UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Rafael Dias de Mendonca Mesquita

Análise e caracterização de issues de localização e internacionalização de projetos Open Source

Recife
2025
Rafael Dias de Mendonca Mesquita

Rafael Dias de Mendonça Mesquita

Análise e caracterização de issues de localização e internacionalização de projetos Open Source

Trabalho de Conclusão de Curso
apresentado ao Curso de Graduação em
Sistemas de Informação da Universidade Federal de
Pernambuco, como requisito parcial para
obtenção do título de bacharel em Sistemas de Informação.

Orientador (a): Breno Alexandro Ferreira de Miranda

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Mendonça Mesquita, Rafael Dias de.
Analise e caracterização de issues de localização e internacionalização em projetos Open source / Rafael Dias de Mendonça Mesquita. - Recife, 2025.
28 p. : il.

Orientador(a): Breno Alexandro Ferreira de Miranda
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Sistemas de Informação - Bacharelado, 2025.
Inclui referências.

1. Internacionalização e Localização. 2. Open Source. 3. Testes de software. I. Miranda, Breno Alexandro Ferreira de. (Orientação). II. Título.

000 CDD (22.ed.)

Rafael Dias de Mendonça Mesquita

Análise e caracterização de issues localização e internacionalização de projetos Open Source

Trabalho de Conclusão de Curso
apresentado ao Curso de Graduação em
Sistemas de informação da Universidade Federal de
Pernambuco, como requisito parcial para
obtenção do título de bacharel em Sistemas de informação.

Aprovado em: 14/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Breno Miranda (Orientadora)

Universidade Federal de Pernambuco

_____

Prof. Dr. Leopoldo Teixeira (Examinador Interno)

Universidade Federal de Pernambuco

Análise e caracterização de issues de localização e internacionalização de projetos Open Soruce

Rafael Dias de Mendonça Mesquita[1], **Breno Miranda**[1]

[1] Informatics Center – Federal University of Pernambuco (UFPE)
Recife – PE – Brazil

{rdmm,bafm}@cin.ufpe.br

Abstract. *This paper collects and analyzes localization and internationalization issues at Github open source projects doing a direct analysis, checking title, description, discussion and another related issues if needed to find the problem type and the language related to that problem. To collect and analysis this issues was used an query doing with Python that uses Github API and stores random issues, based on a percentage predefined and organize it on a table. That table contains main information like: name, url, created data and closed data. To make the characterization of the issues was used to params: error type that is based on a taxonomy that was elaborated for this paper, and language affect on that issue. To check this data is being used variations of graphs that is possible to see what the most problems that the project face when doing you internationalization and localization process.*

Resumo. *Este trabalho de conclusão de curso coleta e analisa issues de localização e internacionalização em projetos open source do Github, fazendo uma análise direta, observando o título, descrição, discussão e se nescessário outras issues relacionadas para obter o tipo de problema e as línguas afetadas. Para a coleta e analise dessas issues foi utilizado uma query em Python que consumia a API do Github e conseguia pegar uma porcentagem de issues que fosse pre definida de maneira aleatória e organizar em uma planilha com suas informações principais: nome, url, quando foi aberta e quando foi fechada. Para a caracterização dos dados dois parametros foram adicionados a planilha um relacionado ao tipo de erro, esse que por sua vez foi baseado em uma taxonomia retiradas de literaturas complementares, e a língua na qual essa issue foi encontrada. Por fim, para observar esses dados foram utilizados gráficos de diversos tipos para que fosse possível enxergar os padrões e os problemas mais enfrentados pelos projetos, tanto em relação ao erro quanto em relação língua relacionada.*

Palavras-chave: Github; internacionalização; issues; língua; localização.

1. Introdução e Caracterização do Problema

Com o aumento da globalização e das tecnologias ao longo dos anos, cada vez mais projetos e softwares são lançados no mercado em várias outras línguas, isso permite que muitos projetos possam ser acessíveis em diversas partes do mundo. Outro fator relevante é que muito desses projetos são de código aberto e podem ser traduzidos e melhorados por vários tipos de contribuidores e também utilizar diversas ferramentas para uma maior facilidade de localizar o projeto.

Porém essa acessibilidade vem acompanhada de alguns problemas que podem acontecer nos projetos de modo geral, é muito comum ver nesses projetos problemas de tradução, interface, data, entre outros, problemas no qual não conseguiram ser vistos antes de ser colocado

no código final e, além disso, por mais que seja um código descentralizado não é qualquer contribuidor que conseguem achar e resolver esses problemas dado que uma parte deles são bem espeficios e não envolvem apenas tradução, mas também data, câmbio ou até a direção no qual o texto é escrito, como no caso do Árabe ou Hindi. E cada vez mais projeto se vêem obrigados a manter sua aplicação bem localizada para que atinja diversos tipo de usuários ao redor do mundo, projeto com esse nível de trabalho são mais bem notados no mercado tanto no internacional quanto no nacional.

Esse trabalho coleta aleatoriamente issues de variados projetos de código aberto e as classifca em uma serie de aspectos, criados com algumas regras de coleta, que serão analisados e discutidos descritivamente por meio de gráficos que mostrarão quais dos aspectos mais afetam ou são afetados nas issues desses projetos.

2. Background

Para fazer nossa análise é fundamental entender o escopo e o tema sendo abordado e por isso será falado sobre todos os tópicos abordados no trabalho mostrando o que é localização e internacionalização e compilando o básico de como são os projetos que iremos tratar de analisar no trabalho.

2.1. Internacionalização

Internaciolização é o processo de desenvolvimento do software para fazer suporte e dar acessibilidade a outras culturas e identidades fazendo com que a aplicação consiga ser usada de maneira mais facil para usuários de outras partes do mundo. Uma dos principais objetivos da internacionalização é separar o que somente pode ser feito em código e deixar a aplicação apenas na necessidade de ajustes na tradução e na língua, então aspectos como: Formatação de data e hora, câmbio, alfabeto, números, direcionalidade (se a língua é escrita da direta pra esquerda ou esquerda para direita), fontes, entres outros aspectos.

2.2. Localização

Localização é o processo de adaptar o produto já internacionalizado para o um terminado local, tipo de mercado ou para versões especificas como, por exemplo, adaptar o texto para certos contextos e adaptar a parte não textual para deixar a aplicação customizada baseada para cada cliente. A localização é muito importante para o mercado internacional e atualmente um aplicação bem localizada abrage muito a area de mercado de um software ou projeto.

2.3. Open source

Open source é um código criado e designado para ser acessível gratuitamente, no qual qualquer pessoa pode acessa-lo tanto parar ler, quanto para modificar. Esses tipos de código são desenvolvidos de maneira colaborativa e descentralizada, ou seja, toda comunidade trabalha para manter o código funcional e acessível para qualquer um que também queria contribuir e muitas vezes isso acaba deixando o projeto mais barato, flexível e longiquo. Muitas empresas grandes como *Microsoft* adotam esse tipo de projeto como no *Visual Studio Code* que é um projeto muito grade e estável desde a década passada.

2.4. Locale

Locale é uma combinação de linguage, território e conjunto de código que identifica um conjunto de convenções de idioma que são:

- **Collation** (que define como comparar e ordenar um dado de acordo com língua)
- **Conversão de caso**.
- **Classificação de caracteres**
- **Representação data e hora**
- **Simbolo monetário**
- **Representação numérica**

As informações do locale são mantidas em arquivos de origem e comandos como *setLocale* acessam e configuram essas informações dentros das aplicações e a variável de ambiente *LANG* é usada para especificação do locale desejado. Para nomear os locales é utilizado uma conveção que junta a língua, código e território por exemplo o conjunto usado no Brasil seria *pt_BR.ISO639-1* que abreviado fica apenas *pt_BR*.

2.5. Github

Github, assim como Git, é um sitema descentralizado de código que administra e armazena versões de código como uma réplica, na qual pode enviar ou receber qualquer informação de outra replica. Porém, o Github vem com bastantes mais funções prórprias e especiais que facilitam a colaboração e a interação social entre os usuários (como o *issue-tracke*r e o suporte aos *pull requests*). Além disso a plataforma providencia uma API gratuitamente que pode acessar os metadados que facilitam muito diversas análises de projetos. [COSENTINO 2017]

2.6. API do Github

O Github disponibiliza dois tipos de APIs uma REST API e outra GraphQL API nas quais as duas são de uso gratuito e podem ser utilizadas pela CLI do Github, sendo a mais antiga e mais usada pelos usuários a REST API.

O Github com seu REST API disponibiliza uma serie de *endpoints* que podem acessar dados váriados dentro dos projetos via requisições HTTP como *https://api.github.com/*, por exemplo podemos usar a requisição *https://api.github.com/repos/:owner/:repo* para conseguir dados de do repositório *:owner/:repo* como dados de: usuário, time, projeto, *pull resquests*, *issues* entre outros.

Os dados devolidos pela requsição vem no formato JSON que pode vir com vários resultados divididos por páginas que por padrão contém 30 itens em cada, mas esse número pode chegar a 100 dependendo do *endpoint* utilizado. Além de paginação é possível trabalhar com diversos comando que podem ajudar na hora de coletar os dados que serão usados como por exemplo: *https://api.github.com/repos/:owner/:repo/issues?&sort=comments&direction=asc* esse comando busca as issues do projeto *:owner/:repo* e as organiza por números de comentários (*sort=comments*) de maneira crescente (*direction=asc*) [MOMBACH 2019]

2.7. Ferrametas de busca do Github

O Github fornece uma meio de pesquisa que funciona muito bem para direcionar o usuário para os projetos que ele deseja encontrar. É possível fazer uma pesquisa global em todo o Github ou limitar o escopo a uma organização ou repositório, para fazer essa pesquisa é possível utilizar uma sintaxe propria usando com por exemplo: *¿n*, *¡n*, *=¡n*, *=¿n* para procurar por números de repositórios ou quantidade de estrelas que um projeto como **localization starts¿1000** que presquisa projetos com a palavra localization que tenham mais de 1000 estrelas. Vários valores e parâmetros podem ser usados como, tamanho do projeto, tópicos, número de issues e datas dos repositórios.

2.8. Ferramentas de tradução automática

Os Globalization management system (GMS) também conhecido como Translation management system é um software que automatiza diversas partes do processo de tradução da língua humana e maximiza a efiencia da tradução. A ideia do GMS é automatizar todo tipo de trabalho repetitivo e não essencial que pode ser feito por máquinas e deixar o trabalho criativo para os desenvolvedores, ele contém dois tipos de tecnologias: o *process manangement* que automatiza o passo a passo do trabalho e a tecnologia que ajuda o tradutor. [YANG 2024]

3. Metodologia

Este tópico mostra como foi feita a coleta dos dados que serão usados para a análise e pesquisa, ele aborda todo processo desde a coleta e organização dos dados das issues, processo de classificação, tipos de parâmetros que serão analisados e como esses dados foram caracterizados e por fim algumas regras e excessões de partes especificas da análise.

