2016 IEEE International Conference on Software Maintenance and Evolution

An Empirical Study of Internationalization Failures in the Web

Abdulmajeed Alameer and William G.J. Halfond
Department of Computer Science
University of Southern California
Los Angeles, CA, USA 90089-0781
Email: {alameer, halfond}@usc.edu

***Abstract*—Web application internationalization frameworks allow businesses to more easily market and sell their products and services around the world. However, internationalization can lead to problems. Text expansion and contraction after translation may result in a distortion of the layout of the translated versions of a webpage, which can reduce their usability and aesthetics. In this paper, we investigate and report on the frequency and severity of different types of failures in webpages' user interfaces that are due to internationalization. In our study, we analyzed 449 real world internationalized webpages. Our results showed that internationalization failures occur frequently and they range significantly in terms of severity and impact on the web applications. These findings motivate and guide future work in this area.**

I. Introduction

Web application internationalization frameworks allow companies to more easily provide versions of their website customized for an end users' locale. This can improve the accessibility of a web site and improve the user experience. Although internationalization frameworks can simplify many localization related tasks, their use can come with unforeseen problems. Translated strings may expand or contract based on the verbosity of the target language. This can cause the intended appearance of a webpage to become distorted or its user interfaces (UIs) to become more difficult for end users to access. Avoiding these problems is essential for the success of a website as studies have shown that website visitors use a website's appearance to form opinions of its trustworthiness and the quality of the services it delivers [1], [2], [3].

This challenge related to internationalization is well-known in the web application developer community and there are many widely used internationalization frameworks [4]. These frameworks allow developers to isolate "need-to-translate" strings in locale specific resource files, which are loaded and used at runtime. A more lightweight option is for developers to integrate translation APIs, such as the Google Website Translator [5], into their websites. These APIs allow visitors to automatically translate text in the page. Although both of these mechanisms allow for the translation of websites, their use does not necessarily result in maintaining the intended appearance of a webpage. As we explain in more detail in Section II, translated versions of text can be of different lengths and heights, depending on the character set of the target language. Other problems, such as not properly setting character encoding information, can also cause the page to appear incorrect.

Despite awareness of the challenges of internationalization, little is known about the actual internationalization problems that plague deployed web applications. The result of this is that well-known blogs and software engineering companies often provide developers with generic and vague advice, such as "Provide for effective presentation of the UI after the expansion that results from translation" and "Avoid implicit assumptions or expressions of reading and writing directions" [6]. Although well meaning, this type of guidance does not provide directly actionable information or advice. Similarly, existing research in this area does not address the question of what type of internationalization problems occur. Instead, existing work has focused on techniques for exercising code in ways that could expose internationalization problems (e.g., Apple Xcode IDE pseudo-localization [7]) or techniques for identifying the symptoms of these problems (e.g., GWALI [8] and fighting-layout-bugs [9]), or techniques that locate strings in web applications that developers need to isolate in resource files for translation [10], [11]. Despite their contributions to detecting internationalization related problems, these techniques and their accompanying papers do not provide analysis or insight into the occurrence of these problems

To learn more about internationalization problems, we conducted an extensive empirical study of internationalization failures in web applications. Using automated detection and analysis techniques, we examined over 449 different web sites and found that a high percentage of them contained internationalization related problems. We analyzed these websites in more detail to understand the type of failures, their severity, and identify useful trends, such as problematic languages and internationalization frameworks. The results of our study were insightful. We identified common types of internationalization failures that were related to the configuration of a webpage and found a correlation between the prevalence of internationalization failures and specific languages and frameworks. Overall, these results can help developers to be aware of and avoid the most common types of internationalization failures.

The rest of this paper is organized as follows. In Section II, we provide additional background information about website internationalization. Then in Section III, we introduce and motivate our choice of research questions for the empirical

978-1-5090-3806-0/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSME.2016.55

study and explain the analyses we developed to address the different parts of the research questions. We present details about the subject applications in Section IV. Our results are in Section V. We discuss threats to validity in Section VI. Finally we contrast our contributions with related work in Section VII and summarize our findings in Section VIII.

II. Background

Many approaches can be used to build an internationalized web application. However, two general techniques have become widespread and popular. The first technique is to isolate the "need-to-translate" text, icons, and images into separate language-specific resource files. A user's web browser provides the user's preferred languages and a server side framework loads the correct language specific resource files and inserts them into placeholders in a webpage. This modularization of the language-specific content makes the management of the internationalized website easier. The second approach is to use online automated translation services (e.g., Google Website Translator [5]). With this approach, the developers install a plugin in their website. The website's visitors can select their desired language from a list of available languages and the plugin will scan the webpage to find all of the textual content. Then, the plugin will send that text to the online automated translation service, which will reply with a translation of the text to the desired language. The plugin then replaces the original text with the translated text. This approach allows the developers to easily make their web applications available in a large number of languages without the need to manually extract and translate the content. However, it is not guaranteed in either of these approaches that the translated text will not cause a layout failure, and it is up to the web developer to verify that the pages' user interfaces have not been distorted due to the text translation.

Various types of internationalization failures can occur on the web. The impact of these failures can lead to distortions in a webpage's appearance that can make its user interface less visually appealing, unreadable, or unusable. There are many potential problems that can lead to these failures, including: incorrect translation of the text in the webpage, not using the proper local conventions, such as measurement systems, and using culturally inappropriate images or colors. Although impactful, the solutions to these specific types of failures are not under the control of software engineers, so we do not focus on them in this study. Instead, we focus on two types of failures that are related to the layout of the webpage and the website's internationalization configuration.

