TString: a tool to locate the target string's screen based on automatic exploration

Lais Felipe
Federal University of Pernambuco
Recife, Brazil
lpf2@cin.ufpe.br

Breno Miranda
Federal University of Pernambuco
Recife, Brazil
bafm@cin.ufpe.br

Maria Couto
Federal University of Pernambuco
Recife, Brazil
mrlc@cin.ufpe.br

ABSTRACT

Developing localized software applications for multiple languages can significantly expand a developer's potential consumer base, offering strong economic incentives. When targeting the global market, it is essential to adapt content to the specific region, culture, or country that the software will serve. Therefore, Localization (l10n) and Internationalization (i18n) testing are crucial for ensuring a seamless user experience, regardless of the locale settings on the application. A key task in localization testing is validating textual content (strings), which often relies heavily on the tester's experience to locate screens containing the strings that need validation. However, the limited availability of automated tools for localization and internationalization testing results in labor-intensive, repetitive manual work. This research seeks to address these challenges by (i) introducing TString, a tool designed to assist localization and internationalization testers in identifying screens containing specific strings for validation, and (ii) conducting a preliminary evaluation to assess the usefulness of TString in an industrial context. To assess the tool's usability, the System Usability Scale (SUS) was employed. A group of localization/internationalization testers received an overview of the tool and subsequently completed the SUS questionnaire. The tool achieved highly positive results, with scores ranging from 92.5 to 100, indicating strong acceptance and a high likelihood of meeting the testers' needs.

CCS CONCEPTS

• **Software and its engineering → Software testing and automation**.

KEYWORDS

Localization, Internationalization, `L10n`, `i18n`, string, tool

1 INTRODUCTION

When it comes to developing software for the worldwide market, it is important to keep in mind the concept of Globalization (G11n). G11n refers to the process of developing software that provides a seamless user experience regardless of the language or region in which it's being used. The process of G11n comprehends two main concepts: Internationalization and Localization. Internationalization refers to developing software in such a way that it can support different languages and regions [15]. Localization is to effectively localize software according to the grammar rules of the languages that are supported while taking into account the cultural and local aspects of each region [16].

Localization testing mostly validates, from the user's perspective, strings shown on screen. In `i18n` and `L10n`, the term "locale" is commonly used instead of language. Languages spoken in different nations may have unique expressions and meanings. For instance, en-US refers to English from the United States, while en-GB represents English from the United Kingdom. In a food app for example, the word "eggplant" should appear when the locale is set to en-US, whereas "aubergine" should appear when it is set to en-GB. Therefore, is essential to validate the content for each locale supported even though the it's the same language.

The first version of an app supports one specific locale, known as source, and subsequently localized for the others. When strings are edited or new strings are added to the app, the tester should validate them for each affected locale. It means that if the source changes in any manner, all locales must be validated to ensure that the change did not cause any issue in other locales. In some circumstances, a string can be altered for a locale that is not the source, therefore validation is only conducted on that locale.

For some apps used around the world that support a large number of locales, `L10n` testing becomes extremely time-consuming and repetitive. If the software is large and supports 52 locales, for example, the tester must explore the app, which is already an extensive work, to locate and check each string. It should be done for all 52 locales supported to ensure that no issues arise in any of them.

Validating newly added or recently modified strings is a time-consuming and repetitive activity that can be automated, however, there is no much tool support for this type of `L10n` testing. To have a tool that could help to find newly added or recently modified strings could speed up this activity and reduce the burden of manual exploration faced by the testers. Motivated by this challenge, the main contributions of this work are:

- introducing TString, a tool to assist localization and internationalization testers in the task of identifying screens containing specific strings for validation;
- a preliminary evaluation for assessing the usefulness of TString in an industrial context.

This paper is organized as follows. Section 2 presents the context and motivation of this research and common `i18n`/`L10n` issues. Section 3 shows the background. Section 4 describes the tool developed (TString). Then Section 5 defines the evaluation, results and discussion, and threats to validity. Section 6 discusses related work, and Section 7 presents the conclusions and future work.