3.1. Coleta das issues

A escolha do projeto não foi uma tarefa simples, considerando que era necessário selecionar um projeto com um número significativo de *issues* relacionadas à tradução. Muitos dos projetos analisados não possuíam uma *label* específica para localização em sua lista de *issues*, e muitos outros que apresentavam essa categorização não possuiam uma quantidade adequada de dados para análise. Por isso o plano utilizado para obter as issues foi de coletar projetos que tenham pelo menos uma label relacionada a localização e depois de coleta-los, analisar uma porcentagem, que será igual para todos, de issues aleátorias de cada projeto e por fim juntar essas issues em uma planilha para serem analisadas. Essa coleta foi feita por uma query construida em python que utiliza a API do github.

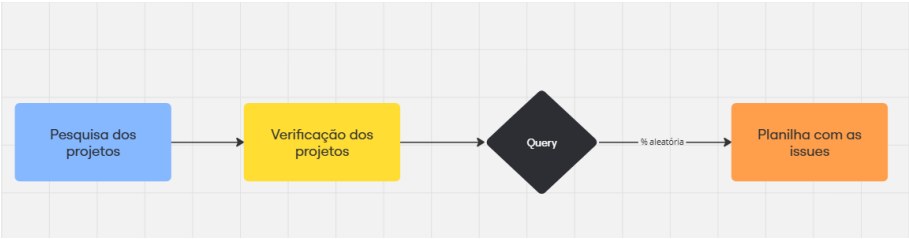
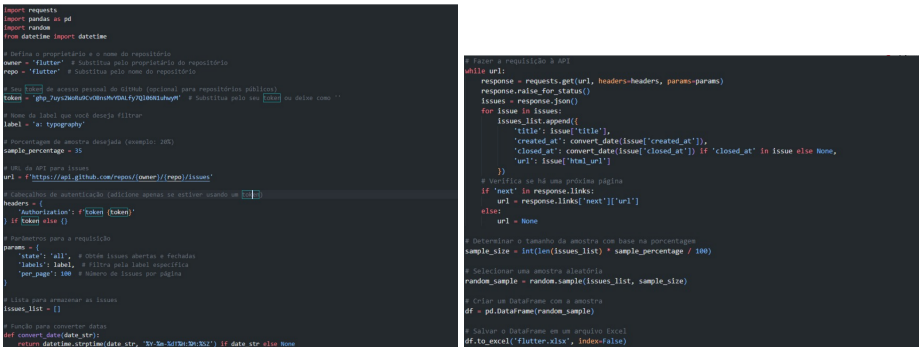


Figura 1. Fluxograma de coleta dos dados

3.1.1. Query

A query utilizada foi feita na linguagem *Python* juntamente com *pandas* para conseguir mexer com os dados e coloca-los em uma planilha, que consome a API Rest do Github para conseguir todas as issues do projetos com uma etiqueta relacionada a localização ou internacionalização, depois disso é feita uma formação para que a lista de issues seja divida em colunas e os valores sejam colocadas nas suas respectivas colunas. Depois que esse processo termina é determinada uma porcentagem da amostra que será coletada e em seguida pega uma amostra aleatoria das issues do projeto e por fim adicionar esses dados em um arquvio *.xlsx*.



```
import requests
import pandas as pd
import random
from datetime import datetime

# Define o proprietário e o nome do repositório
owner = 'flutter'  # Substitua pelo proprietário do repositório
repo = 'flutter'  # Substitua pelo nome do repositório

# Seu token de acesso pessoal do GitHub (opcional para repositórios públicos)
token = "ghp_7uysZWoRuDCvDInsMvYDALfy7Q10bN1uhwyM"  # Substitua pelo seu token ou deixe como ''

# Nome da label que você deseja filtrar
label = 'a: typography'

# Porcentagem de amostra desejada (exemplo: 20%)
sample_percentage = 35

# URL da API para issues
url = f'https://api.github.com/repos/{owner}/{repo}/issues'

# Cabeçalhos de autenticação (adicione apenas se estiver usando um token)
headers = {
    'Authorization': f'token {token}'
} if token else {}

# Parâmetros para a requisição
params = {
    'state': 'all',  # Obtém issues abertas e fechadas
    'labels': label,  # Filtra pela label específica
    'per_page': 100  # Número de issues por página
}

# Lista para armazenar as issues
issues_list = []

# Função para converter datas
def convert_date(date_str):
    return datetime.strptime(date_str, '%Y-%m-%dT%H:%M:%SZ') if date_str else None
```

```
# Fazer a requisição à API
while url:
    response = requests.get(url, headers=headers, params=params)
    response.raise_for_status()
    issues = response.json()
    for issue in issues:
        issues_list.append({
            'title': issue['title'],
            'created_at': convert_date(issue['created_at']),
            'closed_at': convert_date(issue['closed_at']) if 'closed_at' in issue else None,
            'url': issue['html_url']
        })
    # Verifica se há uma próxima página
    if 'next' in response.links:
        url = response.links['next']['url']
    else:
        url = None

# Determinar o tamanho da amostra com base na porcentagem
sample_size = int(len(issues_list) * sample_percentage / 100)

# Selecionar uma amostra aleatória
random_sample = random.sample(issues_list, sample_size)

# Criar um DataFrame com a amostra
df = pd.DataFrame(random_sample)

# Salvar o DataFrame em um arquivo Excel
df.to_excel('flutter.xlsx', index=False)
```

Figura 2. Query para coleta de issues

3.1.2. Utilização da query

A query foi utilizadas em todos os projetos, com pequenas mudanças em elementos chave, como etiquetas (labels), respectivas URLs dos repositórios no GitHub e nome dos arquivos que seriam gerados. Esse processo de coleta não apenas facilitou a junção das informações para que fosse possível analisar as issues, mas também garantiu que as informações fossem organizadas de forma estruturada e acessível. A consulta gera uma planilha estruturada da seguinte maneira:

- **title**: Nome da issue, que resume o conteudo da issue.
- **created at**: Data (aaaa-mm-dd hh:ss) em que a issue foi aberta.
- **closed at**: Data (aaaa-mm-dd hh:ss) em que a issue foi fechada
- **url**: Link direto para a issue no GitHub

3.2. Projetos coletados

No total, foram coletados 30 projetos e a princípio, foram usados dois métodos para a coleta das issues: o primeiro consistia na busca por etiquetas (labels) relacionadas à localização, enquanto o segundo usava a estratégia de palavras chave, como ”**translation**”, ”*localization*, ”*internationalization*”, entre outros, caso nenhuma etiqueta relevante fosse encontrada. No entanto, o método baseado em palavras-chave foi descartado ao longo do processo, pois ele não resultava em um número satisfatório de issues, e a boa parte das ocorrências identificadas não possuía relação direta com localização (l10n) e internacionalização (i18n).

A lista final de projetos selecionados foi:

- **Neovim**: issues totais: 11.980, label: *localization*, issues l10n: 58
- **Notepad++**: issues totais: 10.460, label: *translation*, issues l10n: 1317
- **Vim**: issues totais: 8371, label: *translation e i18n*, issues l10n: 264
- **Geany**: issues totais: 11.980, label: *translation e i18n*, issues l10n: 55
- **Nextcloud**: issues totais: 19.133, label: *feature: language l10n and translation*, issues l10n: 466
- **Openstreetmap**: issues totais: 2306, label: *i18n*, issues l10n: 62
- **Rocket Chat**: issues totais: 16033, label: *feat: i18n*, issues l10n: 138
- **Brave browser**: issues totais: 39.999, label: *l10n* , issues l10n: 500
- **Audacity**: issues totais: 4.452, label: *translation/i18n*, issues l10n: 36
- **QGIS**: issues totais: 32.094, label: *Localization*, issues l10n: 39
- **Flutter**: issues totais: 10.1031, label: *a: internationalization*, issues l10n: 1659

- **Visual Studio Code**: issues totais: 186.794, label: *l10n-platform e L10N*, issues l10n: 543
- **Power Toys**: issues totais: 30.345, label: *Area-Localization*, issues l10n: 590

3.2.1. Porcentagem escolhida

A análise foi feita diretamente e manualmente, analisando issues por issue e se nescessário analisando até mesmo as alterações feitas dentro do *pull request* da issue e outras issues relacionadas, por isso analisar 100% das as issues era iniável. Foi preferido pegar 35% de cada projeto que calculando a soma o resultado será próximo das 2000 issues, um número suficiente para por em prática a taxonomia utilizada no trabalho que será explicada a seguir.

3.3. Analise das issues

A análise das issues de localização (l10n) foi realizada por meio de uma investigação manual, onde cada issue na tabela foi examinada individualmente, ou seja, cada link de issues foi acessado, analisado minuciosamente e classificado dentro de um item da taxonomia . Essa análise considerou o título, a descrição e a forma como os responsáveis pelo projeto lidaram com cada problema identificado. Para facilitar a classificação das issues, foram criadas duas colunas principais:

- **error type**: Tipo de erro de localização identificado na issue será usada a taxonomia apresentada anteriormente.
- **language**: Qual língua a issue apresenta o problema ou em qual língua foi testada o mesmo.

3.3.1. Erros mais comuns

Na coluna **error type**, utilizamos uma taxonomia baseada em literaturas especializadas. Tradicionalmente, os erros de localização são categorizados em quatro tipos principais:

- **Miss translation**: Quando uma parte do texto não é traduzida ou a tradução apresenta erros significativos

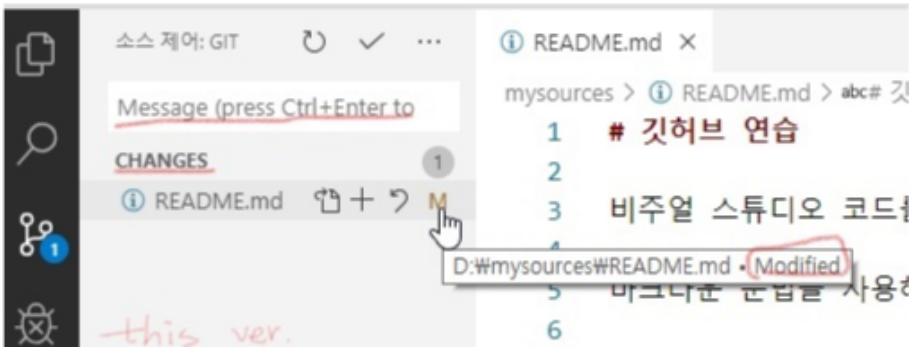


Figura 3. Exemplo Miss Translation

- **Truncation**: Quando um texto ou elemento da interface do usuário (UI) é cortado, tornando o conteúdo ilegível ou não funcional.

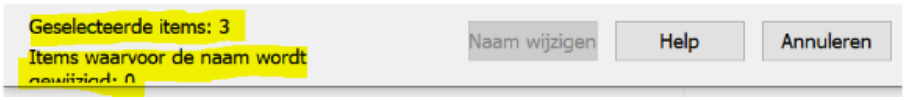


Figura 4. Exemplo Truncation

- **Ellipsis**: Semelhante à truncation, mas o texto é ocultado por reticências, dificultando a compreensão.