The first type of internationalization failure that we focus on in this study is *Layout Failures (LFs)*, which are distortions of a webpage's appearance after its translation to a different language. Figure 1 shows a real world example of an LF taken from the Hotwire website. In the Mexican version of the page, the price text overflows its container. In general, the reason this type of problem occurs is because the size of the text varies significantly based on its language. The resulting change in text size causes the text to overflow its intended containing element or, if the containing element grows with the text, the container growth can cause other elements to move, changing the layout of the page. This change in the size of the text is mainly affected by three factors: the number of characters in the translated text, the language's characters' width, and the language's characters' height. Some of these changes can be rather dramatic. For example, in English the word "please" translates to "Se il vous plait" in French, which is almost three times longer. More generally, IBM internationalization guidelines state that an English text that is shorter than ten characters could increase in size from 100% to 200% when translated to another language [6]. Developers must anticipate this kind of expansion when they design their webpages. Additionally, the correct rendering of a page must be verified, as the fact that it renders properly for one language does not imply that it will render correctly for other languages.

(a) British version (b) Mexican version

Fig. 1: Part of a webpage and its localized version

The second type of internationalization failures that we focus on in this paper is a misconfiguration of an internationalization related property in the website or the web server. We refer to this type of failure as *Configuration Failures (CFs)*. Examples of these failures are not providing content language information in the HTML of the webpage and not specifying the correct encoding used in the webpage. In some situations, this type of CF could cause the browser to render the webpage incorrectly.

III. Research Questions and Protocol

In this section we describe in detail the types of questions and topics related to Internationalization Failures (IFs) that our study addresses. Broadly, our study considers the two previously discussed types of internalization failures, LFs and CFs. For each of these types we study three dimensions that can characterize their occurrence in web applications: frequency, type, and severity. Within each of these dimensions we further explore breakdowns by language and framework type. In the rest of this section, we more precisely define each of the three dimensions, introduce the analyses and metrics used for each, and then summarize the research questions addressed in our study.

A. Dimension: Frequency

Measuring the frequency of the different types of IFs helps us to address the basic question of how often the problem occurs in web applications. Here we are interested in determining the frequency of the problem with respect to the number of internationalized webpages. To gain further insight into the problem, we also break down the results by language and framework type. Reporting the results by language allows us to see if versions of webpages translated to certain languages are generally more problematic than others. For example, determining if translations that lead to text expansion are more problematic than translations that lead to text contraction. The breakdown by framework type gives us insight into whether pages generated using automatic translation framework are more or less likely to result in IFs than those designed to use server side mechanisms.

1) Detecting Layout Failures (LFs): To detect LFs, we ran a detection tool, GWALI [8], against the subject applications. GWALI allows web developers to automatically detect and localize LFs that are introduced in a webpage after translation. The inputs to GWALI are a baseline page, which is considered to represent the correct rendering of the website, and a translated version, which will be checked for LFs. If an LF is detected, then GWALI returns a ranked list of HTML elements that are most likely to have caused the detected failure.

GWALI detects LFs by building and then comparing Layout Graphs (LGs) for the two versions of the website. An LG models the relative layout of the elements in a webpage. The nodes of the LG represent the visible HTML elements in a webpage and an edge between two nodes represents the visual relationships (e.g., intersects, aligns, contains) that the elements have with each other. GWALI compares the LGs of the two versions of the webpage and identifies edges whose annotations have changed, which signifies that their relative positioning has changed due to the translated text. Heuristics are then applied to the reported edges to identify the HTML elements most likely to be responsible for the detected failure. A previous evaluation of GWALI showed that it was very accurate — detecting LFs with 91% precision and 100% recall. GWALI only needs 9.75 seconds to compare two webpages, which makes it fast enough for use on the large number of webpages used in our study.

As we described in Section II, a properly internationalized website maintains the same general structure of its webpages after translation. Only the language-specific content is replaced. If this assumption is violated, then GWALI will report all of the structural changes as LFs. To avoid this situation, we do not detect a webpage as containing an LF if the two pages have a different structure (i.e., the translated version contains more structural changes than those introduced by translated text.) As with the other analyses, the size of the subject pool required us to automate the identification of such pages. To do this we adapted a webpage matching algorithm defined by the WebDiff Cross-Browser Testing (XBT) technique [12]. This algorithm matches elements probabilistically using the elements' attributes, tag names, and Levenshtein distance between XPath IDs. For the purpose of checking whether the structure of the two versions of the page is similar, we computed the matching ratio (number of matched elements in the baseline divided by the total number of elements). Through observation, we determined that if the matching ratio was higher than 95%, then the two webpages did not have any significant structural differences that would preclude the usage of GWALI.

2) Detecting Configuration Failures (CFs): To detect CFs, we ran the World Wide Web Consortium (W3C)'s i18n Checker against each of the subject applications. The i18n Checker is the W3C's official internationalization validation tool [13], [14] and is used by web developers to verify that their webpages conform to certain internationalization standards. Note that the i18n Checker only verifies a webpage's syntactic compliance and cannot verify any properties related to the page's layout or appearance.

As input, the checker retrieves the webpage pointed to by a URL, parses the page and response headers, and then checks for compliance with a set of internationalization standards. The types of standards checked include proper specification of character encodings, language declarations, and text direction. As output, the checker reports to the web developer two different levels of violations: errors and warnings, along with suggestions on how to fix them.

B. Dimension: Severity

When an IF occurs, we are also interested in understanding its severity (i.e., impact) on the web application. In general, an IF can lead to a distortion of the appearance or functionality of a website, which can negatively impact a site's aesthetics, usability, and correctness. However, all IFs may not have the same impact on end users. IFs with low severity may not be noticed by a website visitor, and IFs with high severity will have a more significant impact on the website and are more likely to negatively impact an end user's impressions of the website. Since the impacts of LFs and CFs are very different, below we individually discuss how we measured the severity of the different types of IFs.