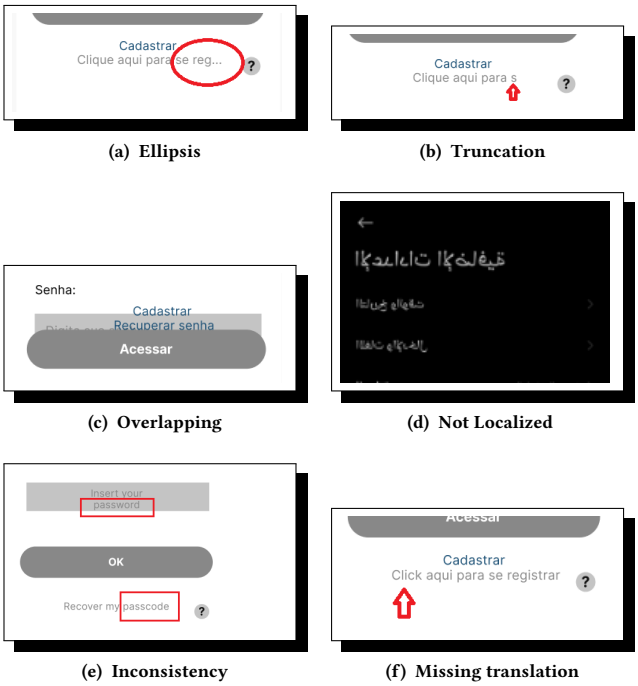
2 CONTEXT AND MOTIVATION

2.1 Common `L10n`/`i18n` issues

There are a lot of issues that can be found on `i18n` and `L10n` testing process. In this section it's going to be presented six common

issues which are: Ellipsis, truncation, overlapping, not localized, inconsistency and missing translation.

- **Ellipsis**, represented by three dots in Figure 1a, are used to reduce the length of a string when it exceeds the screen space limit. Shortening a string with ellipsis might result in unpleasant or derogatory words or expressions, negatively impacting the user experience.
- **Truncation**, seen on Figure 1b happens when the string does not fit on the screen space, but instead of presenting ellipsis, the string is cropped at any part of it which again can result in unpleasant or derogatory words or expressions.
- **Overlapping** happens when visual elements of the UI overlaps each other as seen o Figure 1c.
- **Not localized** is when even though the texts are translated, however the content is not culturally or regionally appropriate, so it's not considered localized. For example the correct currency, or following Right to Left (RTL) languages patterns. RTL languages are languages that written and read from the right side to the left. This way, the elements will be aligned on the right side of the screen, however a issue is when its not as seen on Figure 1d.
- **Inconsistency** is when the same term or word is translated with different words that have the same meaning as seen on Figure 1e.
- **Missing translation** happens when for some reason there are strings on screen that do not follow the locale set and instead presents strings or part of strings that were not translated and still follows the source. Example in Figure 1f.



(a) Ellipsis (b) Truncation

(c) Overlapping (d) Not Localized

(e) Inconsistency (f) Missing translation

Figure 1: Common i18n/L10n issues

2.2 String validation

Among the validation activities carried out by L10n/i18n testers is to validate newly added strings or modified strings. These activities can be triggered for a few reasons. Adding a new string is necessary when a feature is implemented or when a screen is added or modified. A modification to the string is made, for example, when a localization issue is found, and its value needs to be changed to make the string appropriate. In both cases, the strings needs to be validated to make sure that it did not cause a L10n issue.

For instance, consider Figure 2, which contains screens in the Korean locale from an Android app. The navigation flow (order of the screens during app usage) follows from the leftmost screen to the rightmost one. Suppose a tester needs to validate the string *마치다* (on the third screen). To reach that screen, the tester will need prior knowledge of the app's features and the actions that trigger the screen containing the string to be validated. Typically, in L10n testing, the tester does not have proficiency in all locales supported by the software [1]. Therefore, to begin testing, the tester might first need to explore the app in a locale they are comfortable with and also know the value of the string *마치다* in that locale. Once the tester locates the string in the app, they can set the app locale to Korean and validate it in the UI. In this example, we have only three screens; however, the more extensive the app, the more time and effort will be required to complete the validation.

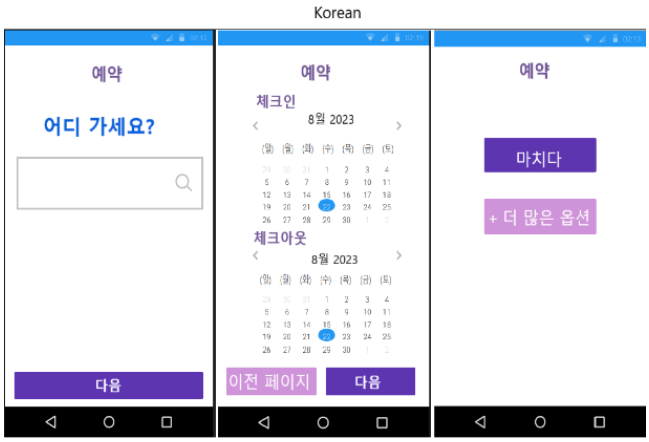


Figure 2: L10n testing example.