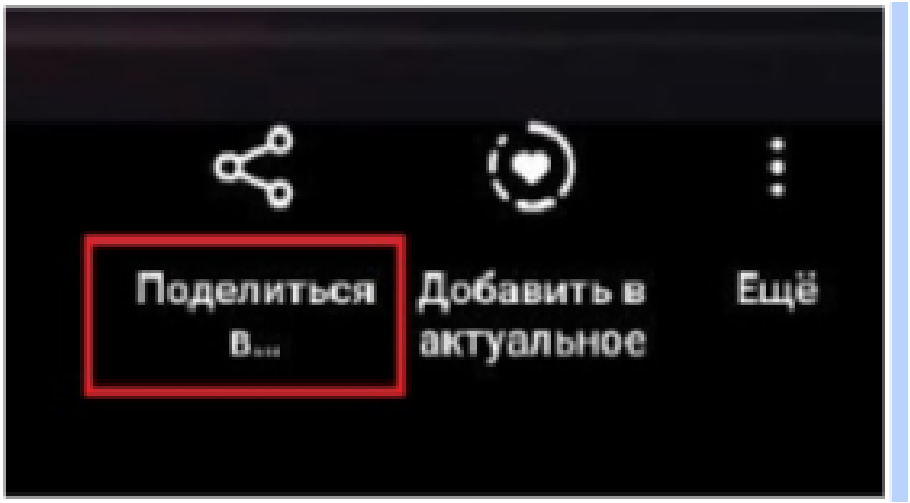


Figura 5. Exemplo Ellipsis

- **Overlap**: Quando um texto ou elemento da UI se sobrepõe a outro, comprometendo a clareza da apresentação.

Figura 6. Exemplo Overlap

3.3.2. Literaturas complementares

Os erros que geralmente são abordados na literatura acadêmica foram insuficientes para classificar de maneira abrangente os variados tipos de issues encontradas nos projetos analisados. Isso se deve ao fato de que cada projeto apresenta particularidades próprias, incluindo diferentes formas de categorizar e tratar problemas relacionados à localização. Dessa forma, foi necessário pesquisar e utilizar literaturas e conteúdos complementares para integrar referências adicionais ao estudo, o que permitiu o aumentdo do alcance da taxonomia adotada para classificação das issues.

No entanto, fazendo a pesquisa foi encontrado um número reduzido de publicações acadêmicas diretamente voltadas para esse tema específico. Entre as referências utilizadas, a maioria dos trabalhos analisados seguiu uma abordagem semelhante à proposta deste trabalho, ou seja, a criação de uma taxonomia própria com o objetivo de classificar e examinar diferentes aspectos da localização e internacionalização. Também é preciso informar que esses estudos não buscaram necessariamente desenvolver uma taxonomia oficial, mas sim conseguir criar classificações personalizadas para atender a contextos específicos de sua pesquisa.

Além disso, as literaturas revisadas complementam a abordagem deste trabalho, sendo o estudo *Automated Internationalization and Localization Testing of GUIs* uma das principais referências para a construção da taxonomia utilizada na análise das issues [ARNAUT 2020] [AWWAD 2016] [HAU 2008]

3.3.3. Taxonomia

- **translation accuracy**: Esse tópico classifica erros nos quais a tradução de um determinado conteúdo não corresponde de maneira correta ao significado dado no idioma utilizado, ou seja, ao contexto do texto ou frase. Isso pode acontecer devido a erros semânticos, interpretações erradas ou falta de adaptação ao contexto cultural da língua, gerando uma experiência possivelmente confusa para o usuário.

Figura 7. Exemplo de translation accuracy
A palavra ”Ignorer”deveria ser trocada pro ”Fermer”que tem mais sentido no contexto

- **terminology consistency**: Acontece quando um mesmo termo ou conceito é traduzido de diferentes formas dentro da aplicação, utilizando palavras ou expressões diferentes

para o mesmo conteúdo. Essa falta de consistência pode gerar confusão para o usuário, dificultando a compreensão do conteúdo e prejudicando a comunicação dentro da aplicação.

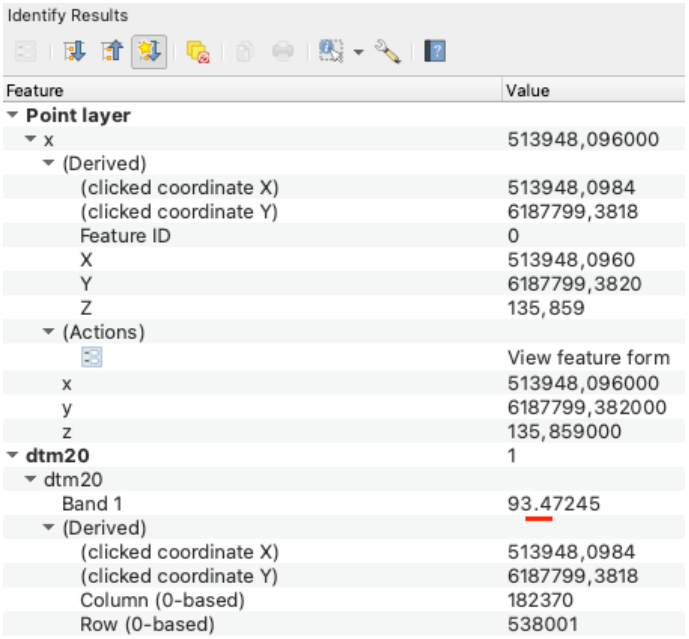


Figura 8. Exemplo de terminology consistency

- **date and/or time error**: Esse tópico classifica erros nos quais a exibição de datas ou horários não segue o formato correto de acordo com idioma e sua região. Isso pode incluir erros de formatação, como a inversão da ordem entre dia e mês, ou o uso de padrões errados para o contexto cultural específico, levando a confusões por parte dos usuários.

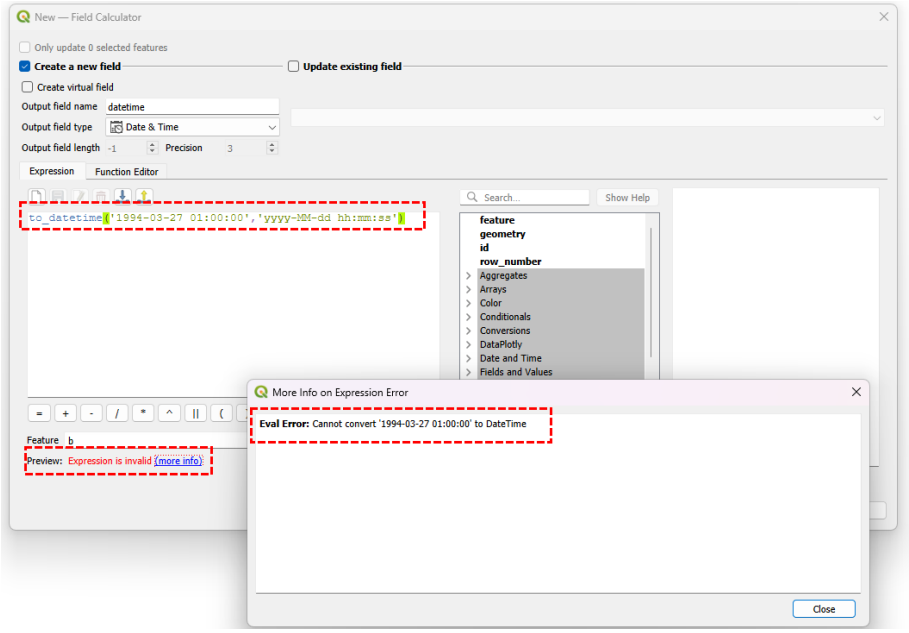


Figura 9. Exemplo de date and/or time error

- **over translation**: Esse tópico classifica erros no qual textos foram traduzidos, mas que, idealmente, deveriam ter sido mantidos no idioma original. Isso ocorre geralmente com nomes próprios, termos técnicos ou marcas, os quais a tradução pode atrapalhar a clareza e a identidade do conteúdo original.

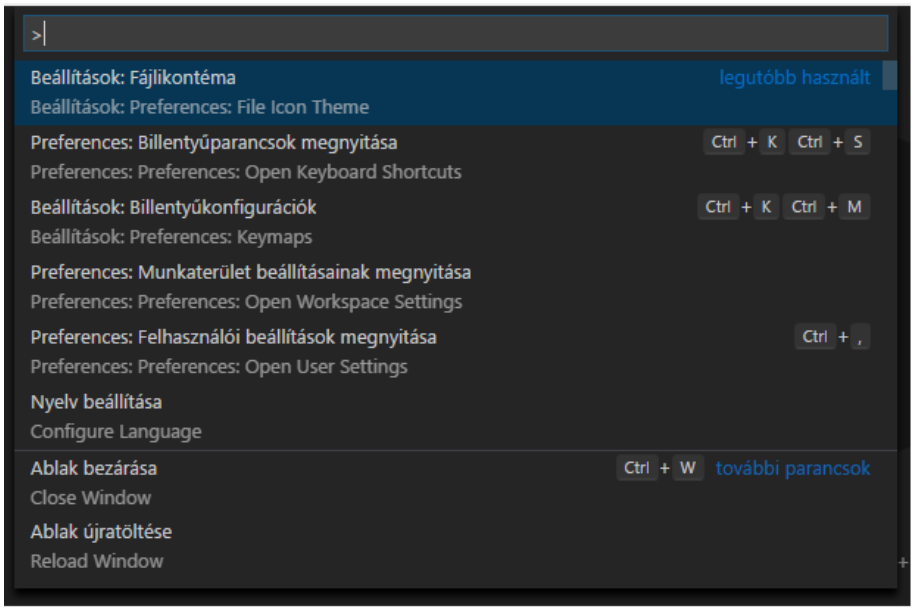


Figura 10. Exemplo de over translation

- **source consistency**: Esse tópico classifica erros nos quais surgem quando há desalinhamentos, truncamentos ou outras inconsistências na localização de um determinado texto ou frase. Isso pode acontecer por falhas no processo de internacionalização da aplicação, e acaba gerando exibições incorretas do conteúdo traduzido.