1) Severity Calculation for Layout Failures (LFs): To calculate the severity of LFs we calculated the impact of the LFs on the UI of the subject application. Note that the detection of an LF implies that there has been some sort of change in the appearance of the subject's UI. Therefore, we define metrics to quantify the magnitude of this change. To do this, we utilized the LGs, which model the layout of the webpages. An LG has nodes that represent the visible elements in the page and edges that are annotated with the visual relationships between the elements. The annotation on the edges captures three types of visual relationships, direction (i.e., North, South, East, West), alignment (i.e., top, bottom, left, right), and containment (i.e., contains and intersects). More details about the Layout Graph are described in our previous work [8]. We use the relationships annotating the edges in the LGs to calculate the following metric. For a pair of LGs, we calculate the number

of changes in the set of visual relationships annotating the edges in each graph and divide this number by the total number of visual relationships in the LGs (i.e., the metric is normalized by the page's size and layout.) This metric reflects the amount of distortion between the two UIs of the webpage.

2) Severity Calculation for Configuration Failures (CFs): To calculate the severity of CFs, we leveraged the severity ratings provided by the i18n Checker. For each web site, the checker generates a report containing two levels of CF, warnings and errors. An error implies that the behavior of a browser that is compliant with the specifications of the type of webpage (HTML4, HTML5, or XHTML 1.x) is undefined, and the document may not be rendered correctly by the browser. An example of this is that the text may be shown as replacement characters (e.g., a white question mark inside a black diamond). A warning is a less serious IF and implies that some less significant functionalities may not work properly, such as spell-checking or accessibility text-to-speech features.

C. Dimension: Type

This dimension defines a broad categorization within each of the two types of IFs. This categorization helps us to understand the common patterns of LFs and CFs in terms of their underlying problem or impact on the appearance of the webpage. The categorizations can also help to guide developers and researchers in the prioritization of the development of new techniques for preventing or repairing IFs.

1) Types of Layout Failures (LFs): We classify LFs by the type of change between the baseline and translated version. To identify the type of change we compare corresponding edges in the LGs of the two webpages for which GWALI detected an LF. For each pair of edges we categorize the change based on the set of annotations associated with an edge. For example, an edge may originally have "West" associated with it and in the translated version the "West" relationship is changed to "North." From this information we could infer that the translation caused two elements to move relative to each other. More broadly, we could classify any change from one directional annotation (e.g., "North" or "South") to a different directional annotation as an LF that caused an element movement. Below we list the different types of changes and how we used the annotation changes to infer them.

Movement: The annotations for an edge change from direction x to direction y. By "change," we mean that the annotation of direction x is removed and another direction y appears in the translated version. If direction x is still annotating the edge after the translation, we do not categorize the failure as a movement failure.

Overflow: The annotations for an edge change from "contains" to "intersects." This change means that an HTML element was containing another element or text, and in the translated page the contained element or text overflowed the container due to the expansion caused by the translation.

Alignment: This change must satisfy two conditions: (1) The movement condition does not apply to the edge, and (2) an alignment annotation (e.g., "Top Aligned" or "Right Aligned") appears in the baseline and disappears or changes in the translated version. This change means that two elements in the page that were aligned with each other are still in their relative positioning in the translated version, but they are no longer aligned with each other.

2) Types of Configuration Failures (CFs): We define types of CFs based on the descriptive message returned by the i18n Checker for each error or warning identified in a subject app. The descriptive message generally returns three pieces of information, the severity (i.e., error or warning), a text string denoting a general type of problem (e.g., missing language tag), and possibly detailed location information for where in the page the CF was identified. We use the second piece of information as the CF type. From our investigation, we found that there were 39 different types of problems that could be reported by the checker. Due to space constraints, we do not list them out, but discuss the most common ones in detail in Section V-B.

D. Summary of Research Questions

We created the research questions for our study based on the three different dimensions frequency, severity, and type. Basically, our set of research questions is the Cartesian product of the two types of IFs with the three dimensions explained above. For each of these research questions we provide aggregate results and then break down the results by language and type of framework. Table I shows a summary of our research questions.

TABLE I: Summary of the Research Questions

#	Research Question
1	What is the frequency of Layout Failures in web applications?
2	What is the level of severity of Layout Failures when they occur?
3	What are the most common types of Layout Failures?
4	What is the frequency of Configuration Failures in web applications?
5	What is the level of severity of Configuration Failures when they occur?
6	What are the most common types of Configuration Failures?

IV. Subject Applications

In this section we describe the web applications that we used to perform our study. To build a pool of subject applications we used an iterative process to identify applications that were appropriate for the evaluation. We began by using URouLette (www.URouLette.com) to identify a set of unique and random URLs. The initial set contained 9,406 unique URLs. We then analyzed the page pointed to by each URL to determine if the website was internationalized (see Section IV-A). If the analysis detected that the website was internationalized, we added it to the subject pool. We repeated this process until we

had a subject pool of 449 web sites. In Figure 2, we show a distribution of our subject applications by the category of the website. URouLette does not provide us with any information, beyond a URL, about a website. Therefore, we manually examined each of the subject applications and classified the webpages based on their intended usage. The subjects ranged from simple pages, having only 17 HTML elements, to large and complex pages, having 6,661 HTML elements.

For each application in the subject pool we downloaded a baseline language version and a set of translated versions. We attempted to download translated versions in each of the twelve most commonly used languages on the web: English, Russian, German, Japanese, Spanish, French, Portuguese, Italian, Chinese, Polish, Turkish, and Dutch. (See Section IV-C for additional discussion on language detection.) Some of the subjects were available in all of the twelve languages, while others were available in only a subset of these twelve languages. The total number of webpage pairs (i.e., a baseline page and a translated page) was 2,241. For each version of a subject, we saved a local copy of the webpage along with all of the embedded images and style sheets required for the page to render correctly. This allowed us to repeatedly run our analyses offline without dealing with network latency. We used the Firefox plugin, Scrapbook X [15], to save the pages. We modified the plugin, so we could automate the process of saving the pages using the Selenium WebDriver framework [16]. We loaded each page using Selenium WebDriver, allowed it to completely render, then disabled the JavaScript running on it using Scrapbook X. We disabled JavaScript because some pages used it to dynamically change content (e.g., rotating main news items) in the Document Object Model (DOM). This made it difficult to compare the two versions of a page using GWALI, unless their change timings were synchronized. The resulting saved page appears exactly as it would be seen by an end user on an initial visit to the page.

