *Defect localizer* service demand

Based on comprehensive validation and verification set of steps, we determined that currently, there is no equivalent tool for direct comparison with the *Defect localizer*. To gain insights into localization practices, methods, and tools employed by industry experts, we conducted interviews with several professionals. We introduced our newly created QA model and presented the experimental evaluation results of the *Defect localizer* service. To prove the necessity of the *Defect localizer* service, we utilized qualitative research method and asked to provide feedback from the experts.

A focus group consisting of two software engineers and one researcher, all having expertise in the field of software localization, were interviewed. The participants were asked to share methods, tools, quality assurance models, and practices they employ in the localization process of software. Additionally, they were asked to identify any specific features they lack in mentioned tools and quality assurance models. One of the software engineers and researcher rely on manual software localization methods and utilize tools that do not offer automatic localization quality assurance resulting in significant manual effort investment to verify the accuracy of localized content. Furthermore, all participants expressed their agreement on the absence of “automated QA check" and guidance for resolving issues in the tools they currently use.

Following the presentation of the newly created QA model and its implemented service, participants were asked to provide their perspectives on the potential success of *Defect localizer* in the Lithuanian market and the identified advantages. One of the software engineers and researcher expressed the belief that “small local business is looking for qualified software with good business language translation. It saves user time and is provided with the report of detected issues, their locations, guidance on how to overcome them and issue statistics”. The new service could contribute to the success of small local businesses by saving user time verifying the localized resources and providing comprehensive reports. Participants believe that the *Defect localizer* implementation has the potential to effectively reduce localization issues.

Subsequently, participants were asked to share their perspectives on why the Lithuanian language is often overlooked in the existing QA models and why manual localization testing

remains dominant. One of the software engineers speculated: “models are prepared of not native Lithuanian language speakers”. The lack of native Lithuanian speakers in the development of existing QA models could be a contributing factor. Another software engineer believes that the relatively small size of the Lithuanian market and the complexity of the language might contribute. Furthermore, participants expressed the belief that “there are no good tools for translation assessment”. The absence of robust automatic localization testing tools is a key factor. They also emphasized that the creation of a new tool would demand significant effort and might not yield enough value compared with manual testing.

Furthermore, participants were requested to share their perspectives on additional features they would like to see incorporated into the new QA model. One software engineer expressed his viewpoint, stating that the: “translation assessment should be done in the business language aspect“. This suggestion highlights the desire for the new service to validate the translations in the business language use.

Based on the insights gathered during the interview, we conclude that the majority of industry professionals rely on manual localization testing and express demand for an automatic alternative. Participants believe that the new *Defect localizer* service has the potential for success in the market due to its ability to address and mitigate Lithuanian localization issues. Moreover, they think that the service would reduce testers' effort in verifying the accuracy of localized resources.

4.7. Advantages of Defect localizer and further improvements

Through a comprehensive set of steps in the validation and verification phase, we have identified several advantages in comparison to existing localization QA models. The development of the *Defect localizer* has allowed us to successfully achieve our established set of goals. However, during the experimentation phase, we identified a couple of areas for potential improvements, to simplify activities of testers.

One of the most significant advantages is the automation of localization testing, which allows for a faster and more efficient way to identify and resolve localization issues. Additionally, the service offers localization QA support specifically for the Lithuanian language, which has been overlooked in existing localization QA models. This helps to ensure that Lithuanian users have access to a reliable service that can address critical localization issues. Moreover, our service proved effective in identifying genuine Lithuanian localization issues, which can lead to user experience improvement for Lithuanian-speaking users. Furthermore, our service is provided free of charge, potentially reducing the high costs of

existing localization QA models. Finally, the service provides QA for critical localization aspects, ensuring that localized resources are accurate and culturally appropriate for the intended Lithuanian audience.

During the validation and verification phase, several false positive edge cases were identified in spacing usage after the number (parameter) and period, number sign, thousand and decimal separator, and capital letter use checks. To improve spacing usage after the number (parameter) and period, also for capital letter use and number sign checks, we propose allowing users to upload terminologies and exception words. This could prevent identifying spacing usage after a number issues in application names with numbers, parameter bindings, spacing after period errors in emails, web page hyperlinks, file extensions, capital letter use issues in names and surnames, and lastly, number sign in HTML symbols. This could also benefit the decimal separation check, as users could define their application name with a version in the exclusion list. Furthermore, to improve checks for spacing after period issues, we suggest having a predefined set of extensions. To improve the thousand separator checking between two dates separated by a comma without a space, we propose correcting the regular expression to match numbers divided by threes only.

