DUNNE, Keiran J. Localiação. *Belas Infiéis*, Brasília, v. 9, n. 4, p. 249-270, jul./set., 2020. e-ISSN: 2316-6614.
DOI: 10.26512/belasinfieis.v9.n4.2020.26401
Recebido: 30/07/2019 Aceito: 11/03/2020 Publicado: 07/2020

LOCALIZAÇÃO

LOCALIZATION

Keiran J. DUNNE
Kent State University
Department of Modern and Classical Language Studies
Kent, Ohio, Estados Unidos da América.
orcid.org/0000-0002-1892-1161
kdunne@kent.edu

Traduzido por*

Carolina Flávia de HENRIQUE
Pesquisadora autônoma
Uberlândia, Minas Gerais, Brasil
orcid.org/0000-0002-1892-1161
carolflavia97@hotmail.com

Jahynne Martins SALVADOR
Pesquisadora autônoma
Uberlândia, Minas Gerais, Brasil
orcid.org/0000-0002-7537-2891
jahynne@gmail.com

Marileide Dias ESQUEDA
Universidade Federal de Uberlândia
Instituto de Letras e Linguística
Bacharelado em Tradução
Uberlândia, Minas Gerais, Brasil
orcid.org/0000-0002-6941-7926
marileide.esqueda@ufu.br

Revisão técnica de

Isabel Borges Ribeiro de Assunção MACHADO
Universidade Federal de Uberlândia
Uberlândia, Minas Gerais, Brasil
orcid.org/0000-0002-0042-7878
isabelmachado97@outlook.com.br

Resumo: Localização é um hiperônimo que se refere ao processo através do qual conteúdos e produtos digitais desenvolvidos em determinada localidade são adaptados para uso e comercialização em outras localidades. Apesar de o termo "localização" ser utilizado desde o começo da década de 1980, sua definição ainda permanece confusa. Para entender o que é localização, é necessário considerar quando, como e por que surgiu, o que mudou ao longo dos anos e sua relação com a tradução e a internacionalização. Nesse contexto, este texto examina a localização e sua evolução dos anos de 1980 até o presente.
Palavras-chave: Internacionalização. Localização. Tradução. Engenharia de software.

Abstract*: Localization is an umbrella term that refers to the processes whereby digital content and products developed in one locale are adapted for sale and use in one or more other locales. Although the term 'localization' has been in use since the early 1980s, confusion persists as to what exactly it means. To understand localization, it is necessary to consider when, why and how it arose, the ways it has changed over time, and its relationship to translation and internationalization. This paper examines localization and its evolution from the 1980s to date.*
Keywords*: Internationalization. Localization. Translation. Software engineering.*

Este é um artigo em acesso aberto distribuído nos termos da *Licença Creative Commons* Atribuição que permite o uso irrestrito, a distribuição e reprodução em qualquer meio desde que o artigo original seja devidamente citado.

This is an Open Access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original article is properly cited.

LOCALIZAÇÃO

Localização é um hiperônimo que se refere ao processo através do qual conteúdos e produtos digitais desenvolvidos em determinada localidade são adaptados para uso e comercialização em outras. Apesar de o termo "localização" ser utilizado desde o começo da década de 1980, sua definição ainda permanece confusa. Para entender o que é localização, é necessário considerar quando, como e por que surgiu, o que mudou ao longo dos anos e sua relação com a tradução e a internacionalização. Desse modo, este artigo examina a localização e sua evolução dos anos de 1980 até o presente.

A prática da tradução manteve-se relativamente inalterada desde o surgimento da escrita até a comoditização dos computadores pessoais, sendo que o advento da comercialização em massa de *softwares* inaugurou a revolução digital nos anos de 1980. Com o aumento do número de computadores em residências e em ambientes de trabalho, "os usuários comuns não eram mais programadores profissionais, engenheiros de *software* e *hardware*" (UREN; HOWARD; PERINOTTI, 1993, p. ix). As empresas de *software* dos Estados Unidos perceberam de imediato que, desenvolvendo produtos tais como programas de planilhas e processadores de texto, usuários comuns poderiam utilizá-los em contextos profissionais ou para lazer, e, dessa forma, a comercialização poderia alcançar um mercado ainda mais amplo. Almejar o mercado de usuários comuns, no entanto, em vez de profissionais da computação, trouxe desafios.

> Enquanto os profissionais mais experientes tinham se tornado *experts* em detectar falhas e solucioná-las, os novos usuários esperavam, e até mesmo exigiam, que o *software* que eles adquiriam operasse exatamente como descrito no manual. Já não se tolerava mais o mau funcionamento de um *software*. (UREN; HOWARD; PERINOTTI, 1993, p. x)

Dessa forma, as publicadoras de *software*[1] concentraram-se no aprimoramento da confiabilidade e usabilidade para desenvolver esse amplo mercado embrionário.

As empresas de *software* dos Estados Unidos logo ampliaram o escopo de suas estratégias de *marketing* para além do mercado doméstico, para visar usuários internacionais. A expansão a mercados internacionais fez com que as publicadoras de *software* desenvolvessem produtos em outras línguas além do inglês. "Para que um *software* pudesse ter grande aceitação de mercado em uma região que não falava inglês, era necessário convertê-lo para que os usuários vissem o produto em sua própria língua e com sólida inserção em sua própria cultura" (UREN; HOWARD; PERINOTTI, 1993, p. x). As publicadoras de *software* consideravam que

a adaptação de seus produtos para mercados internacionais era só uma questão de "tradução". Consequentemente, as primeiras tentativas de adaptar um *software* para usuários internacionais foram descritas como "uma tradução no computador para o computador" (VAN DER MEER, 1995). No entanto, logo ficou claro aos profissionais que esse trabalho estava "relacionado com a tradução, mas ao mesmo tempo era algo diferente e mais complexo que ela" (LIEU, 1997). Essa empreitada, portanto, não estava limitada apenas à tradução do texto presente na interface de usuário, mas envolvia as exigências do mercado-alvo com relação à representação culturalmente dependente dos dados como[2]:

- conjuntos de caracteres e glifos para a representação de vários sistemas de escrita;
- codificação que permite o armazenamento, recuperação e manipulação de dados multilíngues;
- comparação de texto, busca e classificação (agrupamento);
- quebra de linhas e palavras;
- calendários (*e.g.*, budista, alexandrino, gregoriano, hebraico, islâmico, japonês, juliano, chinês e juche);
- formatos de data (mês/dia/ano, dia/mês/ano, ano/mês/dia etc. – por exemplo, 5/11/2014 lê-se "maio 11, 2014" nos Estados Unidos, mas "5 de novembro de 2014" na Itália);
- formatos de hora (o sistema de 12 horas e o de 24 horas; uso de AM e PM);
- formato de números, agrupamentos de dígitos e separadores decimais (ponto ou vírgula);
- tamanhos de papel (A3, A4, Ofício e Carta);
- unidades de medida (métrico x imperial).

Na engenharia de *software*, essas exigências do mercado local são denominadas pelo hiperônimo "localidades"[3] (*locales*). Localidades são designadas como pares de língua-país. Desse modo, "francês-Canadá é uma localidade, ao passo que francês-França é outra" (CADIEUX; ESSELINK, 2002). A necessidade de levar em consideração não só a tradução, mas também a localidade, explica como e por que o processo de adaptação de *software* para mercados internacionais começou a ser conhecido como "localização" no início da década de 1980. A magnitude dessa nova atividade expandiu-se tão rápido que, em menos de uma década,

a localização passou a ser compreendida como uma indústria própria, tal como demonstrado pela criação da *Localization Industry Standards Association,* em 1990 (LOMMEL, 2007, p. 7).

Os custos de adaptação de *softwares* para outras localidades pareciam reduzidos se comparados aos grandes mercados internacionais e possíveis lucros que os produtos localizados poderiam alcançar. As empresas de *software* abordaram a localização de diversas formas: algumas optaram por utilizar equipes internas; algumas terceirizaram o processo para provedores especializados; em alguns casos, outras atribuíram a responsabilidade a subsidiárias ou publicadoras no país (ESSELINK, 2003b, p. 4). Apesar das diferenças entre tais abordagens, todas elas partilhavam a seguinte característica fundamental: em todos os casos, a localização era realizada à parte e subsequente ao desenvolvimento do produto original. "Essa separação entre o desenvolvimento e a localização mostrou-se problemática de diversas formas", conforme observa Esselink (2003b, p. 4).

Primeiramente, o *software* fornecido às equipes de localização muitas vezes não poderia ser localizado, devido à falta de determinados recursos fundamentais, como a capacidade de exibição do sistema de escrita da língua-alvo. Nesses casos, as equipes de localização devolviam o *software* aos seus desenvolvedores para que fossem implementados os recursos necessários, como suporte para a exibição de idiomas asiáticos ou sistemas de escrita da direita para a esquerda para idiomas como o árabe e o hebraico. Em segundo lugar, os textos a serem traduzidos normalmente encontravam-se incorporados ao código-fonte do *software*. Identificá-los e localizá-los era um trabalho penoso para as equipes que não haviam participado do desenvolvimento do *software* (cf. Figura 1). Por fim, e talvez o ponto mais crítico a ser observado é que a localização exigia mudanças diretas no código-fonte do *software*. Como trabalhar diretamente no código-fonte causava problemas, passou a ser importante saber que ele é a matéria-prima a partir da qual a cópia em execução do programa é criada. Em outras palavras, o código-fonte deve ser compilado ou construído em um arquivo executável legível por máquina (binário), que por sua vez deve ser testado (e corrigido, caso algum erro seja encontrado) antes de o *software* ser liberado para uso (cf. Figura 2).

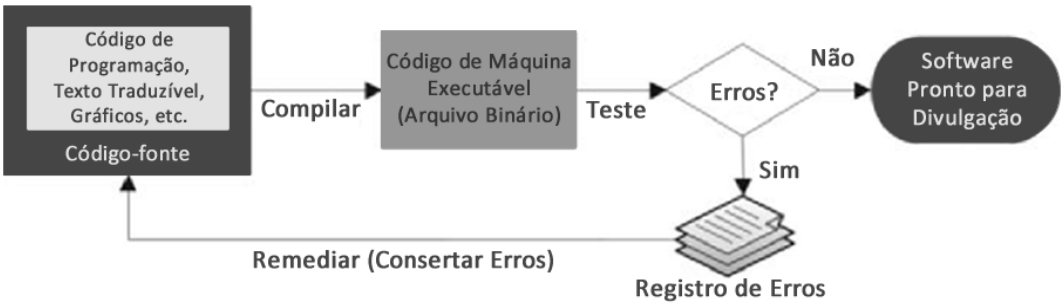
DUNNE, Keiran J. Localiação. *Belas Infiéis*, Brasília, v. 9, n. 4, p. 249-270, jul./set., 2020. e-ISSN: 2316-6614.
DOI: 10.26512/belasinfieis.v9.n4.2020.26401

Figura1 – Representação do código-fonte da caixa de diálogo exibida na Figura 5 (a)

```
IDD_PEN_WIDTHS DIALOG  0, 0, 203, 65
STYLE DS_SETFONT | DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
    WS_SYSMENU
CAPTION "Pen Widths"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,148,7,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,148,24,50,14
    PUSHBUTTON      "Default",IDC_DEFAULT_PEN_WIDTHS,148,41,50,14
    LTEXT           "Thin Pen Width:",IDC_STATIC,10,12,70,10
    LTEXT           "Thick Pen Width:",IDC_STATIC,10,33,70,10
    EDITTEXT        IDC_THIN_PEN_WIDTH,86,12,40,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_THICK_PEN_WIDTH,86,33,40,13,ES_AUTOHSCROLL
END
```

Fonte: elaborado por Dunne (2015).