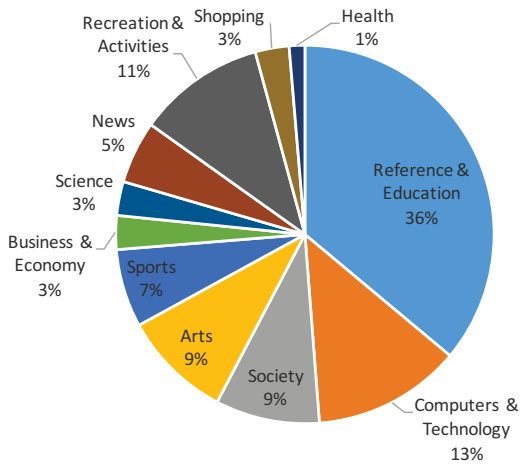

Fig. 2: Categorization of the subject applications

A. Detecting Internationalization

Our study only considered web applications that were internationalized. Due to the large number of subjects in our study, we designed automated heuristics to identify these web applications based on the URL provided by URouLette. For each URL, we retrieved the webpage via HTTP. Our first heuristic was to analyze the HTML and JavaScript content of the webpage to determine if they contained indicators of using an internationalization framework. (More details on the framework detection are provided in Section IV-B.) If a framework was detected, then we considered the page to be internationalized. Otherwise, our second heuristic was to request the page in different languages and check the language of the response. We did this by initially requesting the webpage without setting the HTTP "Accept-Language" header. Then we checked the language of the webpage returned in the response and marked it as the baseline language. After that, we sent multiple HTTP requests for the webpage with the "Accept-Language" header in each request set to a different language. If the page in one of the responses had a language different than the baseline, then we considered the subject to be internationalized. (More details on how we performed the language detection are provided in Section IV-C.)

B. Framework Detection

In Section V, we report the results broken down by translation framework type. The large number of subjects in our study necessitated that we automate the framework detection. To do this, we built a detector for internationalization frameworks. Our detector examined the HTML and JavaScript of the webpage to identify signatures belonging to different translation frameworks. For example, calls to certain JavaScript APIs are only used by a particular framework, while another one inserts HTML tags with ID attributes set using a specific pattern. For the frameworks that use automatic translation frameworks, it was particularly easy to identify the specific framework used, since a large amount of the code was exposed to the client (e.g., to allow them to select a drop down box.) For web sites that used server-side resource files, it was not possible to identify the framework with any specificity except for the fact that the translation was performed on the server side. Therefore, we used the framework detection analysis to determine whether the translation framework was implemented on the client-side (e.g., automatic translation framework) or on the server-side (e.g., resource files.)

C. Language Detection

In Section V, we also break down the results by language. Naively, one might assume that the language of the page is known because it matches the requested translated page. However, there were two problems that precluded the use of this simple heuristic. First, developers may only choose to support a subset of languages with resource files or via a translation drop down box. They may choose to only support certain languages based on business reasons or because they have only tested for correctness in certain key languages. In

this case, the translation frameworks generally return the page in the default language of the website if they do not support the requested language. Second, we found that many websites do not properly set the HTTP content-language header or the HTML "lang" attribute. As with framework detection, due to the high number of subjects in our experiments, identifying the returned language necessitated using an automated analysis.

For automated language detection, we used the popular language detection library, Compact Language Detector 2 (CLD2) [17]. The input to CLD2 is the URL of the website and the contents of the webpage. The library then uses a Naive Bayesian classifier to detect the language of the text in the page. If the webpage has multiple languages, CLD2 returns the top three languages used in the webpage. Our subject applications generally only had one language, which simplified the detection problem. However, if more than one language was identified, we selected the top language (i.e., the most frequently used language in the page) and considered it as the language of the webpage. An evaluation of CLD2 performed by its developers showed that it could correctly detect the languages of complex multi-language subjects with 99% precision and 98% recall [17].

V. Results of Study

In this section we report on the results obtained for each of the research questions. We first discuss the frequency, severity, and type with regards to LFs and then do the same for CFs.

A. Results for Layout Failures (LFs)

To address RQs 1–3, we ran GWALI on the subject applications. As we described earlier in Section III-A1, we excluded all of the pairs (i.e., baseline and translated version) that did not have a matching structure. This left us with 1,020 pairs having a baseline version with similar structure to the translated version. These pairs represented 139 different subject applications. For these pairs, we calculated the frequency of layout failures, analyzed the layout graph to get the severity, and categorized the failures that appeared in them.

1) Frequency: To answer RQ1, we calculated frequency in two ways. First, we calculated frequency across all of the tested pairs (i.e., baseline and translated version), and reported the number and percentage of pairs that had at least one IF. Second, we calculated the results on a subject application basis. To do this, we grouped the results of the pairs that belonged to the same subject application and reported if at least one of the pairs belonging to the subset contained a layout failure. After calculating both frequency numbers, we broke down the results by language and framework.

Of the 1,020 translated pairs, 787 (77%) contained at least one LF detected by GWALI. Reporting the numbers at the subject application level shows that 121 subjects (87%) had at least one LF. This indicates how common LFs occur in real world internationalized web applications.

When breaking the numbers down by baseline language (Figure 3), we found that Japanese, German, and French had the highest rate of failing (average of 99%), while Dutch had the lowest rate (61%). When breaking the results down by target language (Figure 4), we could not find any trends, and the rate of failure was high across all of the languages, ranging between 69% and 86%.