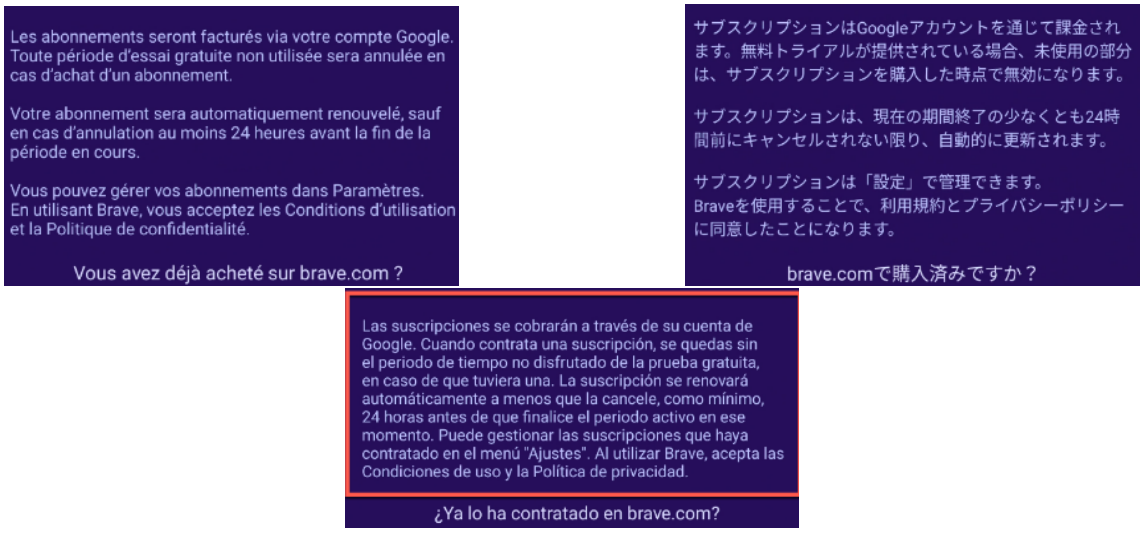


Figura 11. Exemplo de source consistency

- **character display**: Esse tópico classifica erros nos quais caracteres não são renderizados corretamente na interface, gerando uma substituição do texto esperado por

caixas vazias, símbolos como ”?”e ”!”ou caracteres especiais inesperados, como ”&”e ”%”. Esse problema pode estar relacionado à ausência de suporte para determinados conjuntos de caracteres.

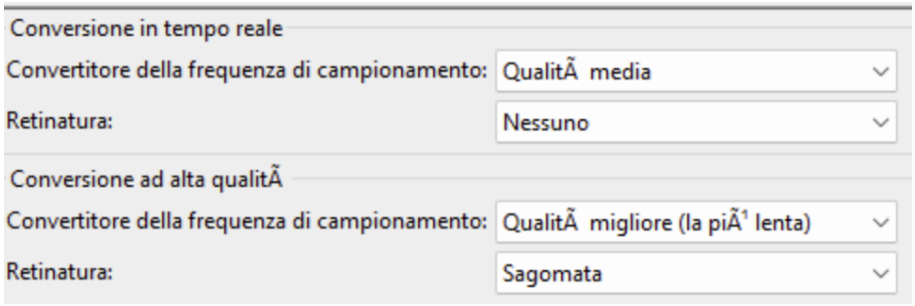


Figura 12. Exemplo de character display

- **link error**: Esse tópico classifica erros nos quais links que não funcionam corretamente em uma determinada língua da aplicação. Esse problema pode ocorrer quando os redirecionamentos para páginas localizadas não são configurados adequadamente ou quando os links da aplicação deixam de funcionar por causa de problemas na tradução ou falta de atualização.

Figura 13. Exemplo de character display

- **wrong layout**: Esse tópico classifica erros nos quais o posicionamento e a disposição dos elementos visuais da interface estão incorretas, isso ocorre principalmente em idiomas que utilizam direções de leitura diferentes, como Árabe (Ar), Hindi (Hi) ou Chinês (Zh). Esse erro pode fazer com que a disposição do conteúdo fique invertida ou desalinhada, prejudicando a usabilidade da aplicação.

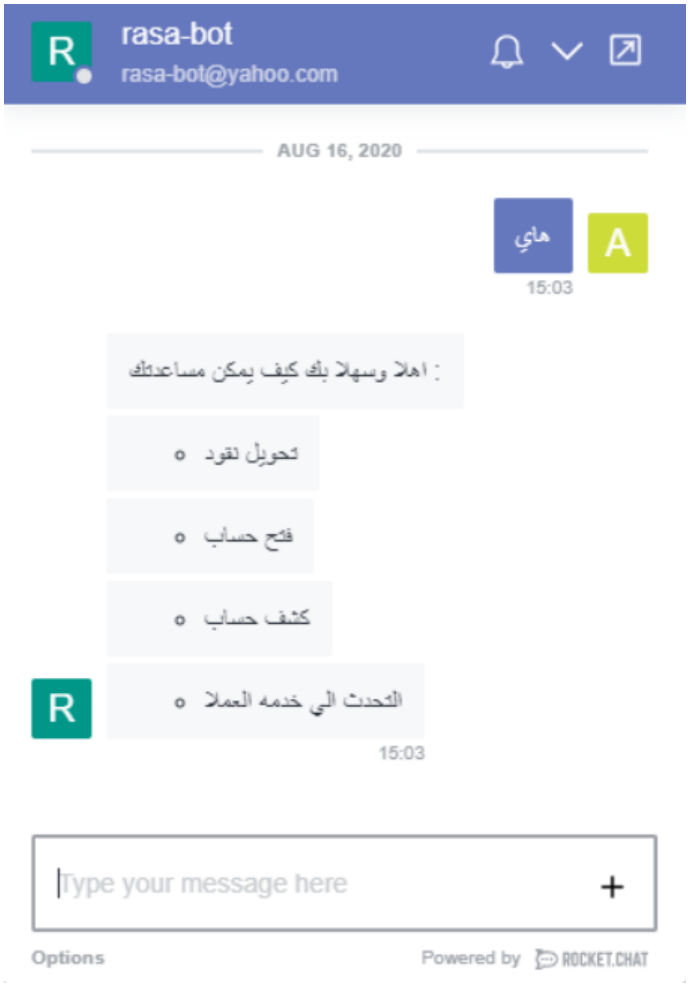


Figura 14. Exemplo de wrong layout

- **hard-coded strings**: Esse tópico classifica erros nos quais determinados textos foram mantidos na língua padrão e não foram colocados no arquivo de tradução da aplicação. Isso pode ocorrer quando *strings* de texto são colocadas diretamente no código em vez de diretamente no arquivo de tradução, impossibilitando sua localização e tradução para outros idiomas.

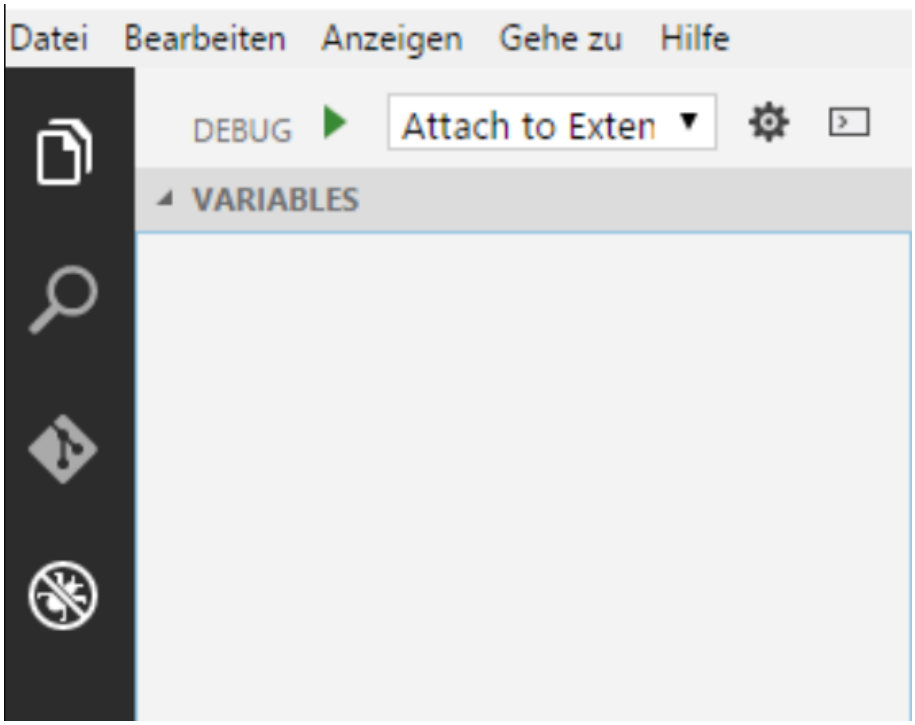


Figura 15. Exemplo de hard coded strings

- **input-output validation**: Esse tópico classifica erros nos quais determinadas funcionalidades da aplicação deixam de funcionar corretamente em uma língua específica. Isso pode ocorrer por falhas na validação e testes antes de disponibilizar o código para o usuário, por diferenças na estrutura da língua ou limitações técnicas que impedem o funcionamente ideal da aplicação em certos idiomas.

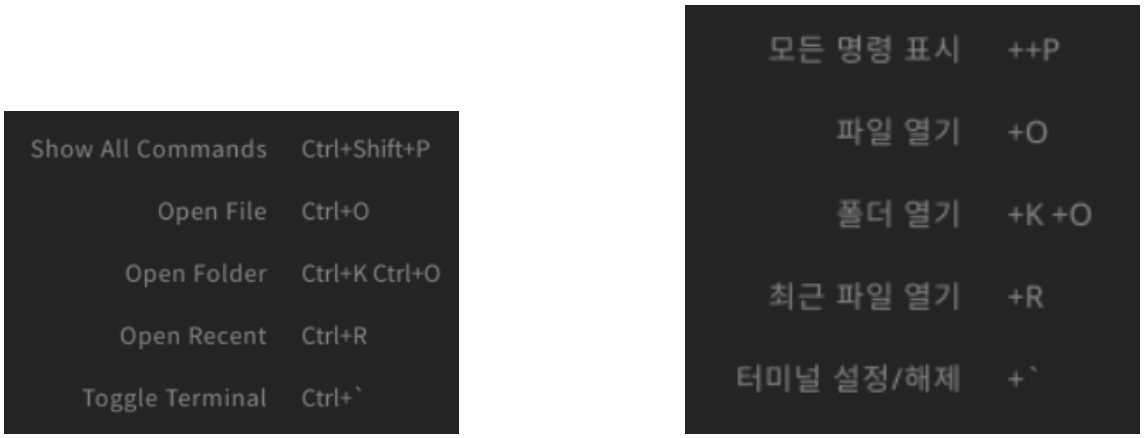


Figura 16. Exemplo de input output validation

- **miss translation (request)**: Esse tópico é usado para representar issues direcionadas a pedidos de adição de tradução no código, em alguns casos o próprio usuário já fornece a tradução.

Figura 17. Exemplo de miss translation (request)

3.3.4. Issues fora da taxonomia

Como cada projeto possui seu próprio conjunto de regras e padrões para categorização de issues, nem sempre uma issue rotulada com uma etiqueta relacionada à localização (l10n) ou internacionalização (i18n) se encaixa nesse escopo específico. Esse problema ocorre porque diferentes projetos podem utilizar nomes e classificações diferentes, resultando na inclusão de issues que, mesmo classificadas na etiqueta de localização, não possuem relação direta com esse tema.

Sabendo dessa variação nos critérios e na falta de definição clara das etiquetas nos repositórios, além dos 13 tipos adicionais de classificação coletados pelas literaturas complementares, foram criadas categorias específicas destinadas a classificar issues que não se enquadram no escopo principal da análise, ou seja, estão fora da área de atuação do trabalho.