In the validation and verification phase, it was found that no non-normative lexis errors were detected. However, to improve this check, we propose the implementation of a predefined list of non-normative or jargon words, including English words with Lithuanian endings.

To conclude, based on the comprehensive validation and verification phase, we believe that *Defect localizer* has the potential to succeed in the market of Lithuanian localization QA models with a few key improvements. The *Defect localizer* was utilized with more than 100 open source projects localized in Lithuanian. The service demonstrated the ability to ensure that localized software products meet the needs and expectations of global audiences, contributing to improved user experiences and increased market success.

Results and conclusions

In this research work, new localization quality assurance model and its implementation, automating the detection of the selected localization issues in the textual localizable software resources of the Lithuanian locale, have been developed.

The following results were achieved during the work:

1. Specific locale elements that can be automatically tested and used in the development of a new QA model were identified.
2. New QA model, combining several aspects of ISO 9000, SAE J2450, LISA QA, and MQM models, has been developed.
3. A new QA model implementation has been developed to detect and fix localization issues. The development of a new service helped us to achieve several goals:
    3.1.automate localization QA checking.
    3.2. provide support for the Lithuanian language localization QA.
    3.3. reduce the high costs of existing localization QA models.
    3.4. provide QA for critical localization aspects.
    3.5. improve the user experience of Lithuanian-speaking users.
4. The new model implementing service has been practically applied with over 100 open source projects to identify and fix Lithuanian locale localization issues.

After obtaining these results, we conclude that:

1. Lithuanian language is overlooked by existing localization QA models and tools.
2. During the software localization into Lithuanian, many cultural elements use English-based locale conventions and formatting.
3. Localized resources contain many property duplications and HTML symbols.
4. Localization industry experts rely on manual localization testing and express a belief in automatic *Defect localizer* potential to address and mitigate Lithuanian localization issues.
5. The amount of bugs found in publicly distributed localized software resources confirms the need for such a service.

The new localization QA model allows us to implement localization QA evaluation service for the Lithuanian language. The new service can assess localized textual resources

using two cases. Using these cases, it is possible to identify and improve the user experience of Lithuanian-speaking users. Service simplifies the activities of testers, and the localization QA becomes more efficient and reliable. The developed service contributes to enhancing quality of software localization to Lithuanian locale.

References

[ASF22] Apache software foundation. Apache POI – the Java API for Microsoft Documents (2022). [visited 2022-11-21]. Access through the Internet: <https://poi.apache.org/index.html>

[BC21a] Baeldung community. A guide to POSIX (2021). [visited 2022-05-17]. Access through the Internet: <https://www.baeldung.com/linux/posix>

[BC21b] Baeldung community. Internationalization and localization in Java 8 (2021). [visited 2022-05-20]. Access through the Internet: <https://www.baeldung.com/java-8-localization>

[Ber21] Berger, C. F. (2021). Software localization (L10N) quality assurance from the tester's perspective. CFB scientific translations.

[BS13] Bhatia, M., Sharma, A. (2013). A generalized quality model for localized software product. Research Inventory: International Journal of Engineering and Science. Vol.3, Issue 10 (October 2013), PP 37-46.

[Bla21] Blackwood, M. (2021). 7 most common localization problems in websites and software. [visited 2021-11-07]. Access through the Internet: <https://www.c-sharpcorner.com/article/7-most-common-localization-problems-in-websites-and-software/>

[BOK20] Bayram, B., Ozkan, H., and Kacabas, I. (2020). An investigation of localization testing processes in the software industry. International journal of advanced computer science and applications, 434-442. DOI: 10.14569/IJACSA.2020.0111156

[Car98] Carey, J. Creating global software: A conspectus and review. Interacting with Computers, Special Issue: Shared values and shared interfaces, 9, 4, 1998, p. 449–465.

[CM22] Couto, M., R., L., Miranda, B. (2022). Towards improving automation support for internationalization and localization testing. Anais Estendidos do Simpósio Brasileiro de Qualidade de Software conference. DOI:10.5753/sbqs_estendido.2022.227653.

[DGJ10] Dagiene V., Grigas G., and Jevsikova T. (2010). Programinės įrangos lokalizavimas. Matematikos informatikos ir institutas.