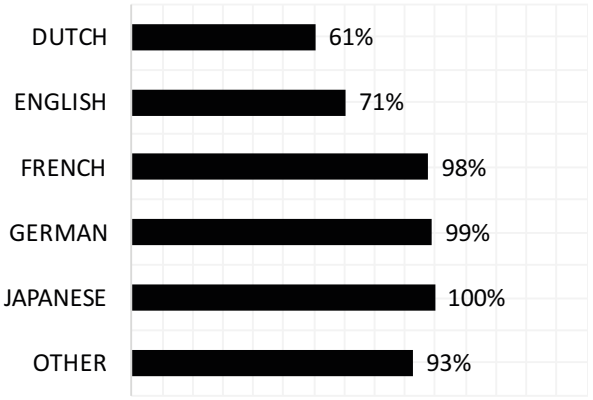

Fig. 3: Layout Failure frequency per baseline language

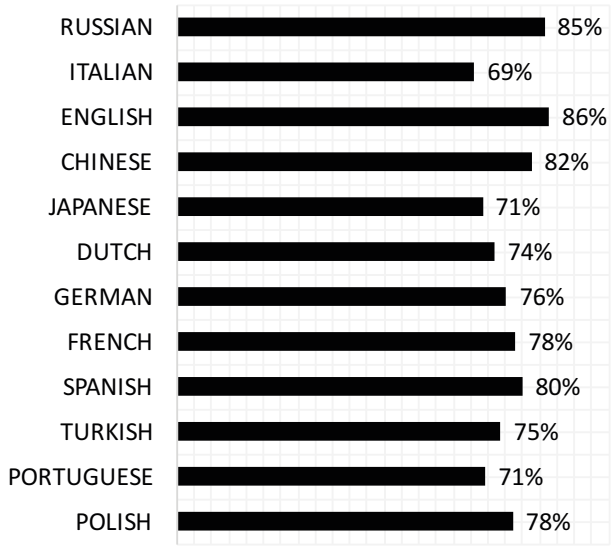

Fig. 4: Layout Failure frequency per target language

When breaking down the numbers by the type of framework, we found that 90% of the pairs using an automatic translation framework had LFs, while only 57% of the pairs using a server side resource file based framework had LFs. Overall, these results show a clearly higher frequency of LFs is associated with the automatic translation frameworks.

2) Severity: To answer RQ2, we computed, for all the test pairs, the severity metric as described in Section III-B (i.e., number of changes in the visual relationships in the LGs divided by the total number of visual relationships in the two LGs).

First, we computed the average severity ratio for all of the pairs of pages where an LF was detected. Then we broke down the results by language and by type of framework.

On average, for the failing pairs, there were 14 visual relationship changes for every thousand visual relationships in the LGs. We broke the results down by baseline language and found that the average severity was highest for French (40.1) and then Japanese (28.4). This result was the only statistically significant difference among the languages (Scott-Knott test [18] with a p-value < 0.05). The differences for the remaining languages ranged from 5.7 to 14.6 but did not have any statistically significant differences. When breaking the results by target language, we found no statistically significant differences for any language.

When breaking down the results by the type of framework, we found a slight increase in the severity of the LFs produced with server side frameworks. However, our analysis of this increase showed that it was not statistically significant (Mann-Whitney U with probability of .05). This is an indicator that the type of framework is not likely to have a correlation with the severity of the LFs.

3) Types: To answer RQ3, we recorded the types of changes detected by GWALI for each pair that contained an LF. Then we computed, for each type of LF, the percentage of tested pairs that had that type of change reported. After reporting the overall numbers, we broke down the results by language and type of framework. Overall, we found that the most common type of change was movement. It occurred in 69% of the LFs, compared to overflowing which occurred in 44%, and alignment which occurred in 47%.

When breaking down the results by target language, we found that movement layout failures occurred most frequently when the target language was Russian (85%) compared to the other languages (ranged from 60% to 76%). For the other types of changes we did not find any trend across the target languages. When breaking down the results by baseline languages, we found that translating from Polish and Japanese led to an overflow the most frequently (100% and 96% of the pairs for these languages), while English and German led to an overflow the least frequently (37% and 45%). Also, we found that alignment failures occur most frequently for pages translated from the Japanese language (in 92% of the failing pairs where the baseline language is Japanese).

When breaking down our results by framework, we did not find any trend. Automated translation frameworks and server side frameworks had similar distributions for the types of failures.

B. Results for Configuration Failures (CFs)

To address RQs 4–6, we ran the i18n Checker on our pool of 449 subject applications. We recorded the results reported by the checker and counted the frequency of each CF. We also recorded, for each reported CF, the level of severity as reported by the i18n Checker (i.e., Error or Warning). After calculating the results, we broke them down by framework. Note that it was not meaningful to break down the results by language. The reason for this is that the properties checked by the i18n Checker do not vary based on the language of the text, they are fixed based on the containing HTML structure.

TABLE II: Frequency of Configuration Failures

Type of Framework	No. of Subjects with Failures	% of Subjects with Failures
Automatic	187	60%
Server Side	94	68%
All Subjects	281	63%

1) Frequency: To answer RQ4, we calculated the percentage of subjects for which the i18n Checker reported the presence of a CF. We then broke down the detection reports by type of framework.

Table II shows the results of this analysis. As can be seen from the table, overall 281 subjects (63%) had at least one CF in them. This indicates that CFs are as common as LFs. When breaking the results down by framework, we found that server side frameworks had a higher rate (68%) of failures than automatic translation frameworks (60%).

By breaking down the type of framework further, we found that websites using the Drupal multilingual module had the highest rate of CFs (83%), while websites using the Wordpress multilingual plugin had the lowest rate of CFs (only 31%).