As categorias utilizadas para classificar issues que se encontram fora do escopo de análise do presente trabalho são as seguintes:

- **Classificação inadequada (bad classification)**: Esse tópico classifica o erro quando a issue não apresenta qualquer relação com os conceitos de localization (l10n) e internationalization (i18n). Além disso, essa categoria representa quando a relação existente não diretamente a problema com interface ou que afetam diretamente os usuários, mas sim a aspectos como refatoração de arquivos, documentação, configuração ou formatação de arquivos. Dessa forma, issues que não possuem impacto direto na localização ou internacionalização para o usuário são classificadas como inadequadamente categorizadas.
- **Não aceita (not accepted)**: Esse tópico classifica issues que, mesmo que apresentem um problema relevante e que se enquadra no escopo da taxonomia, não são consideradas viáveis pelos donos ou contribuidores principais do projeto para correção ou implementação de uma solução. Isso pode acontecer por vários motivos, como a alta complexidade da alteração necessária para solução do problema, o impacto negativo que a alteração poderia causar no funcionamento geral do sistema ou o fato de os responsáveis pelo repositório não enxergarem essa questão como um problema significativo para ser solucionada naquele momento do projeto.

- **Não constitui um problema (not an issue)**: Esse tópico classifica issues abertas que, não representam um problema que seja necessário modificações no código do projeto. Em algumas situações, determinados repositórios utilizam o sistema de issues para registrar atualizações, comunicar novas versões da aplicação ou compartilhar informações relevantes com a comunidade de contribuidores. Porém, esses registros acabam usando as etiquetas utilizadas na categorização tanto em casos de avisos gerais como atualizações quanto dúvidas especificas como por exemplo duvidas em relação a *placeholders* ou palavras usadas na aplicação.

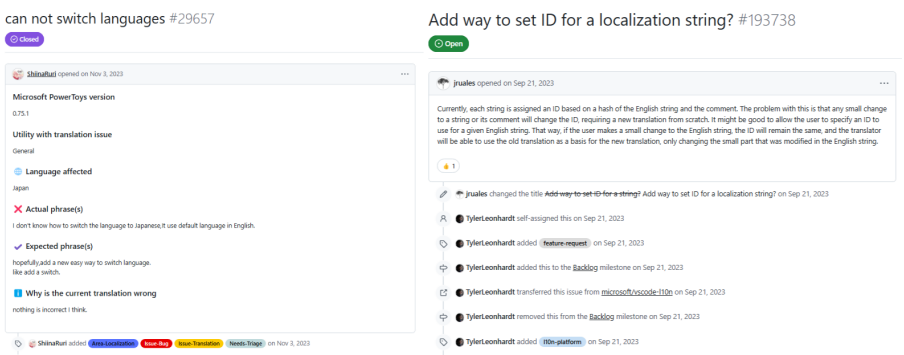


Figura 18. Exemplo de bad classification

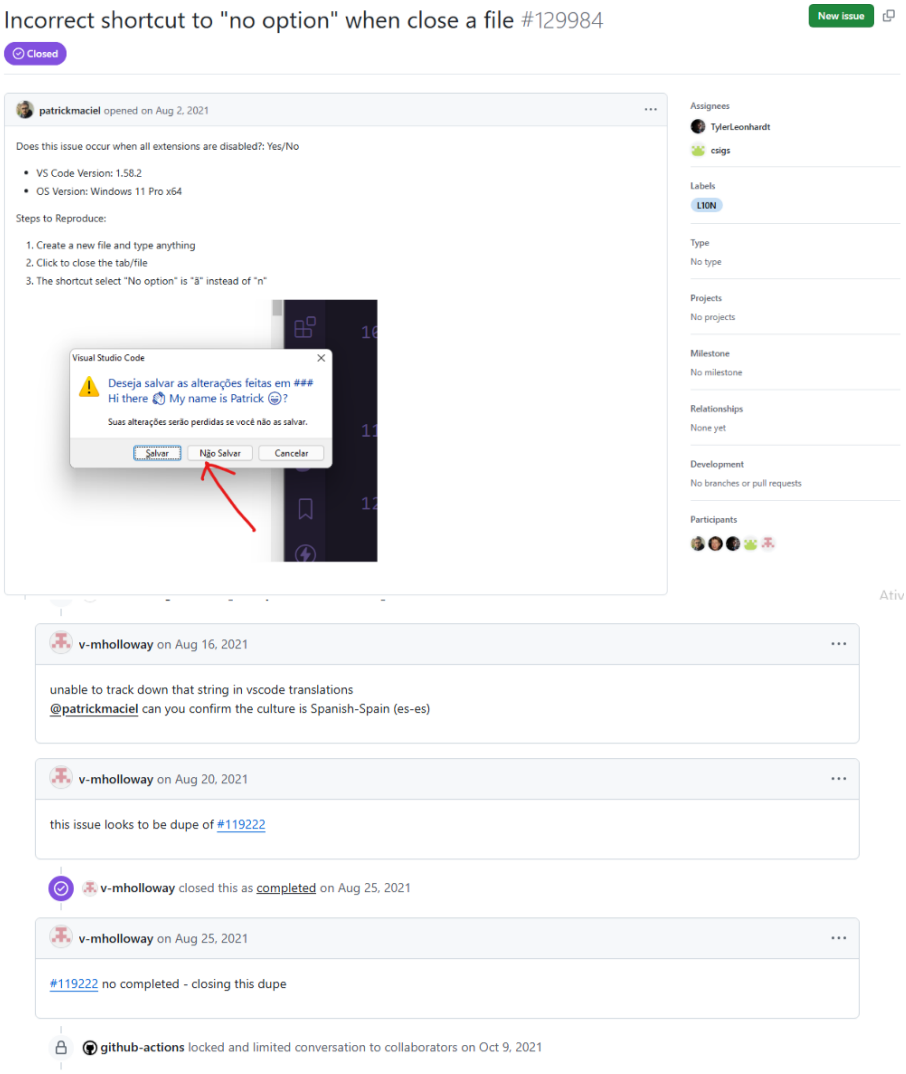


Figura 19. Exemplo de not accepted

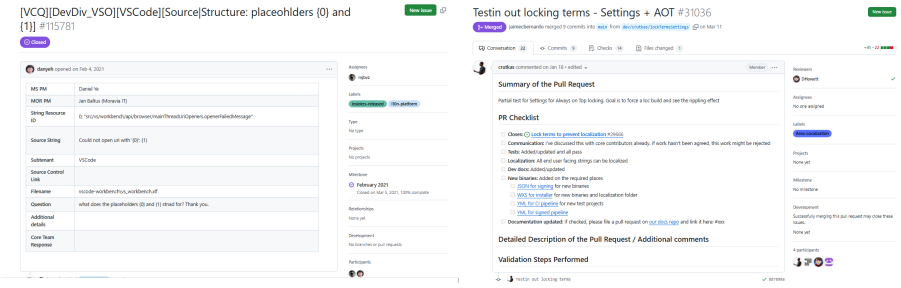


Figura 20. Exemplo de not an issue

3.4. Separação dos dados

Como os três últimos tópicos mencionados não estão relacionados a problemas reais de localização, por isso foi decidido por remover essas *issues* do conjunto de dados analisado. Essa decisão elimina os dados fora de escopo, apresenta maior clareza nos dados utilizados e identifica

de forma precisa quais línguas são diretamente impactadas por problemas de localização. A imagem abaixo apresenta a porcentagem de dados que se manteram na analise dos resultados finais.

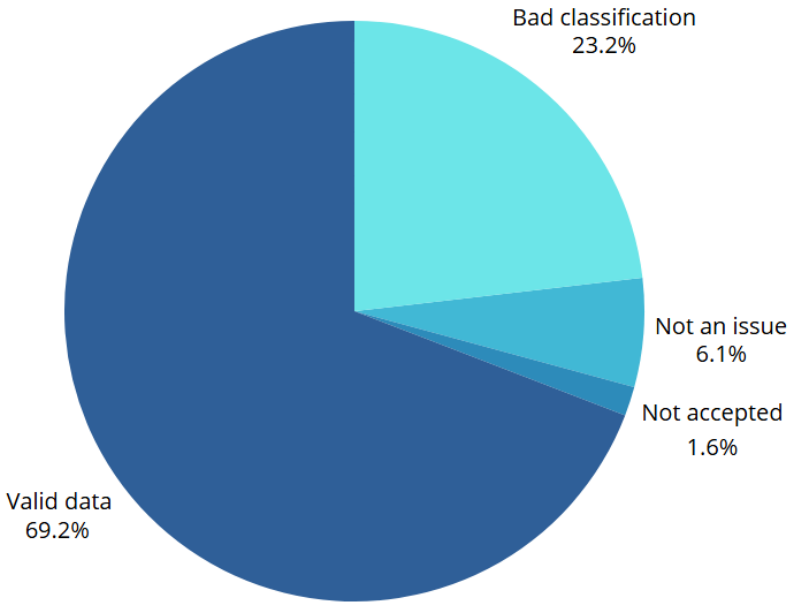


Figura 21. Gráfico de separação dos dados

3.5. Validade da taxonomia

Como ilustrado na figura, dos mais de 2.000 issues analisadas relacionadas à localização (l10n) e internacionalização (i18n), aproximadamente 70% foram consideradas válidas dentro da taxonomia estabelecida. Esse percentual mostra que, apesar da grande quantidade de issues coletadas, muitas delas não estavam diretamente relacionadas a problemas que atrapalham a experiência do usuário final.

Esse comportamento acontece porque não há uma separação clara entre issues que afetam diretamente os usuários e aquelas que dizem respeito a aspectos internos do código ou da infraestrutura do projeto. A categoria que mostra essa distinção no projeto é a **Bad Classification** e esse tipo é o mais discrepante dentre as issues fora do escopo.

Além disso, também é notável que uma parte significativa do espaço do Github destinado as issues é utilizada para outros propósitos, como a abertura de tópicos para esclarecimento de dúvidas, discussões sobre funcionalidades ou até mesmo anúncios de atualizações e novas versões do projeto, o que se enquadra na categoria **Not an Issue**

Por fim, também é notável que um número relativamente pequeno de issues não foi aceito pelos donos ou contribuidores principais dos projetos, o que mostra uma forte interação entre os contribuidores e os responsáveis pela manutenção dos repositórios. Esse cenário mostra que há um alto nível de engajamento e colaboração dentro da comunidade, além disso também mostra um esforço contínuo para aprimorar a qualidade da localização e internacionalização nos projetos analisados, esse dado é retirado do campo **Not accepted**.

3.6. Coleta dos tipos de linguagem

Além disso, o processo de coleta das línguas utilizadas nas issues foi feito por meio do uso do código de linguagem (locale) uma combinação entre lingua, região e terriorio na qual é utilizada para um identificar as convenções dos idiomas, com objetivo de garantir uma análise mais precisa e detalhada dos problemas reportados. A utilização do locale permitiu uma maior compatibilidade entre os dados coletados e os erros relacionados a localização (l10n) e internacionalização (i18n) descritos nas issues, reduzindo o acontecimento de interpretações erradas e descrepancias entre as issues.