Couto and Miranda [7] highlight the importance of the L10n testing as the issues can be easily detected by the end-users. Hence it's notable the great scope that L10n testing has and its importance. It is also essential to highlight that L10n testing is much more than testing strings, it covers cultural aspects, UI elements in general, and also functionalities. However, according to Ramler and Hoschek [13], there is a shortage of automated tool support for localization testing.

Therefore, L10n testing would certainly benefit from the development of a tool that could assist in the process of finding the precise screens where the target strings needs to be validated. Such a tool would ease the workload for testers while simultaneously increasing the effectiveness of the string finding process.

3 BACKGROUND

3.1 L10n/i18n testing

Localization and Internationalization testing entails validating textual content to make sure that problems similar to those illustrated in Section 2.1 do not occur in the software. Additionally, L10n/i18n testing also ensures that contents such as currency, date, time, units of measurement, and UI elements alignment are adhering to the locales requirements.

According to Rajkuma [12], L10n testing is divided into the 5 phases illustrated in Figure 3:

- **Pre-localization test**: Assurance that the testers are going to have all the necessary documentation before starts.
- **Regional test**: Names, colors and etc. must be in accordance with the specific standard of each region/country.
- **Language test**: Grammar, punctuation and other aspects are adequate.
- **Appearance and Layout test**: Check the interface layout. (e.g., RTL).
- **Functionality test**: Verify if the localized application functions correctly.

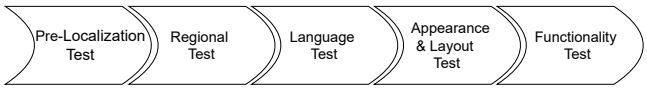


Figure 3: Localization Testing (adapted from [12]).

The fourth phase (Appearance and Layout test) shows that the scope of L10n testing extends beyond textual content validation to include the orientation of UI components on screen. For instance, some languages are written and read from the left to the right (LTR) and some are written and read from the right to the left (RTL). As depicted in the Figure 4, the difference in reading direction is apparent in the Arabic and English languages. Arabic, being an RTL language, requires that elements are positioned differently from English, which is an LTR language. L10n/i18n testing must consider and accommodate such differences to ensure that the UI is also appropriately adapted to cater to the needs of the target audience.

It is also important to ensure that regional aspects, such as date format, are in accordance with the needs of the target audience. The Unicode common Locale Data Repository (CLDR)[1] , is used to know the correct format of dates, numbers, currencies, times, and time zones from different locales [4]. For example, for the locale pt_BR (Brazil > Portuguese), the correct format of date is dd/mm/yyyy. It means that the date January 2, 2024 should be presented as 02/01/2024 for pt_BR locale. However the same date for ko_KR (South Korea > Korean) should be presented as 2024.01.02.

Localization/Internationalization testing is essential to ensuring that software can properly meet the diverse needs of audiences, regardless of their locale. This entails not just verifying the textual content but also taking other crucial factors into account, like the UI elements and regional aspects.

[1]https://cldr.unicode.org/index

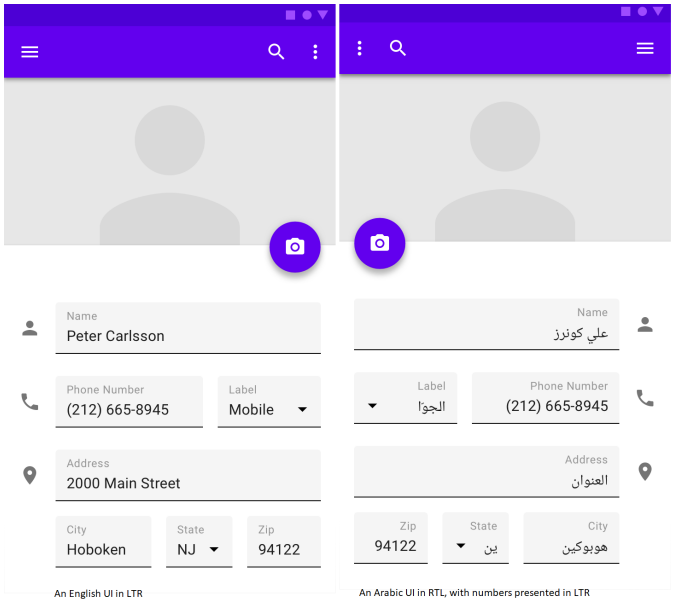


Figure 4: Example of RTL [3].

3.2 DroidBot

DroidBot [10] is as a lightweight UI-guided test input generator that can interact with Android applications. It is an open source tool that provides to the users autonomy to use custom scripts along with the exploration. DroidBot performs a combination of random and systematic techniques to explore the app's functionality. Whilst exploring Android applications, DroidBot captures screenshots and takes dumps of the app's screen contents. The files produced by the DroidBot exploration are used as input for our tool (TString) described next.