Figura 2 – Devido ao fato de que o código-fonte precisa ser compilado e testado sempre que for modificado, localizar diretamente no código é um trabalho bastante árduo



Fonte: traduzido de Dunne (2015).

Para aqueles que não são programadores, identificar os textos a serem traduzidos não é um trabalho fácil. No exemplo da Figura 1, os itens entre aspas são traduzíveis, com exceção do nome da fonte padrão (*MS Sans Serif*). Cada grupo de quatro dígitos separados por vírgulas representa a formatação.

Trabalhar diretamente no código-fonte ocasionou profundas ramificações para a localização. Como resultado, a adaptação de *softwares* para outras localidades não implicou apenas meras mudanças nas versões já compiladas, testadas e corrigidas de programas que já haviam sido comercializados no mercado interno. Em vez disso, a localização de dado programa exigia que um conjunto separado de código-fonte fosse mantido e que um executável

diferente fosse compilado, testado e corrigido para cada localidade. Consequentemente, criar várias versões localizadas de um programa exigia que a publicadora mantivesse uma versão adicional do conjunto de código-fonte: uma para cada localidade e mais uma para o mercado doméstico. Além disso, cada conjunto de código-fonte tinha que ser localizado, compilado, testado, corrigido, atualizado e gerenciado separadamente (LUONG *et al.*, 1995, p. 3). Por exemplo, uma publicadora dos Estados Unidos que quisesse comercializar seu produto em três localidades internacionais, como alemão-Alemanha, francês-França e japonês-Japão, precisaria gerenciar paralelamente quatro versões diferentes do código-fonte – um conjunto para a localidade doméstica inglês-Estados Unidos mais um conjunto para cada uma das três localidades referenciadas (cf. Figura 3). O processo de compilar e testar um produto localizado de forma distinta das versões de origem é chamado de engenharia de localização (ESSELINK, 2002).

Criar, gerenciar e prover manutenção de várias versões localizadas em paralelo mostrou ser um processo caro e demorado. Testar e corrigir *softwares* já era um trabalho intenso e dispendioso antes mesmo do processo de localização. Um trabalho inaugural sobre os custos da engenharia de *software* de Boehm (1981) mostrou que "erros que não são corrigidos tornam-se muito mais caros a cada fase em que eles não são solucionados" (p. 8). Os valores relacionados à correção de erros aumentaram ainda mais devido à abordagem de localização *post hoc*, na qual a adaptação de um *software* para outras localidades – e consequentemente a detecção de erros – começava, apenas, após desenvolvidas as versões do mercado interno. Esse problema no aumento dos custos estava relacionado ao gerenciamento de um código-fonte distinto para cada localidade-alvo, já que o problema descoberto em um dos conjuntos deveria ser corrigido em todos os outros. Esselink (2002, p. 4) aponta que a engenharia de localização envolvia tradicionalmente "um número expressivo de correção de erros". Era de se esperar, portanto, que a localização logo se revelasse também complexa (ESSELINK, 2000b). Dito isso, a maior parte dos problemas impostos pelos primeiros esforços da localização surgia de uma única causa: o planejamento ineficaz do código-fonte do *software* para as múltiplas localidades[4].

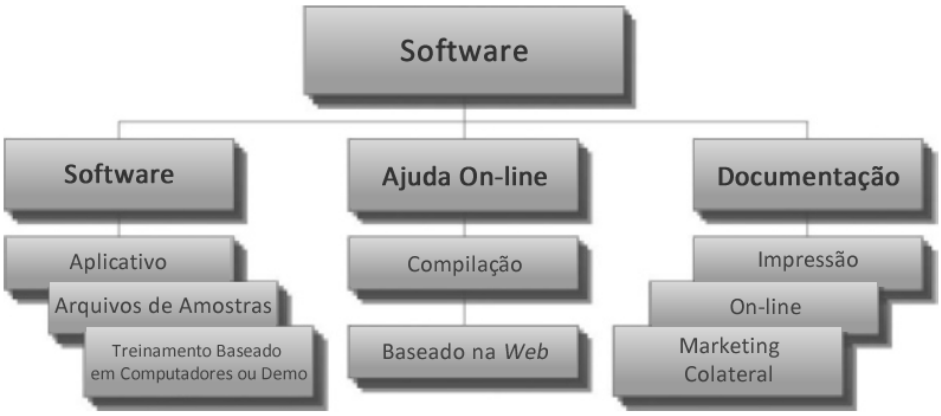
DUNNE, Keiran J. Localiação. *Belas Infiéis*, Brasília, v. 9, n. 4, p. 249-270, jul./set., 2020. e-ISSN: 2316-6614.
DOI: 10.26512/belasinfieis.v9.n4.2020.26401

Figura 3 – Quando a localização é realizada no código-fonte, é necessário manter um conjunto do código separado para cada localidade-alvo mais um conjunto para o mercado doméstico

Fonte: traduzido de Dunnes (2015).