Nos casos em que a lingua não era identificada claramente por parte do autor da, foram utilizadas uma serie de regras para garantir a melhor padronização possível. Caso fosse claro na issue a língua na qual ocorre o erro ou na qual o erro foi testado, foi utilizado o locale padrão do idioma em questão, caso não fosse diretamente especificado a região do locale (que é o código apos o hífen) foi considerado apenas a primeira parte do código como: **en**, **nl**, **ru** etc.

A seguir podemos ver quais regras foram adotadas caso a issue não tenha seu locale especificado na sua descrição ou título de forma clara.

3.6.1. Identifição via evidências visuais

- Nos casos em que a issue não mencionava diretamente o idioma, mas imagens do problema relatado foi adotada a lingua que estava na imagem.
- Como a região da língua não podia ser especificamente com precisão foi adotado apenas a primeira parte do locale como dito anteriormente.
- Em alguns projetos, muitos dos pull requests eram feitos diretamente nas issues, que também tinha o tipo do arquivo que foi alterado, logo também era possível saber em que língua aquele erro esta acontecendo ou sendo testado.

3.6.2. Ausência total de informações sobre o idioma

- Se a issue não possuia nenhum tipo de texto, imagem ou arquivo modificado dentro da issue, que pudesse inidicar a lingua utilizada, foi adotado que aquele problema foi testado na lingua padrão do projeto..
- Como a maioria dos projetos analisados tem o **Inglês (en)** como idioma padrão, presumiu-se que o erro ocorria nesse idioma, a menos que houvesse indícios claros em contrário.

3.6.3. Exceção

- O Chinês tem diferenças significativas entre o **Chinês Simplificado (zh-CN)** e o **Chinês Tradicional (zh-TW)**.
- Como a maior parte das aplicações tem o **zh-CN** mais utilizados pelos usuários e contribuidores no caso do Chinês, em caso do erro ser na lingua zh, mas não possuir especificação, foi adotado o **zh-CN** para classificar aquela issue.

3.7. Tradução dos projetos

Ao longo da análise das issues nos 13 projetos examinados, foi possível notar que cada projeto tem maneiras diferentes de lidar com sua maneira de localizar o código. Cada projeto tem uma maneira especifica de organizar e burocratizar suas issues relacionadas a tradução, seja de maneira mais simples, tendo seus proprios contribuidores trabalhando nisso ou de maneira automatica utilizado bibliotecas especializadas para o tema.

Alguns projetos, como o **VSCode**, utilizam plataformas especializadas, como o *Transifex*, para gerenciar suas traduções de maneira automatizada. Esse tipo de ferramenta facilita a coordenação do processo de localização, permitindo que as traduções sejam realizadas de forma dinâmica e integradas diretamente ao fluxo de desenvolvimento do projeto.

Por outro lado, há projetos que preferem por um modelo colaborativo, contando diretamente com sua comunidade de desenvolvedores e usuários para realizar as traduções. Um exemplo dessa abordagem é o **PowerToys**. Esse modelo é descentralizado e barato de trabalhar e além disso reduz o trabalho dos organizadores do projeto com essas questões de localização, além disso ele também permite um envolvimento ativo da comunidade e uma adaptação mais fácil as necessidades dos usuários.

Por fim, alguns projetos, como o **Notepad++**, utilizam uma estratégia diferente, usando o trabalho de linguistas profissionais ou a falantes nativos dos idiomas para revisar e atualizar as traduções. Esse método garamte um alto nível de precisão linguística e cultural, evitando erros que possam comprometer a experiência do usuário.

Essas variações no processo de tradução demonstram que não há um modelo único que se aplica a todos os projetos. Cada método tem suas vantagens e desafios específicos, sendo escolhida de acordo com os recursos do projeto, a estrutura da equipe e o nível de controle desejado sobre a qualidade da localização.

4. Resultados e Discussões

Após a finalização do processo de classificação, todos os dados coletados foram organizados em uma tabela dentro de uma planilha, na qual torna a compreensão e analise mais simples e fácil de serem feitas. A partir dessa organização inicial, foi realizada uma etapa de formatação e refinamento dos dados, na qual todas as issues que não se enquadravam na taxonomia estabelecida foram excluídas, usando apenas as issues dentro do escopo fossem usados para análise.

Além disso, com o objetivo de gerar uma visão mais clara e detalhada dos padrões observados, foram gerados diversos gráficos que permitem uma análise descritiva dos dados de diferentes perspectivas. As interpretações dessas análises serão detalhadas de maneira descritiva estatísticas, tirando algumas conclusões baseadas nos dados retirados da análise, além disso apontar os desafios enfrentados pelos projetos para gerenciar e manter seu código bem localizado e por fim mostrar alguns padrões que foram notados ao longo da análise.

4.1. Tipos de erros mais frequentes

O primeiro parâmetro a ser analisado são os tipos de erros mais recorrentes nos projetos examinados. A identificação desses padrões permite saber quais sãos os desafios mais enfrentados no processo de localização (l10n) e internacionalização (i18n) relacionado aos tipos de erro, ou seja, entender quais são os erros mais enfrentados nas issues e o porque isso acontece.

O erro que mais é encontrado, ou seja, a moda da distribuição, corresponde ao tópico **Miss Translation**, com um total de 432 issues registradas. Esse dado mostra uma grande dificuldade na gestão de erros de tradução que passam despercebidos durante o processo de localização. O alto número dessas issues mostra que de que nenhum método de tradução, seja ele automatizado, colaborativo ou feito por especialistas, é totalmente eficaz. Pequenos erros e defeitos podem se acumular na medida que o software aumenta e se atualiza, dificultando assim a acessibilidade e a boa compressão do usuário como aquele aplicativo.

O segundo tipo de erro mais reportado foi **Translation Accuracy**, que ocorre, na maioria das vezes, devido a falhas de interpretação, uso inadequado de palavras dentro de um determinado contexto ou ainda devido a atualizações do projeto que tornam algumas traduções desatualizadas e fora de contexto. Esse problema está diretamente relacionado a necessidade de um controle rigoroso e burocrático sobre a terminologia utilizada, fazendo com que as traduções permaneçam fiéis ao significado original e adequadas para o contexto de suas respectivas línguas.

Por fim, o terceiro erro mais recorrente foi **Input-Output Validation**, que, mesmo que menos comum, demonstra que alguns projetos apresentam dificuldades na realização de testes de integração ao longo do código, especialmente quando se trata de idiomas que possuem diferentes símbolos, estruturas gramaticais ou sistemas de escrita. Essas particularidades podem gerar falhas inesperadas, que são muitas vezes extramamente difíceis de identificar, mas que podem comprometer significativamente a funcionalidade do software para determinados tipos de língua.

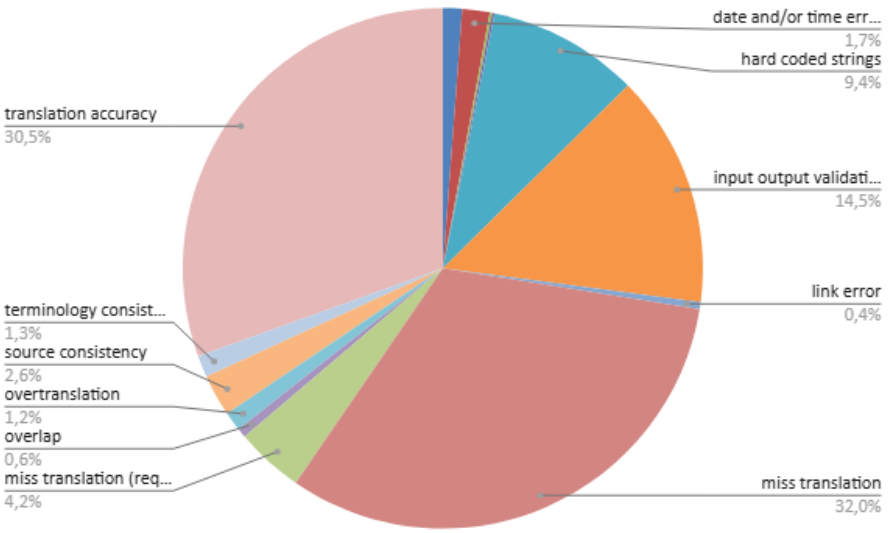


Figura 22. Gráfico de tipos erro

4.2. Línguas que mais enfrentam issues

Nesse tópico vamos analisar a distribuição das línguas que mais tiveram envolvimento com as issues, podendo entender quais delas que geram mais problemas ou tem mais erros sendo testado por elas. Além disso, será possível prever o motivo das línguas terem uma discrepancia em relação as demais.

A primeira língua que tem o maior número de issues relacionadas foi o Inglês (en), apresentando uma discrepância significativa em relação aos outros idiomas. Essa diferença

de dados pode ser explicada por alguns fatorese que influenciam diretamente a quantidade de issues registradas para essa língua:

- O erro **Hard-coded Strings** está diretamente relacionado a língua padrão dos projetos, que, na maioria dos casos, é o Inglês (en). Isso ocorre porque textos escritos diretamente no código em vez de nos arquivos de tradução são feitos na língua padrão do código, que é o inglês. Como resultado, esse tipo de erro representa, no mínimo, **9,4%** das issues reportadas.
- Um fator que conta bastante para análise desse númemro também é a falta de clareza em muitas issues quanto a língua em que o erro foi testado ou ocorreu. Em diversas situações, os usuários relatam problemas sem especificar o idioma no qual o problema foi reportado, sendo necessário utilizar a regra de padronização citada anteriormente.
- Alguns projetos requerem atualizações frequentes na tradução em Inglês para garantir que as traduções para outros idiomas sejam consistentes e fiéis ao significado original. Como consequência, parte significativa das issues classificadas como **Translation Accuracy** está relacionada a necessidade de refinamento da versão em Inglês, garantindo que os termos e expressões sejam utilizados de forma para que também façam sentido nos outros idiomas que foram adicionados tradução.

A segunda língua com maior número de issues reportadas foi o **Chinês Simplificado (zh-CN)**, que, mesmo não apresentando uma discrepância tão grande quanto o Inglês, ainda se destaca em relação as demais línguas. Esse dado pode ser atribuído a diversos fatores, incluindo as particularidades e simbologias da escrita e interpretação do idioma.

Além disso, o Chinês possui um padrão específico para a representação de símbolos, caracteres e fonte utilizada para digitação do texto, o que pode gerar mais problemas para a adaptação do software para esse idioma. Outro aspecto relevante é o fato de que o Chinês é uma das línguas mais faladas do mundo, o que contribui para um volume considerável de usuários testando e encontrados problemas em chinês.

Por fim, o Alemão que foi bem descrepante em relação aos demais e isso acontece devido a sua língua ter um vocabulário extenso e palavras com muitas letras que acaba quebrando bastante o padrão das interfaces, além disso e mais importante, alguns dos maiores projetos possuem vários contribuidores importantes que são alemães e por isso várias issues acabam sendo testadas nessa linguagem e por fim, parecido com a o Chinês muitos contribuidores utilizam o alemão para navegar e utilizar as aplicação fazendo com seja mais comum encontrar mais problemas nessa língua.