4 TSTRING

TString is a tool that has been developed aiming to simplify the string validation process. It's a tool that uses DroidBot outputs, such as screen captures and a JavaScript file containing information about the screens (activity, package, content, etc.). The tool accepts as input, for search, a text or a list of strings.

4.1 Droidbot exploration Workflow

The files from DroidBot are the result of an already performed execution of the tool. This execution is a sub process, from TString workflow, that is named DroidBot Exploration. The exploration is initiated when a new Android Application Pack (apk) is detected on the repository. The apk will be downloaded and DroidBot will explore that app version, generating the files that are used as an input for TString. The overview of this process is presented in Figure 5

4.2 TString Workflow

The first step in the TString process requires the user to input either a string content or a directory to a file that contains a list

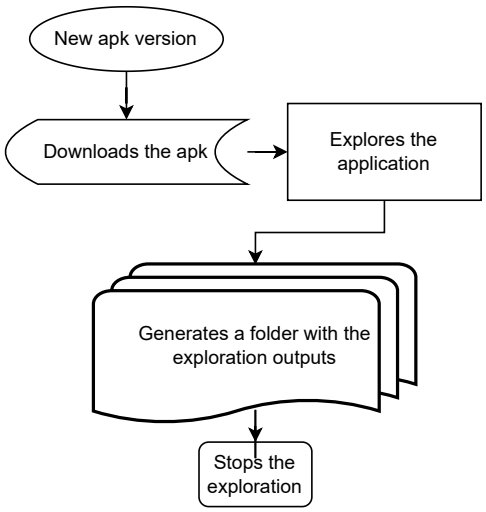


Figure 5: DroidBot Workflow.

of string contents. Once the input is provided, the tool accesses the reports generated by DroidBot, converts the report (utg.js) into two JSON files (nodes and edges) and searches for the content provided by the user. Then, the software generates an HTML file, named TString.html, which contains the results of the search. The is illustrated in Figure 6, providing a visual representation of the tool's workflow.

4.3 Walk-through

Figure 7 illustrates the main screen of TString. The first step required for using TString involves entering the string that the user wants to search for in the designated text box. Alternatively, the user can opt for a directory (path) that points to a file that contains a list of strings. The interface offers two options to choose from: "Search for one string" or "Search for more than one string", providing users with the flexibility to select the most appropriate search method for their needs.

Once the user has decided on the search method they wish to use, the next step is to select the application, the app's version and also the locale they want to employ for the search. Then the user will click on "Confirm" to perform the search. It is essential to note that the search can only be executed on versions that have been previously covered by the DroidBot exploration sub process and are available on its repository. This is because the outputs generated during the exploration phase are indispensable for the execution of TString.

After executing the search, TString generates an HTML report that presents the results. The generated report provides a step-by-step guide, outlining the sequence of screens that must be navigated to locate the specific string content entered by the user. Figure 8 provides an example of TString report. In the first example, the tester needs to validate the string "Weitere Informationen" for the locale de_DE (German, Germany). The report indicates that three steps are required to reach this string. First, on the home screen, the tester clicks "Weiter" to proceed to the next screen. On the second screen, the tester clicks "Weiter" again, and finally, the