A maioria das empresas de *software* e *hardware* que fizeram investimentos em mercados internacionais concluíram que localização e tradução não eram parte integrante de seus negócios. Esselink (2000a, p. 5) ressalta que "o aumento e a complexidade de projetos de localização logo forçaram as empresas a adotarem um modelo terceirizado. A maioria das publicadoras simplesmente não tinha tempo, conhecimento ou recursos para gerenciar traduções multilíngues ou projetos de localização". Como resultado, um grande número de empresas decidiu que seria mais eficiente terceirizar a adaptação de seus produtos para mercados internacionais a fornecedores externos de serviços linguísticos como parte do projeto de trabalho. Além da adaptação do *software*, um típico projeto de localização também envolvia tradução e/ou adaptação de vários outros componentes, como arquivos de amostra, *demos* e tutoriais, sistemas de ajuda, documentos de usuário *on-line* e impressos, além de garantia de *marketing* (cf. Figura 4). O fato de esses componentes terem sido criados em uma variedade de formatos digitais, alguns dos quais precisavam ser compilados e testados antes do lançamento, significava que a localização envolvia novas formas de trabalho além da tradução convencional. Nisso estavam inclusos materiais de ajuda *on-line*, conversão de documentos para diferentes formatos, criação e gerenciamento de memória de tradução, bem como gerenciamento de projetos (ESSELINK, 2000b; ESSELINK 2003a, p. 69; DUNNE; DUNNE, 2001). A localização exigia, portanto, que tradutores tivessem grandes habilidades instrumentais e técnicas, além do conhecimento e domínio de tradução. "Ao longo da década de 1990, a indústria da localização tentou transformar tradutores em semiengenheiros", relembra Esselink (2003b, p. 7).

Figura 4 – O escopo de um projeto tradicional de localização de *software* pode abranger vários componentes além do aplicativo de *software* em si



Fonte: traduzido de Esselink (2000a, p. 10).

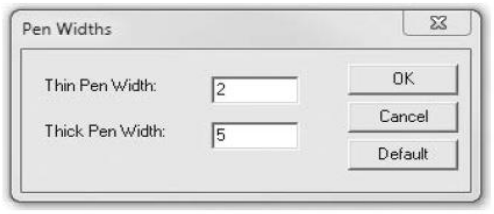
A terceirização transferiu os desafios de gerenciamento de projetos complexos de localização multilíngue a fornecedores externos, mas não abordou o problema fundamental de duplicação de esforços exigidos para gerenciamento paralelo de múltiplos conjuntos de código-fonte. No fim dos anos de 1980 e começo os anos de 1990, diante do desafio de controlar a complexidade e o custo do processo de localização, desenvolvedores começaram a perceber que "dadas etapas poderiam ser executadas com antecedência para tornar a localização mais fácil: separar as linhas de texto traduzíveis do código executável, por exemplo. Isso se refere à internacionalização ou à preparação para localização" (CADIEUX; ESSELINK, 2002). A internacionalização é um processo de engenharia que precede a localização e implica a separação de "todos os componentes que são cultural e linguisticamente sensíveis dos *softwares* [...] do núcleo da aplicação" (HALL, 1999, p. 298). Na prática, o escopo da internacionalização está tipicamente restrito aos conteúdos da interface de usuário que são dependentes nos quesitos linguístico e cultural e que podem requerer adaptação, designada coletivamente por meio do hiperônimo "recursos". Quando uma parte de um *software* é internacionalizada corretamente, "não há nem código de programação nos recursos, nem textos traduzíveis no código do programa" (UREN; HOWARD; PERINOTTI, 1993, p. 60). Os recursos em um aplicativo de *software* de *desktop* típico podem conter:

- aceleradores: atalhos de teclado que habilitam execução direta de comando. Aceleradores são tipicamente associados a uma tecla de função ou uma combinação da tecla Ctrl mais uma letra específica do teclado. Por exemplo, pressionar a tecla

de função F1 em um aplicativo típico do Windows abre a “Ajuda”, ao passo que pressionar Ctrl+C executa o comando “Copiar”;

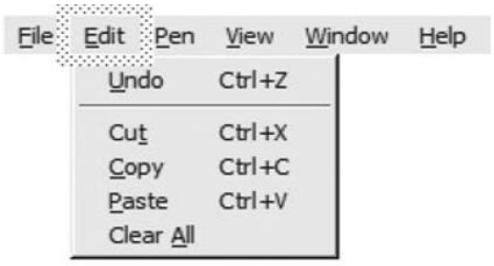
- caixas de diálogo: janelas secundárias que permitem que o usuário execute um comando e/ou lhe solicitam que forneça informações adicionais (cf. Figura 5a). Exemplos comuns de caixas de diálogo incluem “Salvar Como” e “Imprimir”. Os recursos da caixa de diálogo também contêm as coordenadas que configuram o leiaute e a exibição da interface de usuário (cf. Figura 1);
- ícones: imagens que simbolizam e fornecem atalhos clicáveis para programas, arquivos e dispositivos (cf. Figura 5b);
- menus: lista de opções ou comandos que aparecem no topo da janela principal do programa (cf. Figura 5c). Menus secundários, chamados de “menu contextual” ou “menu *popup*”, que aparecem quando o usuário clica com o botão direito do *mouse*;
- grupos de *strings*: *string* é a abreviação de “*string* [sequência] de caracteres” e designa qualquer texto armazenado e manipulado como um grupo. As *strings* incluem as legendas de botões, títulos das caixas de diálogo, mensagens de erro, itens do menu, dicas de ferramenta e assim por diante. O menu e as *strings* de caixa de diálogo podem ser representados visualmente em um editor que contenha a funcionalidade WYSIWYG (*what you see is what you get*; “o que você vê é o que você tem”) durante o processo de localização, ao passo que os grupos de *strings* normalmente não (cf. Figura 5d);
- barra de ferramentas: gráficos *raster* que contêm imagens de botão da barra de ferramentas, normalmente em formato bitmap (*.bmp) ou PNG – Portable Network Graphics (*.png), (cf. Figura 5e).