As demais línguas aparecem de maneira mais equilibrada, com um percentual variando entre 3% e 7% das issues registradas. Esse padrão sugere que, embora problemas de localização e internacionalização sejam comuns em diversas línguas, eles ocorrem de forma mais distribuída entre os diferentes idiomas suportados pelos projetos analisados.

Em muitos desses casos, os erros relatados estavam relacionados a ajustes na tradução e refinamento da interpretação textual, garantindo maior precisão e coerência na versão localizada. Além disso, algumas dessas issues resultaram em correções efetivas por parte dos desenvolvedores, evidenciando um esforço contínuo para aprimorar a qualidade das traduções e a acessibilidade das interfaces em múltiplos idiomas.

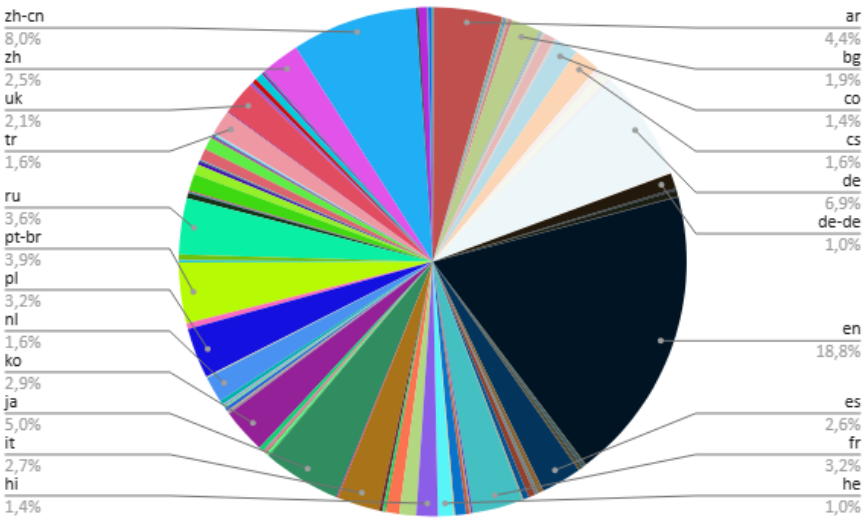


Figura 23. Gráfico de código de línguas

4.3. Erros que mais acontencem nas línguas com mais issues

Nesse tópico vamos falar sobre os tipos erros que mais comuns nas 3 línguas que mais foram relacionadas nos ultimos dois tópicos e a partir disso poderemos tirar alguns dados de qual erro afeta mais qual língua com um número bem expressivo e entender o que pode causar esses erros em cada uma delas.

4.3.1. Inglês

A língua mais erros como visto anteriormente foi o inglês, que acontece por ser a língua padrão de vários projetos e por várias issues não definirem suas issues com qual língua foi usada pra testar o erro em questão. Porém, mesmo assim podemos ver números bem expressivos, como o **hard coded strings** que possui 32% dos problemas em inglês, isso acontece pelo fato que todos as issues classificadas como *hard coded* também serem classificadas em inglês que é a lingua padrão caso não tenho sido especifico na issue qual língua aquele problema estava sendo testada.

Além disso, é possivel ver que inglês também sofrem com miss translation e translation accuracy e por mais que seja a língua padrão dos projetos, ela também precisa estar bem escrita e com um contexto bem definido, vários problemas de erro de digitação e de palavras com contexto errados e isso acaba atrapalhando todas as outras línguas a serem traduzidas, tendo em vista que os projetos serão traduzidos baseados no inglês padrão do projeto.

Por fim o último erro mais discrepante é o *input output validation* que é esperado já que váriados bugs podem acontece quando se adiciona ou atualiza seu projeto com intenção de internacionalizar e localizar, além disso alguns deles também caem na regra da issue não detalhada citada a pouco.

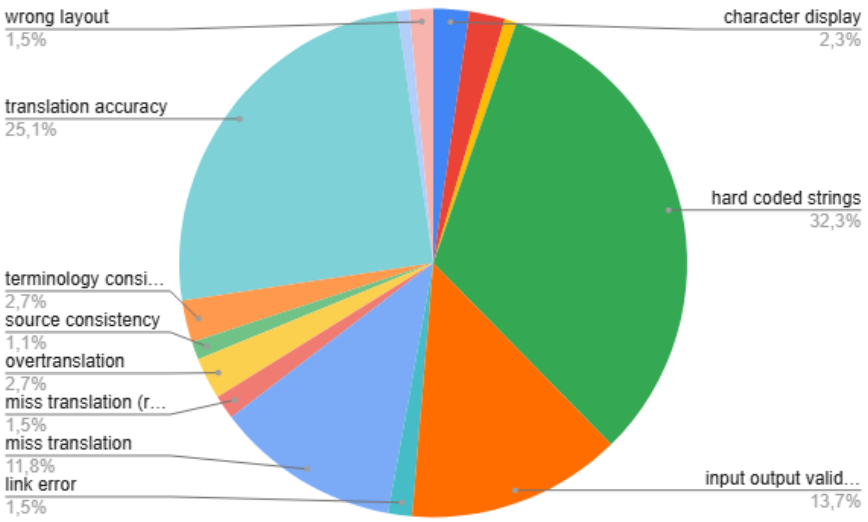


Figura 24. Gráfico dos erros que ocorreram em Inglês

4.3.2. Chinês

O Chinês que é a segunda língua com mais problemas na nossa análise, tem uma grande parte, aproximadamente 40%, dos seus erros concentrados em *input output validation* isso pode acontecer devido a diferenças entre as línguas já que o chinês usa o mandarim diferente do alfabeto usado no inglês, além disso boa parte da comunidade dentro dos projetos era chinesa e isso contribui para que a maioria dos erros seja testado e resolvido em Chinês.

Os valores de *miss translation* e *translation accuracy* são parecidos com o em inglês isso mostra como toda as línguas precisam de ajuste com seu digitação e contexto e que realmente esse problemas acabam passando despercebido na hora de testar o código para subir ao público

Por fim, o *source consitency* que foi bem discrepante em relação aos outros e isso acontece em muitos casos pelas peculiaridades da língua chinesa escrita, mandarim tem uma serie de simbolos que algumas vezes saem cortados ou ilegíveis, além disso em alguns casos a fonte que deveria ser usada não é a correta e acaba dificultando a leitura do usuário.

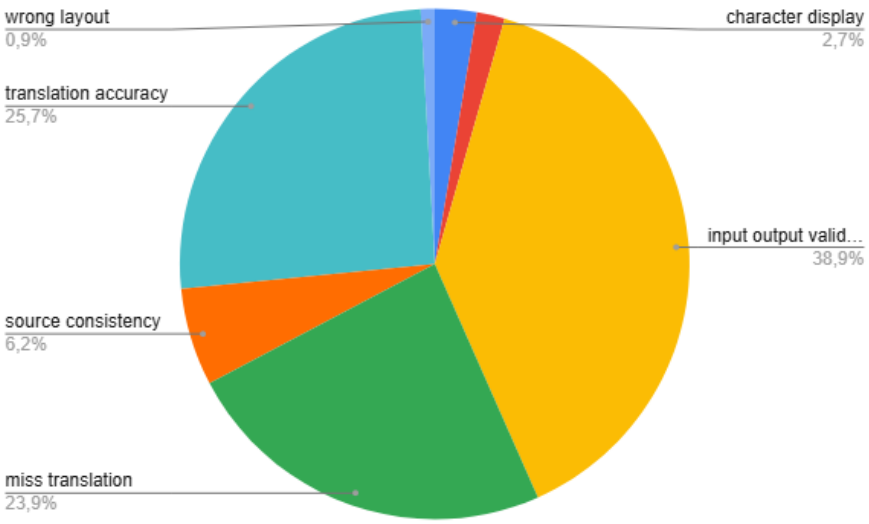


Figura 25. Gráfico dos erros que ocorreram em Chinês

4.3.3. Alemão

O alemão teve a maior taxa de *miss translation* e *translation accuraccy* dentre as 3 línguas, tendo quase 3/4 dos seus problemas voltados para erros tradução e contexto, boa parte da comunidade dos projetos fala alemão e por isso foi uma língua muito testada e relevante para as análises e mesmo assim teve muitos problemas desse estilo, muito vem da falta de preparo dos contribuidores quando trabalham em suas traduções e acabam errando pequenas palavras ou até esquecendo de localizar algumas frases e também uma falta de burocracia com as traduções que são utilizadas pelo projetos, muitos deles tem poucas formas de revisar, tendo em vista que estão trabalhando em outra língua e por isso achar esses erros se tornam uma dificuldade maior.

Além disso, um dos erros que se destacou foi o hard coded string, isso acontece pois um dos projetos com mais issues, o VSCode, teve um periodo inicial sendo trabalhado por contribuidores que falavam alemão, então o projeto estava na língua padrão alemão e pouco tempo depois seu padrão foi mudado para inglês. Outro motivo para isso acontecer também se deve a boa parte da comunidade testar o código em alemão e achar partes do texto que não estão escritos diretamente no código e não nos arquivos de tradução.

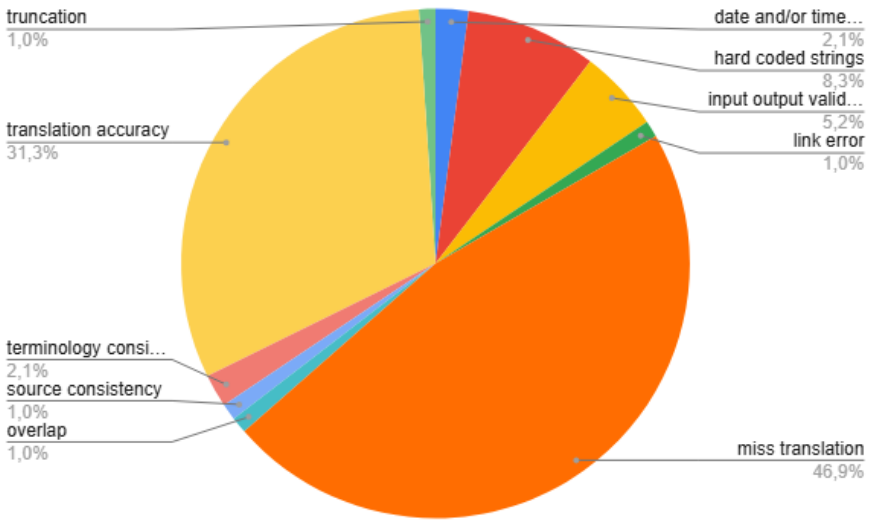


Figura 26. Gráfico dos erros que ocorreram em Alemão

4.4. Tempo médio de fechamento dos 6 erros mais comuns

Outro parametro analisado foi o tempo médio no qual as issues foram fechadas, desse cálculo foram retirado as issues que estavam abertas, que são por volta de 250 issues, esses dados foram tirados da quantidade total de issues ou seja, parte dessas 250 issues temos issues fora do escopo. Foi feita uma coleta dos campos *created_at* e *closed_at* baseada em cada tipo de erro e esse dado será mostrado em número de dias.