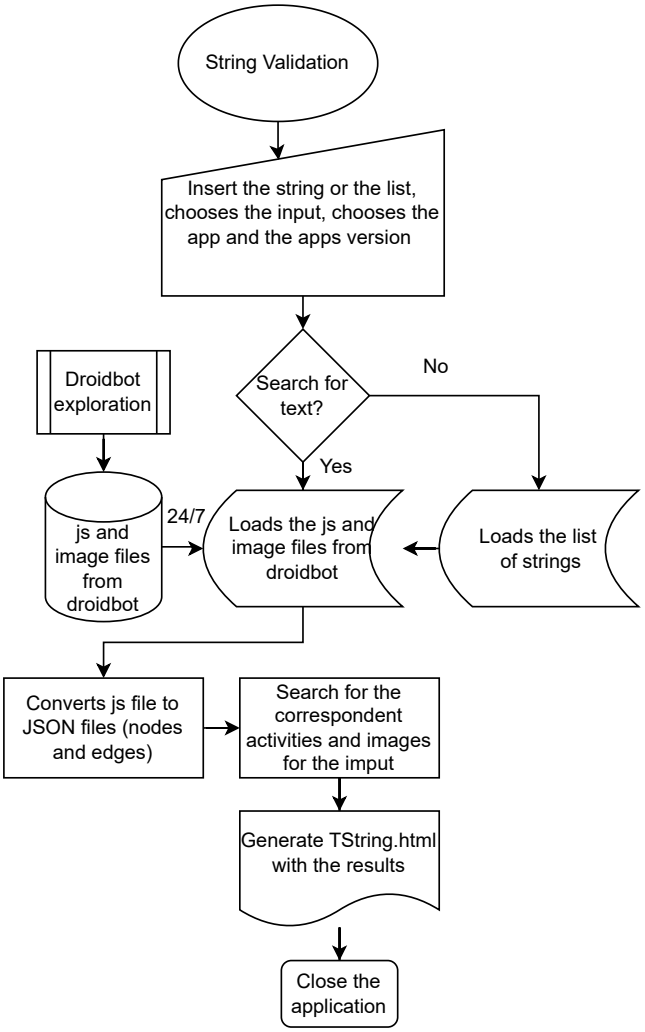


Figure 6: TString Workflow.

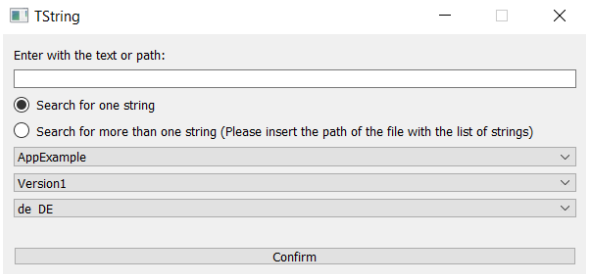


Figure 7: TString main screen

string is visible and can be validated on the third screen. The report includes screenshots of the app's screens, which assist the tester in quickly validating the content. In the second example, the tester must validate the string "Datenbank verschlüsseln". Starting from the main screen, the tester clicks "Weiter" to reach the next screen.

On the second screen, the tester needs to disable the toggle to reveal the target string, which is displayed on the third screen. In this example, the report highlights an issue: the target string is truncated. Based on this information, the tester could promptly identify the problem and submit a change request.

5 EVALUATION

To evaluate the effectiveness of TString in assisting localization and internationalization testers with identifying screens containing specific strings for validation, we utilized the System Usability Scale [5]. We recruited software testers from an industrial partner that performs localization and internationalization testing on mobile devices.

The testers were introduced to TString through a comprehensive demonstration that showcased the main interface, input options, and the tool's report generation capabilities within a real-world scenario. Following the demo, the testers were asked to complete the SUS questionnaire, providing their impressions and feedback on the proposed tool.

5.1 System Usability Scale (SUS)

System Usability Scale is a ten-item scale to evaluate usability. Brooke [5] has proposed to use 10 statements that cover a variety of aspects of system usability and the participants then rate their level of agreement with each statement based on a Likert scale scale ranging from 1 to 5, with 1 indicating strong disagreement and 5 indicating strong agreement. Brooke [5] proposed the following statements:

(1) I think that I would like to use this system frequently
(2) I found the system unnecessarily complex
(3) I thought the system was easy to use
(4) I think that I would need the support of a technical person to be able to use this system
(5) I found the various functions in this system were well integrated
(6) I thought there was too much inconsistency in this system
(7) I would imagine that most people would learn to use this system very quickly
(8) I found the system is very cumbersome to use
(9) I felt very confident using the system
(10) I needed to learn a lot of things before I could get going with this system

In our SUS questionnaire, the statements 1 to 8 were adopted as proposed by Brooke [5]. Statements 9 and 10 were slightly adjusted to better reflect the context in which the questionnaire was applied:

(9) I would feel very confident using the system
(10) I will need to learn a lot of things before I can get going with this system

After the participants have evaluated the statements above on the range level of agreement from 1 to 5, the SUS score is calculated as follows: each item (statement) score will range from 0 to 4. For each of the odd numbered item the score contribution is the scale position minus 1. For each of the even numbered item, the contribution is 5 minus the scale position. After that the scores are added and multiplied by 2.5 to obtain the overall value of SUS. The final SUS score will be on the range of 0 to 100 [5].

For interpreting the results of the SUS we adopt the acceptability score classification and grading scale from Bangor et al. [2] (Figure 9). For scores below 50, the usability of the evaluated system lies into the *not acceptable* range and could be classified as *worst imaginable* or *poor*. On the other hand, systems with scores from 70 and up lie in the *acceptable* range and their rating can vary across *good*, *excellent*, and *best imaginable*.