Figura 5 – Recursos típicos encontrados em um *software* incluem: (a) uma ou mais caixas de diálogo; (b) um ícone do programa e um ícone de documento (imagens da esquerda e da direita, respectivamente); (c) um ou mais menus; (d) uma *string*; e (e) uma ou mais barras de ferramentas[5]. Cf. também Figuras 8 e 9



(a)

(b)



(c)

Value	Caption
57632	Erase the selection\nErase
57633	Clears the drawing
57634	Copy the selection and put it on the Clipboard\nCopy
57635	Cut the selection and put it on the Clipboard\nCut
57636	Find the specified text\nFind
57637	Insert Clipboard contents\nPaste
57640	Repeat the last action\nRepeat

(d)

(e)

Fonte: elaborado por Dunnes (2015).

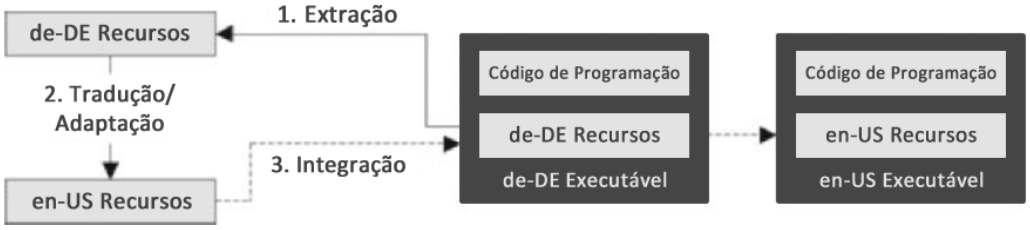
A criação de uma forma padronizada para representar o material de interface de usuário que é dependente da cultura e armazená-lo de forma independente do código funcional do programa facilitou muito o processo de localização. Não era mais necessário modificar o

DUNNE, Keiran J. Localiação. *Belas Infiéis*, Brasília, v. 9, n. 4, p. 249-270, jul./set., 2020. e-ISSN: 2316-6614.
DOI: 10.26512/belasinfieis.v9.n4.2020.26401

código-fonte ou compilar, testar e corrigir os erros de cada versão do programa de forma separada (LUONG *et al.*, 1995, p. 3). Em vez disso, a equipe de desenvolvimento poderia com facilidade extrair os recursos do arquivo executável binário, fornecê-los a uma equipe de localização que traduziria o texto e faria qualquer outra modificação necessária e, em seguida, retornaria os recursos-alvo para os desenvolvedores, que os integrariam em cópias do arquivo executável binário para criar a versão ou versões-alvo necessária(s) (cf. Figura 6). Esse processo de extração-adaptação-integração é uma das características que define a localização.

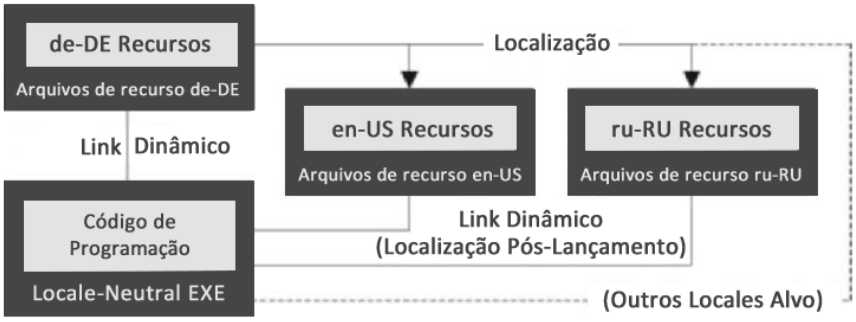
Ao permitir o uso de um único conjunto de código-fonte para auxiliar várias localidades-alvo, a internacionalização diminuiu o esforço e o custo associados à localização, e aumentou a velocidade e a precisão com a qual pode ser realizada (SCHMIT, 2007, p. 51). Logo ficou claro às empresas desenvolvedoras de *software* que, além de incorporar recursos em um arquivo executável de um programa e vinculá-los diretamente ao código do aplicativo, elas também podiam externalizar os recursos, armazená-los em arquivos dedicados chamados bibliotecas-satélites (*satellite assembly*) e conectar dinamicamente esse arquivo de recurso externo ao aplicativo. Para localizar um aplicativo desenvolvido usando essa estratégia de internacionalização, é necessário que se crie uma versão localizada do(s) arquivo(s) de recurso para cada localidade-alvo. Por exemplo, uma publicadora que criou um programa para a localidade alemão-Alemanha e quisesse comercializar versões para os Estados Unidos e para a República Popular da China deveria criar uma versão localizada do conjunto de recursos para cada localidade (cf. Figura 7).

Figura 6 – A internacionalização permite a separação lógica de conteúdos culturalmente dependentes da interface de usuário do núcleo funcional do programa, transformando a localização em um mais processo simples de substituição de recursos



Fonte: traduzido de Dunne (2015).

Figura 7 – Externalizar recursos, armazená-los em arquivos dedicados e vinculá-los de forma dinâmica a um núcleo de programa de localidade-neutra é a culminância lógica de estratégias de internacionalização de *software*



Fonte: traduzido de Dunne (2015).