A issue com a média mais alta de issues é o **Input output validation** e isso acontece devido ao alto grau de especificidade dos problemas que acontecem em determinadas línguas, já que esse problema tem issues que apenas acontecem em um determinado contexo com uma língua específica, alguns desses problemas envolvem algumas refatorações ou até mesmo dependem de bibliotecas externas em caso do projeto ter uma tradução automática para que o problema seja resolvido, todos esses fatores tornam bem dificil a resolução desse tipo de issue.

O segundo maior foi o *Source consistency* que, por mais que em menor escala, também envolve problemas bem especificos, porém diretamente ligado a interface e aos textos. Com a variadade de tamanhos de palavras, caracteres e simbolos de diversas línguas, manter uma interface ideal para todos os contextos é uma tarefa dificil para os contribuidores, além disso é necessário manter o texto bem coerente para todas as línguas do projeto e por isso o tempo médio de resolução se torna tão alto.

Os valores de *Miss translation*, *Translation accuracy* e *Hard coded strings* são muito altos tendo em vista que em sua grande maioria são problemas mais simples que as vezes apenas envolvem uma linha código ou uma pequena troca nos arquivos de tradução. Isso acontece devido a baixa prioridade que a comunidade e os donos do projeto para resolver problemas desse tema, já que tradução não configuram, em sua maioria, problemas muito graves para o usuário e acabam sendo deixadas mais de lado em relação a outros tipos de problemas.

Por fim, o *Miss translation (request)* que tem um número razoável, tendo em vista que traduzir o projeto praticamente ou totalmente do zero seja uma tarefa mais complexa que demanda tempo e alguma burocracias no qual o contribuidor precisa passar. Além disso, é

possível trabalhar com traduções mais rápidamente devido a parte dos projetos fazerem uso de ferramentas de tradução automática.

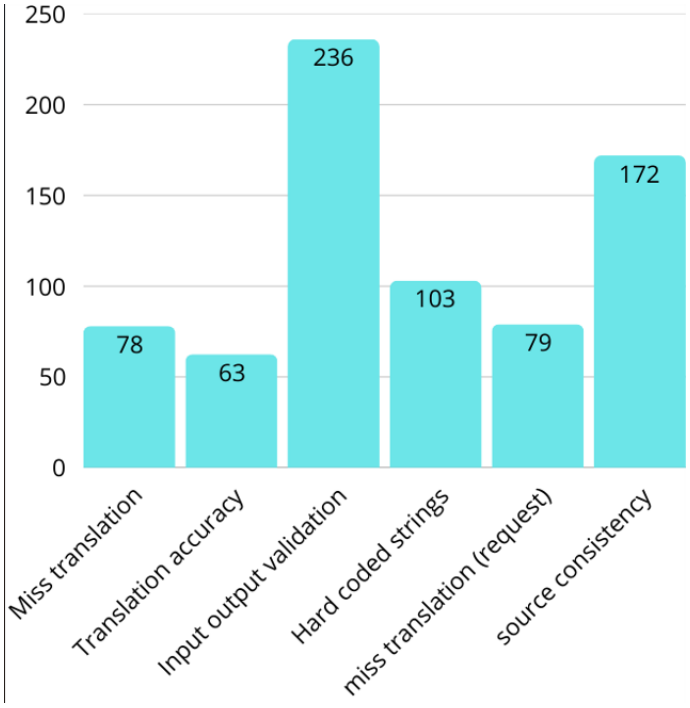


Figura 27. Gráfico de tempo médio dos erros mais comun

4.5. Erros ao longo do tempo

Outra dado que pode ser analisado é o número de issues de cada erro em relação ao tempo, ou seja, quantos erros tiveram no começo do projeto e como eles se comportaram ao longo do tempo. Esses dados mostrados na figura 12 mostram apenas os erros que tiveram mais discrepancia com o tempo, todos os outros ficaram em níveis baixos de erros e foram bem lineares ao longo do tempo que foi recortado para o trabalho. Esse recorte abrange desde 2016 onde parte do projetos já estavam encaminhados e outra parte ainda estavam pra começar e termina no ano de 2024.

Primeiramente, é possivel notar que o número cresce bastante a partir de 2019 já que vários dos projetos selecionados para as análise começam entre 2017 e 2019 e todos eles tiveram um apice por um tempo que naturalmente diminui ao longo que as issues são resolvidas.

Também é notável que os erros *Miss translation* e *Translation accuracy* são os erros que mais crescem ao longo do tempo, isso acontece por conta dos projetos que quando novos sempre criam novas interfaces e atualizam seu código gerando mais texto que acabam precisando de tradução ou polimento, além disso pelos projetos serem open source o número de linguas que são adicionadas é bem elevado fazendo com que erros dessa categoria aumentem substancialmente.

Além disso, o valor de bugs gerados por localização aumentaram junto com o pico geral, mas acabaram se mantando estável, muitos erros passam muito tempo tempo sem ser corrijido e por isso a linha se mantem pouco variável ao longo dos anos.

Por fim, o número de issues que reportam texto sem localização varia devido a criação e atualização dos projetos, mas sempre tendem a cair, mesmo que lentamente, ao longo dos

anos, porém como já citado anteriormente demora mais do que deveria, por serem issues menos urgentes para o que os contribuidores e donos dos projetos precisam trabalhar.

Um ponto chave do gráfico é o pico de issues que existem entre 2022 e 2023 em relação as issues, principalmente de tradução, mas esse número se dá aos projetos que possuem muita issues como VSCode ou Notepad++ terem passado por uma grande atualização tanto de interface quanto da maneira que o código é localizado como no caso do VSCode, por isso muito trabalho foi feito para ajustar os prolemas vindos dessa grande atualização que no ano seguinte mostra uma queda brusca de problemas relacionados a localização que tendem a dimuir ao longo dos anos

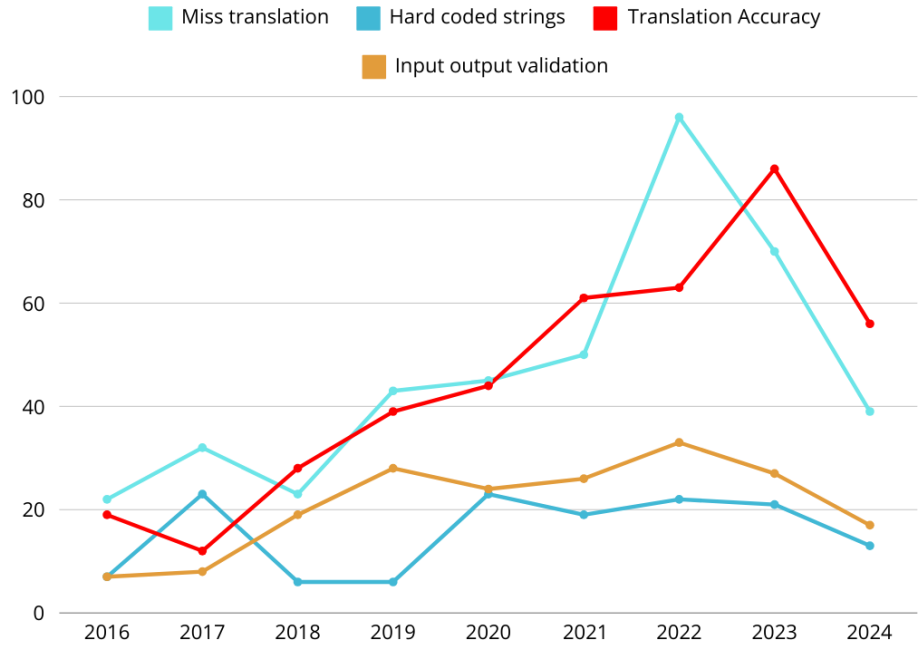


Figura 28. Gráfico de erros ao longo de tempo

5. Conclusão

Os projetos Open source abriu portas no mundo da programação, muitos projetos adotam esse modo descentralizado, que além de ser barato para os criadores, eles conseguem se manter de maneira estável e ainda ajudam seus próprios usuários. Para o contexo de localização e internacionalização, adotar esse estilo de projeto potencializa muito a distâcia que projetos internacionalizados podem atingir tendo uma maior facilidade na hora de localizar projetos com pessoas de diversas partes do mundo que falam a língua nativamente.

Porém, o primeiro problema encontrado é a organização das issues, em todos os projetos tivemos existiam issues que estavam mal relacionadas a label de localização em questão, muitos contribuidores atribuem problemas relacionados código e documentação a problemas de localização e até a dúvidas relacionadas a código entram nessas labels, além disso muitas issues são feitas sem detalhamento, em relação ao tipo do erro e quem qual locale ela ocorre.

Ainda, é notado que quando o projeto precisa de manuntenção para essas questões os contribuidores acabam deixando a prioridade bem baixa, fazendo com que esses pequenos erros de tradução, coerência, coesão e interface passem para o usuário final e quando essas

issues são percebidas e abertas no Github, elas acabam demorando muito mais do que deveriam para serem resolvidas, mesmo que parte das alterações envolvam poucas linhas de código.

Alguns dos projetos, adotam algumas estratégias para contornar problemas de localização usando ferramentas *Globalization Management Systems* (GMS), como o *Trasifex* que são usados para diminuir o trabalho feito pelo contribuidores na hora de criar e fazer as traduções, a unica limitação é o número de línguas a qual o GMS usado atinge, mas mesmo assim isso ajuda a manter os texto sempre bem escritos nas suas respectivas línguas e diminui o trabalho dos contribuidores de localizar o projeto

Para trabalhos futuros, seria interessante coletar mais projetos e issues para aumentar os números da analise e descobrir outros problemas que atingem os projetos, quais outras línguas também são afetadas e se as porcentagens se mantem as mesmas nesse novo contexto, adicionando novas nomeclaturas na taxonomia abragindo assim ainda mais issues para serem analisadas dentro do escopo do trabalho.

Além disso, seria interessante outras análises de outros pontos de vista, refazendo a planilha do zero com mais participantes que analisariam com base na sua interpretação e compariam seus pontos vistas para coletar os dados e analisar com mais precisão os problemas de internacionalização e localização.

Referências

ARNAUT, S. (2020). *Automated Internationalization and Localization Testing of GUIs/*. Sinan Arnaut.

AWWAD, Aiman M. Ayyal; SLANY, W. (2016). Automated bidirectional languages localization testing for android apps with rich gui. *Mobile Information Systems*.

COSENTINO, Valerio; IZQUIERDO, J. L. C. C. J. (2017). A systematic mapping study of software development with github. *Ieee access*, pages 7173–7192.

HAU, Elvis; APAR´ICIO, M. (2008). Software internationalization and localization in web based erp. *Proceedings of the 26th annual ACM international conference on Design of communication*.

MOMBACH, T. O. e. a. (2019). A comparative study of apis for querying github data.

YANG, Z. e. a. (2024). Exploring and unleashing the power of large language models in automated code translation. In *Proceedings of the ACM on Software Engineering*. n. FSE.