5.2 Participants

We recruited localization and internationalization testers from an industrial partner that collaborated with our study. The testing team at this company consists of five members, one of whom is the technical leader who is not a target user of our proposed tool. The four team members who are actively involved in `L10n/i18n` testing on a daily basis were invited to participate through a recruitment form. All four testers completed the form and voluntarily agreed to participate in our study.

Table 1 presents the demographic data of the participants. All of them have an educational background in IT, specializing in software testing. Regarding their experience in `L10n/i18n` testing, two participants are at the senior level, with three to four years of experience, while the other two are at the junior level, with one year of experience. The participants' ages ranged from 24 to 27 years.

5.3 Results and Discussion

After the participants finish their evaluation through the SUS questionnaire, the scores were calculated as explained in Section 5. The SUS scores for each participant are shown in Table 1. As the questionnaire was anonymous, each participant is referred as "P" + number.

The arithmetic average of the results was 96.8 putting TString in the acceptable range and classifying its usability somewhere between *excellent* and *best imaginable*. In addition to that, the individual scores ranged from 92.5 (*excellent*) to 100 (*best imaginable*).

Participant	Age	Exp. Level	Years in SW Testing	SU
P1	24	Senior	4	95.0
P2	26	Senior	3	100
P3	27	Junior	1	100
P4	24	Junior	1	92.5
System Usability average				96.8

Table 1: Participant's demographic information and SUS scores based on each survey answer

5.4 Threats to Validity

Throughout the course of this study, we encountered several challenges that threatened the external validity of our findings. One significant threat stems from the limited number of participants, as only four `L10n/i18n` testers completed our SUS questionnaire. It is important to note that in the industrial context where our study was conducted, the testing team responsible for `L10n/i18n` testing consists of five members. One of these members, the technical