2) Severity: To answer RQ5, we checked the level of severity for all of the CFs reported by the i18n Checker and computed how many of these failures were errors and how many were warnings. Then we broke down our results by the type of framework.

Overall, among the subjects that were reported by the i18n Checker as having a CF, we found that 27% of them had failures with severity level Error. This means that almost a third of the subjects could have serious flaws in their rendering and the rendered page's appearance could vary by browser implementation irrespective of the browser's compliance to HTML specification (i.e., the Error resulted in undefined behavior with respect to the standard.)

When breaking down the result by framework, we found that among all of the CFs reported for subjects using automatic translation frameworks 30% had failures with severity Error. For server side frameworks, 21% had failures with severity Error. This indicates that CFs reported for automatic translation framework subjects tended to have more severity than server side framework subjects.

3) Types: To answer RQ6, we counted the different types of general messages provided with the errors and warnings reported by the i18n Checker. For each type of error or warning we counted its frequency as a percentage of all of the errors and warnings.

Table III shows the results of this analysis. The second column in the table contains the types of CFs. The third column contains their frequency, and the fourth contains their percentage among all the failures at the same level of severity. As can be seen in the table, "No character encoding

information" was the most common type of error reported by the i18n Checker. It constituted 35% of all of the reported errors. The most common type of warning was "The html tag has no language attribute," which constituted almost half of the reported warnings. Note that the failure "A tag uses an xml:lang attribute without an associated lang attribute" appears both as an error and as a warning in the table. This is because HTML5 and XHTML5 require developers to use a "lang" attribute if they use an "xml:lang" attribute, while in older standards this is not required and only recommended. This type of failure is classified as an error for pages served as HTML5 or XHTML5, and a warning for pages using older standards.

C. Discussion of Results

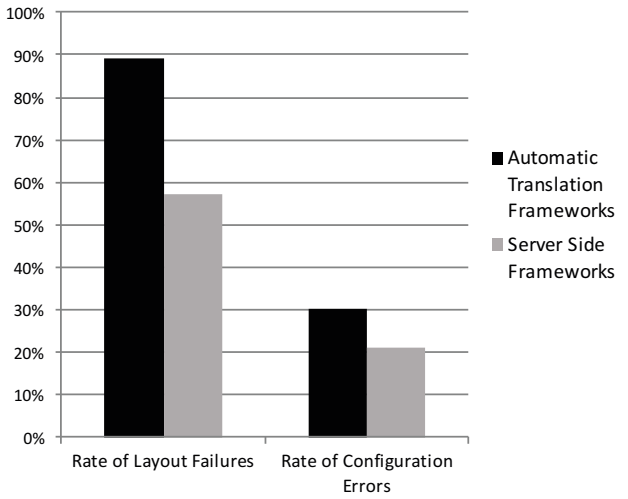

Fig. 5: Rate of Layout Failures and Configuration Errors for Automatic Frameworks vs. Server Side Frameworks

Figure 5 shows the rate of Layout Failures and Configuration Errors (note, warnings are omitted from this count) both for automatic translation frameworks and server side frameworks. As can be seen from the graph, the results of our study show that websites using automatic translation frameworks are more likely to have LFs compared to websites using server side frameworks. They also have a higher rate of configuration errors. While the cause of this is unknown, we hypothesize that since the automatic translation framework is a lower-cost option, as compared to the server-side framework, development teams that are using the automatic translation frameworks could be those on a tighter development budget. These teams may also be less likely to have the resources to thoroughly test the translated versions. Alternatively, this could be the symptom of using a third party translation service that has not carefully crafted the translated strings to fit into the available space. It is important for developers to note that even though automatic translation frameworks are easier to install, they tend to have more problems since they are not fine tuned for the website and include many language options that require extra testing effort.

The frequency results for CFs show that they occur frequently in websites built by both automatic translation frameworks and by server side resource file based frameworks. A common underlying theme for the most common types of CFs (shown in Table III) is that their root cause could be attributable to developer errors with respect to the correct way of using the language and encoding declarations. For example, the two most common types of errors are related to setting the character encoding correctly and within the first 1,024 bytes of the document. Neither of these actions are particularly hard to carry out correctly. In fact, they are both easy to detect and easy to fix, but avoiding them requires that developers be made aware of these requirements. Developers can incorporate a checker, such as the i18n checker, into their tool chain, or frameworks can be designed in such a way that they can automatically manage many of the relevant settings. Delegation to the framework would lower the possibility of configuration failures making it into deployed web sites.

VI. Threats to Validity

A. External Validity

To avoid any bias in the selection of the subject applications, we randomly selected them from a random URL generator, URouLette. The selected pool of subject applications has variety in the popularity, complexity, and size of the webpages. To ensure variety in the languages we performed our experiments on the 12 most commonly used languages on the web. This variation allowed us to ensure that our subjects are representative of web applications in general. However, one limitation of our approach is not including right-to-left languages, such as Arabic, Farsi, and Hebrew. In these languages the layout of the page is mirrored, which needs special handling that is not supported by the underlying framework (GWALI) we used in our analysis. A complete list of the webpages used in our experiments is available from the projects website [19]

The fact that we disabled JavaScript after the webpages were loaded may seem like a threat to external validity. However, disabling JavaScript does not mean that our analysis cannot handle subjects that use JavaScript. Note that we only disable JavaScript after the page was fully loaded. We did this to ensure that dynamically changing content was synchronized between the two pages (i.e., at its initial state), because GWALI cannot compare pages if they are changing (e.g., sliding news banners). The rendering of the pages we performed our analysis on was exactly what the website visitors would see when the page is fully loaded in the browser. Note that disabling JavaScript would only reduce the recall of our detection results if the JavaScript triggered another part of the page to be displayed that contained an IF.