[DG06] Dagiene, V., Grigas, G., (2006). Software Lithuanianization. State commission for Lithuanian language. [visited 2022-05-13]. Access through the Internet: <http://www.vlkk.lt/naujienos/kitos-naujienos/programines-irangos-lietuvinimas>

[DJ09] Dagienė, V., Jevsikova, T. (2009). Cultural Elements in Internet Software Localization. In: Lenca, P., Brézillon, P., Coppin, G. (guest eds.) Revue d'intelligence artificielle. Human-centered processes – Current trends. Volume 23 – no 4/2009. Hermes – Lavoisier, p. 485–501.

[DOG22] Dokumentation ohne Grenzen. ErrorSpy quality assurance. [visited 2022-06-09]. Access through the Internet: <https://www.dog-gmbh.de/en/products-shop/errorspy-quality-assurance/>

[DOG23] Dokumentation ohne Grenzen. Quality assurance with ErrorSpy. [visited 2023-03-10]. Access through the Internet: <https://www.dog-gmbh.de/technologien/qualitaetssicherung-mit-errorspy/>

[Ela22] Elastic. Logstash: centralize, transform & stash your data (2022). [visited 2022-11-21]. Access through the Internet: <https://www.elastic.co/logstash/>

[Ecl22] Eclipse. Eclipse JGit (2022). [visited 2022-12-28]. Access through the Internet: <https://www.eclipse.org/jgit/>

[Ess02] Esselink, B., (2002). The evolution of localization. Lionbridge. [visited 2022-05-07]. Access through the Internet: <http://www.intercultural.urv.cat/media/upload/domain_317/arxius/Technology/Esselink_Evolution.pdf>

[GFG20] Geeks for geeks. Limitation of distributed system (2020). [visited 2021-11-07]. Access through the Internet: <https://www.geeksforgeeks.org/limitation-of-distributed-system/>

[GNU23] GNU org. The format of PO files. (2023). [visited 2023-03-28]. Access through the Internet: <https://www.gnu.org/software/gettext/manual/html_node/PO-Files.html>

[Hall02] Hall, P. (2002). Bridging the digital divide, the future of localization. Electronic Journal of Information Systems in developing countries. [visited 2022-05-03]. Access through the Internet: <https://www.researchgate.net/publication/277856000_Bridging_the_Digital_Divide_the_Future_of_Localisation>

[Her21] Hernandez, R. D. (2021). The Model View Controller Pattern – MVC Architecture and Frameworks Explained. [visited 2023-01-02]. Access through the Internet: <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>

[IBM16] IBM corporation. Test cases, test suites, and test execution (2016). [visited 2023-01-05]. Access through the Internet: <https://www.ibm.com/docs/en/elm/6.0.3?topic=testing-test-case-test-suite-overview>

[ISO99] ISO/IEC 15897 (1999). Information technology – Procedures for registration of cultural elements.

[ISO15a] ISO/IEC 9000 (2015). Quality measurement systems – fundamentals and vocabulary.

[ISO15b] ISO/IEC 17100 (2015). Translation services - requirements for translation services.

[Kar11] Karusinen et al. (2011). Trade-off between automated and manual software testing. International journal of system assurance engineering and management.

[Kub22] Kubernetes. Production-grade container orchestration (2022). [visited 2022-11-23]. Access through the Intenet: <https://kubernetes.io/>

[Lah22] Lahsen, N. (2022). Open API 3 and Spring boot. [visited 2022-11-21]. Access through the Internet: <https://springdoc.org/>

[LBU13] Lommel, A., R., Burchardt, A., Uszkoreit, H. (2013). Multidimensional Quality Metrics: A Flexible System for Assessing Translation Quality.

[Lin22] Lingoport, inc. LocalyzerQA: linguistic review made easy [visited 2022-06-09]. Access through the Internet: <https://wiki.lingoport.com/LocalyzerQA>

[Lin23] Lingoport, inc. LocalyzerQA: a new approach to improving translation quality and speed [visited 2023-03-10]. Access through the Internet: <https://lingoport.com/localyzerqa-linguistic-qa/>

[Loc21] Localise, inc. Localization testing: what is it and how to do it. (2021). [visited 2023-03-17]. Access through the Internet: <https://lokalise.com/blog/localization-testing/>

[Loc22] Localise, inc. Localise: localization workflow management [visited 2022-06-04]. Access through the Internet: <https://lokalise.com/product/localization-workflow-management>

[Loc23a] LocalePlanet. ICU Locale “Lithuanian (Lithuania)” (lt_LT) (2023). [visited 2023-01-02]. Access through the Internet: <https://www.localeplanet.com/icu/lt-LT/index.html>

[Loc23b] Localise, inc. Localise: translation quality assurance [visited 2023-03-10]. Access through the Internet: <https://lokalise.com/product/translation-quality-assurance>

[MH15] Mukundan, S., Hedge, J. (2015). Software localization: some issues and challenges.