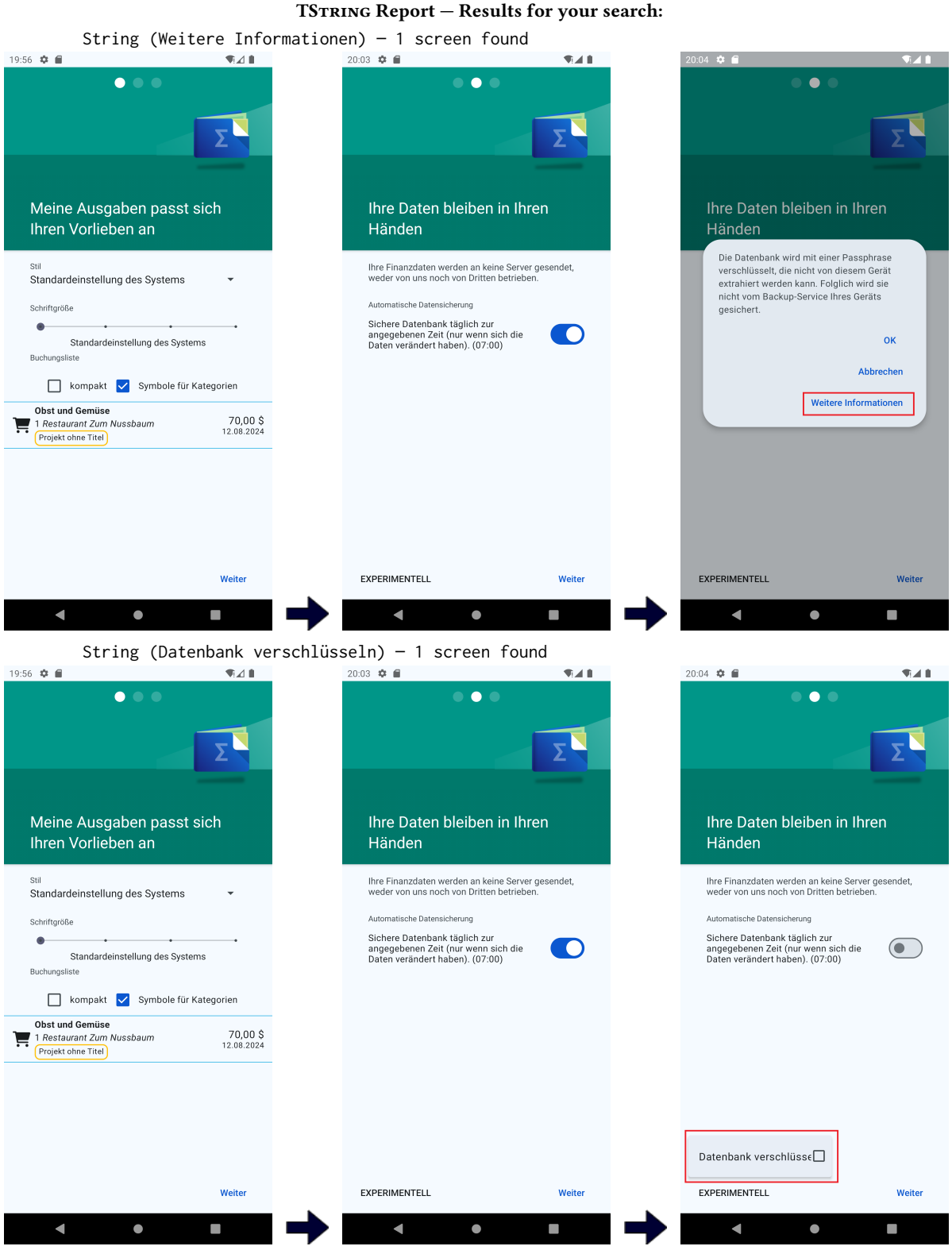


Figure 8: Example of a report generated by TString

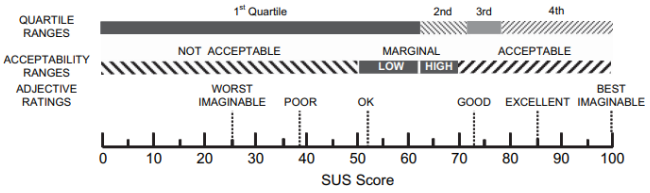


Figure 9: Acceptability score classifications and grading scale SUS according to [2]

leader, is not a daily user of our tool, and thus was not included as a target participant. Consequently, while the number of participants may seem small in absolute terms, it actually represents 80% of the entire testing team, or 100% of the potential tool users. To fully address this limitation, additional empirical studies, ideally conducted across different companies, will be necessary.

Moreover, the Hawthorne effect [11] presents another potential threat to validity. Given that participants were aware of their involvement in the study, their behavior might have been influenced, potentially leading to more favorable evaluations of our proposed tool. Such behavior changes due to observation may not accurately represent their usual usage patterns, which could skew the results and diminish the reliability of our findings. To mitigate this threat, we plan to conduct additional studies that include control groups to better isolate the effect of the tool itself.

Finally, while we acknowledge the valuable contributions and insights gained within the specific context of our investigation, we recognize that our conclusions cannot be generalized without further experimentation and validation in a broader set of circumstances.

6 RELATED WORK

6.1 Test Automation

Zhifang et al. [17] introduce the idea of testing a framework that utilizes a combination of model-based testing and user interface testing to automate the testing process on mobile devices. The authors point out the importance of testing automation to improve the quality of software on mobile devices.

Wang and Wu [14] proposes a framework that utilizes Appium, an open-source mobile automation tool, to automate the testing process on mobile applications. The framework is tested on a sample mobile application, and the results show the effectiveness and efficiency of the proposed method. It provides insights on the use of Appium for automation testing. The authors indicate that using a tool to automate the testing activity can make it be more convenient, be more efficient and reduce costs.

Ayyal Awwad and Slany[1] present an automated bidirectional localization testing approach for Android apps. The study discusses the importance of automated testing in order to support localization testing, given that developers and translators are usually different people. The study addresses specific challenges associated with BiDi-languages, particularly Arabic. The solution present differentiate itself by not only focusing in translation and adoption of locales, but a complete consideration for BiDi-language issues in general. The proposed approach effectively identify issues related to design, translation and cultural aspects based on the locale conventions.

6.2 L10n Testing

Leiva and Alabau [9] debate the importance of visual contextualization in user interface (UI) localization. A study was conducted in which they compared the effectiveness of UI localization with and without visual contextualization. The authors present four hypotheses: H1. Visual context improves localization quality; H2. When the UI is available, translations are finished later; H3. In-place localization leaves strings untranslated; and H4. Visual context improves translator's productivity. The results showed that visual contextualization significantly improved the localization quality and reduced user confusion.

Couto and Miranda [6, 7] highlight the challenges on training novices testers on `L10n/i18n` testing. They present a tool to assist this process. The purpose of the proposed tool is to replicate the failures discovered in i18n/l10n testing by seeding errors in the string files of mobile open source apps. The seeded application helps the novice testers can become familiar with the procedure and communicate with i18n/l10n failures while still in the training phase.

Velásquez et al.[8] discuss the challenges of internationalizing (i18n) mobile apps, specifically focusing on the impact of translating UI strings on Android apps' graphical user interfaces (GUIs). To translate apps manually for different languages, it's time-consuming and prone to errors, which can result in unexpected changes and bugs in the UI. The authors analyzed 31 Android apps and their translated versions to identify how translating English strings into seven different languages affects Android app GUIs. It was identified various i18n-related changes and bugs. A taxonomy of these changes and bugs is presented, and the authors discuss the implications for developers and researchers.