B. Internal Validity

A threat to internal validity is that we used automated analysis tools to classify and identify characteristics of our

TABLE III: Most Common Configuration Failures

Level of Failure	Top Reported	No. of Subjects	Percentage
Error	No character encoding information	29	35%
	Character encoding declaration in a "meta" tag not within 1024 bytes of the file start	16	19%
	A tag uses an "xml:lang" attribute without an associated "lang" attribute	14	17%
	Conflicting character encoding declarations	7	8%
	Content-Language "meta" element used to set the default document language	6	7%
	Multiple encoding declarations using the "meta" tag	5	6%
Warning	The "html" tag has no language attribute	186	47.1%
	Encoding declared only in HTTP header	103	26.1%
	The language declaration in the "html" tag will have no effect	20	5.1%
	A tag uses an "xml:lang" attribute without an associated "lang" attribute	19	4.8%

subjects. These tools can have false positives and false negatives, which could impact the accuracy of our results and claims. However, these tools have been shown to have high accuracy, which minimizes this threat. The language detector we have chosen, CLD2, is used by popular web browsers, such as Google Chrome. It has high accuracy and can reliably detect the language of a webpage. An evaluation shows that it can detect the language of a text with 99% precision and 98% recall [17]. We also tested our framework detector and verified that it can accurately detect whether the type of the translation framework used in the web application is a client based automatic translation framework or a resource files based framework. To do so, we manually examined a large number of subjects that used an automatic translation framework, and made sure that they could be reliably detected. To detect LFs in the translated webpages, we used GWALI. A previous evaluation of GWALI [8] showed that it could detect LFs with high accuracy (91% precision and 100% recall).

Another threat to internal validity is the method we used to identify the baseline language for each of subject applications. To identify the baseline language, we set the browser to send an HTTP request without specifying the Accept-Language header. However, some websites will use the geographical location of the user to respond with a localized version of the requested webpage instead of responding with the original version. We performed our experiments from a machine in the United States, so if this geo-localization occurred then our analysis would have erroneously identified the American English version as the baseline. However, we do not think this would affect our results regarding the frequency of IF since the widespread popularity of English on the web means that this language version is generally well tested and is highly likely to be rendered correctly.

C. Construct Validity

A threat to construct validity is our selection of the metrics to measure the severity of LFs and CFs. For LFs, an alternative metric for severity can be measured by having a set of users visually examine a pair of webpages and rate the amount of distortion the translated version exhibits. However, such a rating can be highly subjective. Instead, we chose to use a more objective and quantifiable metric, the number of inconsistent relationships in the edges of the LG. Intuitively, a high number of inconsistent edges in the LG means more distortion has occurred to the webpage after translation. We think the amount of distortion is a good indicator of the severity of the failure since changes in elements' position and visual relationships affects a website's rendered appearance, which can impact usability and, more broadly, the user experience. To measure the severity of CFs, we used the number of errors and warnings reported by the i18n Checker. We decided to use this classification because it represents the consensus judgment of the W3C Internationalization (i18n) Group on the severity and impact of a particular CF on a webpage.

VII. Related Work

There are two areas of related work that we discuss in this section. First, we discuss several studies that have been conducted to investigate issues related to internationalization on the web. Although these have not focused on analyzing the presence of internationalization failures on the web, they have covered related topics. Second, we give an overview of techniques that are used to detect the presence of layout failures in web and mobile applications.

A. Studies on Internationalization

Work by Becker [20] studied the usability of the webpages of online companies from an internationalization perspective. The study analyzed 17 localized websites to explore sensitivities to cultural and religious differences in the global marketplace. It also explored the quality of translation for bi-directional languages. In his work, Becker did not focus on analyzing the presence of IFs. Instead, the focus was on the usability of websites in terms of colors and images that are acceptable in targeted cultures.

A study by Kondratova and Goldfarb [21] explored the UI design preferences of different countries and cultures around the world. In particular, the study focused on the color preferences in different cultures. The authors surveyed the usage of color in website interfaces in different countries, then provided recommendations for country-specific color palettes that were appropriate for the different cultures. This study helped in understanding cultural preferences in UI design. However, it does not address the issue of IFs' presence on the web.

B. Detection and Prevention Techniques

Our previous work GWALI [8] and the W3C i18n Checker [13] are the techniques we used in this paper for detecting the two different types of internationalization failures. GWALI provides the ability to detect LFs in webpages, while the i18n Checker can detect CFs. Neither of the previous works performed an empirical study to investigate the frequency, types, or severity of internationalization failures in the web.

TranStrL [10] is a technique that locates hard-coded strings in web applications that developers need to isolate in resource files for translation. This technique simplifies the process of translating web applications that were not internationalized at the beginning of the software development process.

In the rest of this subsection, we discuss techniques that are related to finding UI related failures in web applications. Although these techniques cannot be used to detect IFs, they are closely related due to their focus on finding failures in the appearance of web application UIs.

LayoutTest [22] is a software library developed by LinkedIn to test the layout of iOS applications. The library allows developers to define a data specification (dictionary), which is used to generate different combinations of testing data. Then the interface of the app is exercised with the generated data. The library automatically checks for overlapping elements in the UI and allows the developers to write assertions to verify that the layout and the content in the interface are correct.

Apple provides users of its Xcode IDE with the ability to test the layout of their iOS apps and the apps' ability to adapt to internationalized text [7]. This is performed using "pseudo-localization," which is a process that replaces all of the text in the application with other dummy text that has some problematic characteristics. The new text is typically longer and contains non-Latin characters. This helps in early detection of internationalization faults in the apps. The technique also tests right-to-left languages by changing the direction of the text. Manual effort from the developers is still needed though since they have to verify that elements in the GUI reposition and resize appropriately after the pseudo-localization and direction change.

WebSee [23], [24], [25] and FieryEye [26] are techniques that utilize computer vision and image processing techniques to compare a browser rendered test webpage with an oracle image to find visual differences. The two techniques are useful in usage scenarios where the page under test is expected to match an oracle (e.g., mock-up driven development and regression debugging).