O surgimento da internacionalização de *software* foi facilitado pela ampla mudança de programação procedural e estruturada para programação orientada a objetos nos anos 1980 e 1990. As linguagens de programação procedurais e estruturadas, que predominavam durante o período de 1960 e 1970, eram adequadas para aplicativos autônomos relativamente simples e pequenos. No entanto, conforme os aplicativos se expandiam em tamanho, complexidade e níveis de interação com outros sistemas, as linguagens procedurais e estruturadas começaram a mostrar suas limitações (CLARK, 2013, p. 2-5). Quanto maior era o programa, mais difícil era mantê-lo e modificar um aspecto da funcionalidade existente sem impactar negativamente o sistema como um todo. Os programadores precisavam de um entendimento abrangente de como um programa funcionava e não poderiam concentrar seus esforços em funções isoladas. Além disso, a ausência de formas de notação padronizadas para representar e codificar funções dificultou a portabilidade e reutilização do código-fonte do *software*, com o resultado de que programas eram, normalmente, construídos do zero.

A programação orientada a objetos resolveria esses problemas de forma efetiva. Na programação orientada a objetos, dados e funções que usam aqueles dados são agrupados e encapsulados em estruturas lógicas denominadas objetos. Os dados e funções encapsulados de objetos podem ser usados ou acessados por outras funções ou programas. A comunicação entre os objetos em um programa é realizada por meio de mensagens, e sua interface é definida pelas mensagens que se pode enviar e receber. Na programação orientada a objetos, o envio de uma mensagem para um objeto também é chamado de configuração de uma propriedade desse objeto. Objetos são definidos por classes, as quais determinam seu código, dados e mensagens que podem enviar e receber (ou seja, suas propriedades). Objetos específicos herdam todas as propriedades e funções da classe a que pertencem. A herança permite a criação de objetos e

subclasses “filhos”, que herdam todas as propriedades e funções da classe original do objeto “pai”. A herança facilita a manutenção, atualização e correção de erros do *software*, porque uma mudança feita em uma instância de um objeto é aplicada a todas as instâncias de objetos naquela classe. Objetos também podem ser reutilizados dentro e através de programas, o que simplifica o desenvolvimento de novos programas.

Os programas orientados a objeto não são escritos, mas sim desenhados em ambientes de desenvolvimento WYSIWYG integrados (*what you see is what you get*, “o que você vê é o que você tem”) usando uma variedade de objetos, incluindo menus, caixas de diálogo e controle de usuários, como botões de comando, caixas de checagem e etiquetas de texto. A partir da programação orientada a objetos, a internacionalização padroniza a representação, definição e armazenamento do inventário dos objetos de interface de usuário como classes de recurso. Consequentemente, a localização é propriamente entendida como a modificação das propriedades dos objetos. Por exemplo, traduzir a legenda de um botão de comando requer modificação da propriedade “Legenda” do objeto “Botão” (cf. Figura 8).

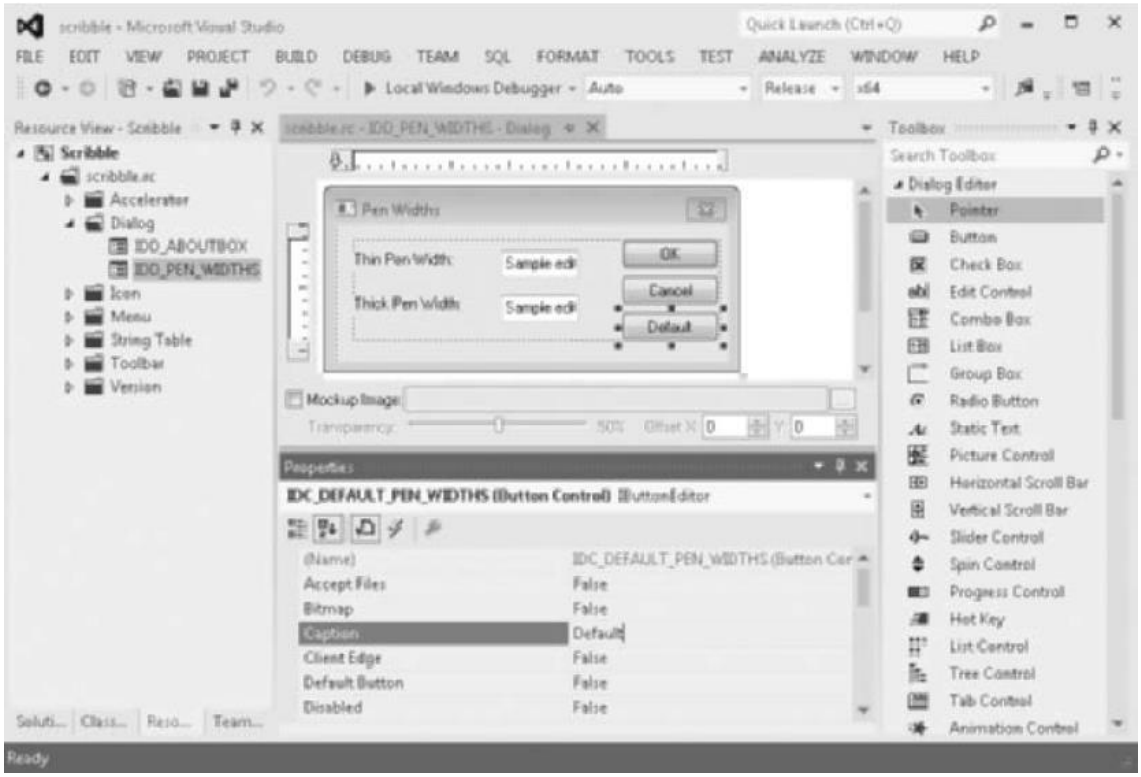
A internacionalização não só eliminou a necessidade de manter um conjunto de códigos-fonte para cada localidade, mas também esclareceu os respectivos papéis dos programadores, engenheiros e tradutores.