7 CONCLUSIONS AND FUTURE WORK

This paper introduces TString, a tool designed to assist localization and internationalization testers in identifying screens containing specific strings for validation. For the task of exploring an Android app while capturing the strings for each screen TString leverages DroidBot.

The primary goal of TString is to reduce the manual effort associated with `L10n/i18n` testing activities. We conducted a preliminary evaluation to assess the tool's usefulness in an industrial context. Four software testers who are actively engaged in `L10n/i18n` testing on a daily basis were invited to provide feedback on TString using the System Usability Scale (SUS). The results of our evaluation indicate a high level of acceptance among testers for using TString to support their daily validation tasks.

Looking ahead, we plan to enhance TString with the capability to automatically highlight potential `L10n/i18n` faults. This improvement would enable testers to prioritize the most critical strings and screens, further streamlining the validation process.

ACKNOWLEDGMENTS

This work was supported by the research cooperation project between Motorola Mobility Comércio de Produtos Eletrônicos Ltda

(a Lenovo Company) and CIn-UFPE, and by a grant from the National Council for Scientific and Technological Development (Grant CNPq-Universal 408651/2023-7).

REFERENCES

[1] Aiman M. Ayyal Awwad and Wolfgang Slany. 2016. Automated Bidirectional Languages Localization Testing for Android Apps with Rich GUI. *Mobile Information Systems* 2016 (2016), 13 pages.

[2] Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction* 24 (2008), 574–597. https://doi.org/10.1080/10447310802205776

[3] BIDIRECTIONALITY. [n. d.]. https://m2.material.io/design/usability/bidirectionality.html#mirroring-layout

[4] Francis Bond and Ryan Foster. 2013. Linking and extending an open multilingual wordnet. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1352–1362.

[5] John Brooke. 1995. SUS: A quick and dirty usability scale. *Usability Eval. Ind.* 189 (11 1995).

[6] Maria Couto and Breno Miranda. 2023. An Industrial Experience Report on the Challenges in Training Localization and Internationalization Testers. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing* (Campo Grande, Brazil) *(SAST '23)*. Association for Computing Machinery, New York, NY, USA, 3 pages. https://doi.org/10.1145/3624032.3624045

[7] Maria Couto and Breno Miranda. 2023. l10n-trainer: a Tool to Assist in the Training of Localization (l10n) and Internationalization (i18n) Testers. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 6 pages. https://doi.org/10.1145/3613372.3613420

[8] Camilo Escobar-Velásquez, Michael Osorio-Riaño, Juan Dominguez-Osorio, Maria Arevalo, and Mario Linares-Vásquez. 2020. An Empirical Study of i18n Collateral Changes and Bugs in GUIs of Android apps. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 581–592. https://doi.org/10.1109/ICSME46990.2020.00061

[9] Luis Leiva and Vicent Alabau. 2014. The impact of visual contextualization on UI localization. (04 2014). https://doi.org/10.1145/2556288.2556982

[10] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: a lightweight UI-Guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 23–26. https://doi.org/10.1109/ICSE-C.2017.8

[11] Rob McCarney, James Warner, Steve Iliffe, Robbert Van Haselen, Mark Griffin, and Peter Fisher. 2007. The Hawthorne Effect: a randomised, controlled trial. *BMC medical research methodology* 7 (2007), 1–8.

[12] Rajkumar. 2023. A complete beginners guide to localization testing. https://www.softwaretestingmaterial.com/localization-testing/

[13] Rudolf Ramler and Robert Hoschek. 2017. Process and Tool Support for Internacionalization and Localization Testing. *Product-Focused SW Process Improvement* (2017), 385–393.

[14] Junmei Wang and Jihong Wu. 2019. Research on Mobile Application Automation Testing Technology Based on Appium. In *2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*. 247–250. https://doi.org/10.1109/ICVRIS.2019.00068

[15] Xin Xia, David Lo, Feng Zhu, Xinyu Wang, and Bo Zhou. 2013. Software Internationalization and Localization: An Industrial Experience. *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, 222–231. https://doi.org/10.1109/ICECCS.2013.40

[16] Chunsheng Zhao, Zhiyong He, and Wei Zeng. 2010. Study on International Software Localization Testing. *2010 Second World Congress on Software Engineering* 2 (2010), 257–260.

[17] Liu Zhifang, Liu Bin, and Gao Xiaopeng. 2010. Test Automation on Mobile Device. In *Proceedings of the 5th Workshop on Automation of Software Test* (Cape Town, South Africa) *(AST '10)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/1808266.1808267