[Mie22] Mieke, (2022). Software localization quality assurance: what is it and why does it matter? [visited 2022-10-30]. Access through the Internet: <https://laoret.com/blog/software-localization-quality-assurance-what-is-it-and-why-does-it-matter/>

[OA13] Owens, D., Anderson, M. (2013). A generic framework for automated quality assurance of software models supporting languages of multiple paradigms. Department of Computing, Edge Hill University, Ormskirk, Lancashire. Academy publisher.

[OC10] Oracle Corporation. What is a locale? (2010). Oracle documentation. [visited 2022-05-13]. Access through the Internet: <https://docs.oracle.com/cd/E19253-01/817-2521/overview-39/index.html>

[OC22] Oracle Corporation, and/or its affiliates. Internationalization tutorial. Oracle documentation. [visited 2022-06-04]. Access through the Internet: <https://docs.oracle.com/javase/tutorial/i18n/intro/index.html>

[Ora13] Oracle corporation. Introduction to Java technology (2013). [visited 2022-11-21]. Access through the Internet: <https://www.oracle.com/java/technologies/introduction-to-java.html>

[PGD14] Plasseraud, S., Graber, A., Deguerry, G. (2014). The true cost of localization QA: an empirical study. Proceedings of the 17th Annual Conference of the European Association for Machine Translation (EAMT).

[Phr23] Phrase. Localization management: what it is, and why it matters for global worth. (2023). [visited 2023-03-20]. Access through the Internet: <https://phrase.com/blog/posts/localization-management/>

[RH20] Red Hat. What is a REST API? (2020). [visited 2022-11-26]. Access through the Internet: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

[Sch02] Schäler, R. (2002) The Cultural Dimensions in Software localization. Localisation Focus, vol. 1, isssue 2.

[Sin17] Singh et al. (2017). Fault localization in software testing using soft computing approaches. 4th International Conference on Signal Processing, Computing and Control (ISPCC). Access through the Internet: <https://www.researchgate.net/publication/322713266_Fault_localization_in_software_testing_using_soft_computing_approaches>

[Twa18] Twardar, M., O., (2018). A brief history of localization. [visited 2022-05-10]. Access through the Internet: <https://www.translationroyale.com/history-of-it-localization/>

[Tom20] Tomedes. Translator's blog. What is localization testing? (2020) [visited 2022-10-24]. Access through the Internet: <https://www.tomedes.com/translator-hub/localization-testing>

[Tut22] Tutorials point. Spring boot – introduction (2022). [visited 2022-11-21]. Access through the Internet: <https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm#>

[UCLDR21] Unicode CLDR: CLDR 40 release note. [visited-2022-06-03]. Access through the Internet: <https://cldr.unicode.org/index/downloads/cldr-40>

[UCLDR22] Unicode CLDR: Unicode CLDR project. [visited 2022-05-29]. Access through the Internet: <https://cldr.unicode.org/>

[UO22] Unicode organization. Unicode data markup language: Part 2. [visited 2022-05-30]. Access through the Internet: <https://unicode-org.github.io/cldr/ldml/tr35-general.html#Contents>

[Woy01] Woyde, R. (2001). Introduction to SAE J2450 translation quality metric.

[Yam23] Yamagata. QA Distiller: find translation mistakes easy way (2023). [visited 2023-04-23]. Access through the Internet: <https://www.qa-distiller.com/en>

[Yij05] Yijun et al. (2005). Making XML document markup international.

Term definitions

Internationalization	Process of generalizing a product so that it can process (maintain) different languages and cultural attitudes without redesigning it.
Locale	Definition of the subset of a user's information technology environment that depends on language, territory, or other cultural customs.
Localization	Process of adapting a product or content to a specific location or market.
Localization issue (bug)	The issue in software cultural elements caused by adapting a product or content to a specific location or market.
Localization quality assurance	Process of ensuring that software can be used in different geographic areas this could mean considering language, local conventions etc.

Abbreviations

API	Application programming interface
CLDR	Common locale data repository
ICU	International component for Unicode
JVM	Java virtual machine
LISA	Localization industries standards association
LTS	Long-time support
MQM	Multidimensional quality metrics
MVC	Model viewer controller
NLP	Natural language processing
OS	Operating system
POSIX	Portable operating system interface
REST	Representational state transfer
SCMS	Source control management system
QA	Quality assurance
TMS	Translation management systems
TQS	Translation quality score
UI	User interface