> A internacionalização permite que os programadores e engenheiros se centrem no código e os tradutores, na tradução. Isso significa que o *software*, com toda a sua lógica complexa, não tem que ser alterado só porque alguém quer acrescentar um novo idioma. Tudo o que se deve fazer é traduzir alguns arquivos. (UREN; HOWARD; PERINOTTI, 1993, p. 63)

De fato, a maioria dos aspectos da localidade dependentes de dados de armazenamento, recuperação, manipulação e apresentação pode ser, atualmente, gerenciada por meio de capacidades de internacionalização construídas no sistema operacional hospedeiro e/ou usando estrutura de desenvolvimento e ambientes que oferecem apoio sólido para a internacionalização, como o Java (DEITSCH; CZARNEKI, 2001) ou a estrutura do Microsoft.NET (SMITH-FERRIER, 2007). Se um programa tiver sido internacionalizado de maneira correta e todos os textos traduzíveis tiverem sido retirados do código-fonte, o localizador trabalhará somente com os arquivos de recurso e não poderá acessar ou modificar o código funcional do programa. Consequentemente, os componentes básicos da localização de *software* agora envolvem a tradução das *strings* nos menus, caixas de diálogos e grupos de *strings.* Caixas de diálogo e controles de usuário como botões e etiquetas podem demandar

redimensionamento para possibilitar a expansão ou redução das *strings* de tradução, dependendo das línguas de origem e destino envolvidas. As ferramentas visuais de localização facilitam essas tarefas, ao permitirem que localizadores vejam os recursos em contexto, possibilitando, também, que usem memória de tradução e bancos de dados terminológicos enquanto trabalham (cf. Figura 9).

Figura 8 – Uma interface de programa orientada a objetos é desenhada usando classes de objetos de controle de usuário (painel direito); objetos de interface composta são definidos e armazenados como recursos (painel esquerdo). A localização de um *software* orientado a objeto é entendida corretamente como uma modificação das propriedades de objetos, como a legenda do botão de comando Default ("Padrão") (painéis superior e inferior do meio). Essa figura ilustra a criação do aplicativo de amostra Scribble (cf. Figuras 1 e 5) no Microsoft® Visual Studio® 2012



Fonte: utilizado com permissão da Microsoft.

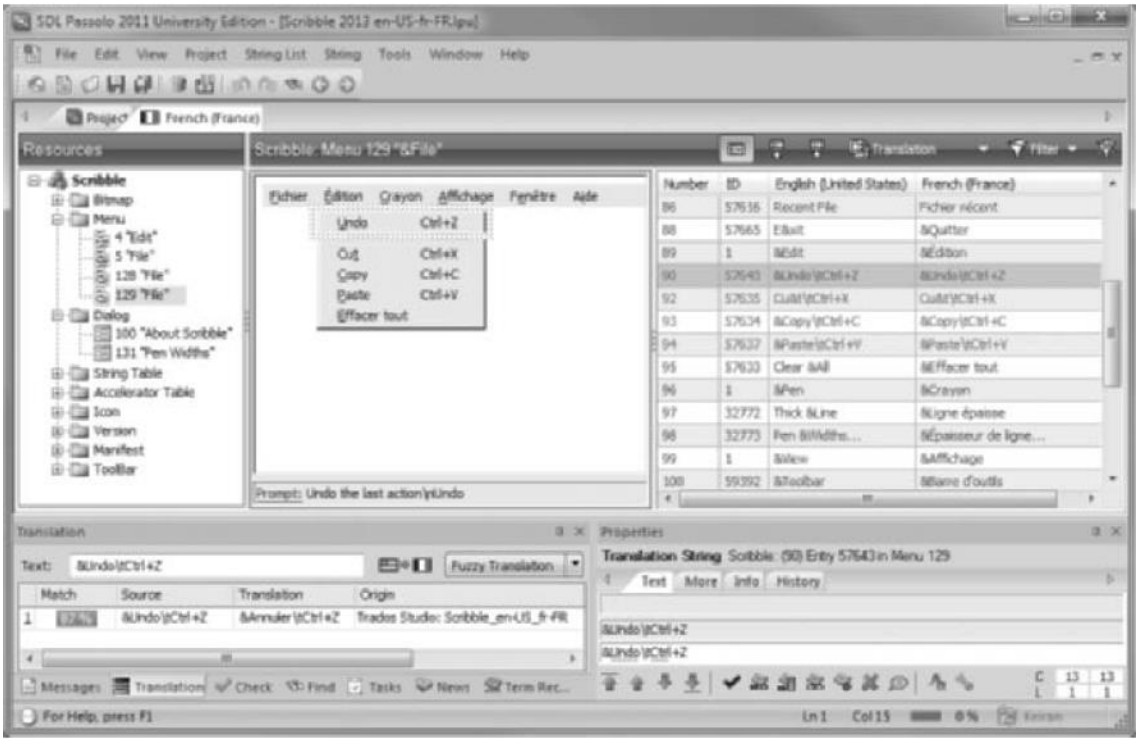
A internacionalização e o uso de arquivos de recursos-satélites permitem que as publicadoras de *software* abordem de maneira proativa, bem antes da localização, os problemas relacionados à localidade durante o processo de desenvolvimento do *software*. Nesse ponto, é necessário saber como – e talvez até se – a localização difere da tradução atualmente. O fato de a tradução das *strings* de textos compreender a maior parte do trabalho na prática atual sugere que o termo "localização" voltou a significar a mesma coisa, isto é, "tradução no computador, para computador". A indefinição dos limites entre tradução e localização também pode ser vista

DUNNE, Keiran J. Localiação. *Belas Infiéis*, Brasília, v. 9, n. 4, p. 249-270, jul./set., 2020. e-ISSN: 2316-6614.
DOI: 10.26512/belasinfieis.v9n4.2020.26401

como evidência de uma convergência desses processos, uma vez que autoria e publicação sofrem uma evolução semelhante à da localização de *software* (Esselink, 2003b).