Invariant specification techniques such as Selenium WebDriver [16], Cucumber [27], Cornipickle [28] and Crawljax [29] allow developers to write assertions on the page's HTML and CSS data. These assertions will then be checked against the webpage's DOM. These techniques are useful in regression testing to verify that the functionality and the layout of the webpage has not changed after modifying the code.

VIII. Conclusion

In this paper, we presented the results of an extensive empirical study we performed to understand internationalization failures in the web. For this study, we analyzed over 449 internationalized websites to find common trends in the frequency, severity, and types of internationalization failures.

The results of our study show that internationalization failures are common in the web. However, not many tools have been developed to handle this problem. This motivates software engineering researchers to do further research to tackle the problem. In addition, web developers should be aware of the amount of expansion and contraction that could occur to the text after translation, and they should allow the layout of their websites to adapt this expansion and contraction. Also, when using automatic translation frameworks, developers should be aware of the tradeoff between the easiness of installing them and their correlation with higher amount of internationalization failures.

In general, our results provided interesting insights for developers about the common types of internationalization failures. We hope these findings help developers avoid internationalization failures by understanding them in more depth.

Acknowledgment

This work was supported by National Science Foundation grant CCF-1528163.

References

[1] B. J. Fogg, J. Marshall, O. Laraki, A. Osipovich, C. Varma, N. Fang, J. Paul, A. Rangnekar, J. Shon, P. Swani, and M. Treinen, "What Makes Web Sites Credible?: A Report on a Large Quantitative Study," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '01. New York, NY, USA: ACM, 2001, pp. 61–68.

[2] F. N. Egger, ""Trust Me, I'm an Online Vendor": Towards a Model of Trust for e-Commerce System Design," in *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '00. New York, NY, USA: ACM, 2000, pp. 101–102.

[3] A. Everard and D. F. Galletta, "How Presentation Flaws Affect Perceived Site Quality, Trust, and Intention to Purchase from an Online Store," *J. Manage. Inf. Syst.*, vol. 22, no. 3, pp. 56–95, Jan. 2006.

[4] B. Esselink, *A Practical Guide to Localization*. John Benjamins Publishing, 2000, vol. 4.

[5] "Google Website Translator," https://translate.google.com/manager/website/.

[6] "IBM Guidelines to Design Global Solutions," http://www-01.ibm.com/software/globalization/guidelines/a3.html.

[7] "Apple Internationalization and Localization Guide," https://developer.apple.com/library/ios/documentation/MacOSX/Conceptual/BPInternational/BPInternational.pdf.

[8] A. Alameer, S. Mahajan, and W. G. Halfond, "Detecting and localizing internationalization presentation failures in web applications," in *Proceeding of the 9th IEEE International Conference on Software Testing, Verification, and Validation (ICST)*, April 2016.

[9] M. Tamm, "Fighting Layout Bugs," https://code.google.com/p/fighting-layout-bugs/.

[10] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun, "Locating Need-to-Externalize Constant Strings for Software Internationalization with Generalized String-Taint Analysis," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 516–536, April 2013.

[11] ——, "Locating Need-to-Translate Constant Strings in Web Applications," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: ACM, 2010, pp. 87–96. [Online]. Available: http://doi.acm.org/10.1145/1882291.1882306

[12] S. Choudhary, H. Versee, and A. Orso, "WEBDIFF: Automated Identification of Cross-Browser Issues in Web Applications," in *2010 IEEE International Conference on Software Maintenance (ICSM)*, Sept 2010, pp. 1–10.
[13] "W3C Internationalization Checker," https://validator.w3.org/i18n-checker/.
[14] "A Java implementation of the W3C Internationalization Checker," https://github.com/w3c/i18n-checker.
[15] "Scrapbook X Add-on for Firefox," https://addons.mozilla.org/en-US/firefox/addon/scrapbook-x/.
[16] "Selenium," http://docs.seleniumhq.org/.
[17] "Compact Language Detector 2," https://github.com/CLD2Owners/cld2.
[18] A. J. Scott and M. Knott, "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, vol. 30, no. 3, pp. 507–512, 1974.
[19] "GWALI: Global Web Testing," https://sites.google.com/site/gwaliproject.
[20] S. A. Becker, "An Exploratory Study on Web Usability and the Internationalization of US E-Business," *Journal of Electronic Commerce Research*, pp. 3–4, 2002.
[21] I. Kondratova and I. Goldfarb, "Culturally appropriate web user interface design study: Research methodology and results," *Handbook of Research on Culturally-Aware Information Technology: Perspectives and Models*, pp. 1–21, 2010.
[22] "LayoutTest iOS," https://github.com/linkedin/LayoutTest-iOS.
[23] S. Mahajan and W. G. J. Halfond, "Detection and localization of html presentation failures using computer vision-based techniques," in *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, April 2015.
[24] ——, "Finding html presentation failures using image comparison techniques," in *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE) – New Ideas track*, September 2014.
[25] S. Mahajan and W. G. Halfond, "Websee: A tool for debugging html presentation failures," in *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST) - Tool Track*, April 2015.
[26] S. Mahajan, B. Li, P. Behnamghader, and W. G. Halfond, "Using visual symptoms for debugging presentation failures in web applications," in *Proceeding of the 9th IEEE International Conference on Software Testing, Verification, and Validation (ICST)*, April 2016.
[27] "Cucumber," http://cukes.info/.
[28] S. Halle, N. Bergeron, F. Guerin, and G. Le Breton, "Testing Web Applications Through Layout Constraints," in *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, April 2015, pp. 1–8.
[29] A. Mesbah and A. van Deursen, "Invariant-Based Automatic Testing of AJAX User Interfaces," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 210–220.