A autoria e a publicação[6] (*authoring and publishing tools*) foram processos e profissões separadas até os anos 1980 e 1990, quando o advento das ferramentas digitais de autoria, como o processador de textos, transformou autores em editores digitais que eram não só capazes de criar conteúdos digitais, mas também de controlar a maneira de sua apresentação (ROCKLEY; KOSTUR; MANNING, 2003, p. 165). No entanto, a abordagem baseada em *desktop* – e documentos (*desktop-based authoring and publishing*) – para autoria e publicação impediu a reutilização de conteúdo, da mesma maneira que os esforços iniciais da localização, que exigiam modificação no código-fonte.

Figura 9 – A localização de um aplicativo de amostra chamado Scribble usando uma ferramenta de localização visual. O painel esquerdo exibe a árvore de recurso, o painel do meio exibe o recurso selecionado em modo WYSIWYG, e o painel direito exibe a cadeia de origem e as *strings* de destino em formato tabular



Fonte: SDL Passolo 2011.

A adaptação de conteúdo armazenado em documentos criados utilizando processadores de textos e outros *softwares* tradicionais é geralmente um processo manual e intenso. Uma regra geral é que digitadores (*technical communicators*) que usam o processador de textos e ferramentas autorais de documentação gastam a metade do tempo com a formatação de documentos (BARLETT, 1998). "Para reutilizar o conteúdo, os autores devem aplicar a

formatação apropriada para cada produção. Tirar e reaplicar a formatação é complicado e não é totalmente efetivo. A conversão de formatos sempre requer correção manual ou *scripts* complicados" (ROCKLEY *et al.*, 2003, p. 165). Outros esforços para reutilizar conteúdos complicados eram necessários para gerenciar várias versões de documentos de arquivos-fonte em versões paralelas. Os profissionais logo descobriram que o gerenciamento de arquivo e o controle de versão tornaram-se exponencialmente mais complexos à medida que o número de versões paralelas dos arquivos-fonte e o número de línguas de chegada aumentavam. Esses problemas eram agravados pela *web* e sua ampla adoção como uma plataforma de comunicação empresarial. A autoria e publicação baseadas em *desktop* não conseguiram acompanhar a velocidade das mudanças na *web* nem poderiam atender à demanda de conteúdo em uma gama, cada vez maior, de formatos para uso em uma variedade de dispositivos, desde computadores pessoais e *notebooks* a *palmtops*, *tablets* e *smartphones*.

Os desafios de se utilizarem conteúdos baseados em documentos foram muito semelhantes aos esforços iniciais da localização de *software*. Assim, talvez não seja surpreendente que as estratégias adotadas para facilitar a reutilização de conteúdos sejam semelhantes às estratégias desenvolvidas para facilitar a localização de *software*. Da mesma maneira que a internacionalização simplifica a localização, separando de forma lógica os aspectos linguísticos e culturais da interface de usuário do núcleo funcional de um programa, as estratégias de reutilização de conteúdo são baseadas na separação do conteúdo da apresentação. Essa abordagem é chamada de fonte única: "[f]onte única implica que há uma versão e um local únicos para o conteúdo; o conteúdo é escrito uma vez, armazenado em um local de fonte única e reutilizado várias vezes" (ROCKLEY *et al*., 2003, p. 15).

A implementação de fonte única envolve estratégias autorais baseadas em XML (SAVOUREL, 2001, p. 7; ROCKLEY *et al.*, 2003, p. 159-171). XML (*eXtensible Markup Language*) é a linguagem de metamarcação que fornece um mecanismo universal de aplicação independente para representar textos em um formato estruturado. A linguagem XML foi criada em resposta aos desafios de reutilização de conteúdos associados a publicações digitais de larga escala. Conforme nota veiculada no World Wide Web Consortium Press, de dezembro de 1997, que anunciava a publicação da versão 1.0 XML como uma prosposta de recomendação,

> a XML destina-se principalmente a atender aos requisitos dos provedores de conteúdo da *web* em grande escala para marcação específica do setor, intercâmbio neutro de dados entre provedores, publicação independente de mídia, *marketing* individualizado, gerenciamento de fluxo de trabalho em ambientes de autoria colaborativa e processamento de documentos da *web* por clientes inteligentes. (W3C, 1997)

Enquanto a HTML é uma linguagem de marcação de apresentação que especifica como o conteúdo deve ser exibido, a XML é uma linguagem de marcação semântica que especifica o que o conteúdo significa. Como a XML provê uma representação semântica e estruturada do conteúdo e não prioriza a apresentação dos aspectos do documento, a autoria do conteúdo em XML é geralmente chamada de autoria estruturada. A formatação da XML em documentos é especificada por diretivas de estilo armazenadas em arquivos separados e aplicados de forma dinâmica em resposta à demanda do usuário. Dessa maneira, o mesmo conteúdo pode ser processado e produzido em qualquer formato para o qual a organização possui um conjunto definido de regras, como as páginas de web (HTML), documentos em PDF, documentos do Word, bem como em Ajuda Eclipse, ajuda HTML, ajuda Java e WebHelp, para citar apenas alguns exemplos. Hoje, a fonte única de documentação técnica e procedural é frequentemente implementada usando a DITA (*Darwin Information Typing Architecture* – Arquitetura Darwin de Informação de Dígito) baseada em XML (BELLAMY; CARY; SCHLOTFELDT, 2012).

A implementação de uma fonte única, muitas vezes, também envolve o uso de sistema de gerenciamento de conteúdo (ROCKLEY *et al.*, 2003, p. 178-191), que são depósitos centralizados "designados a gerenciar fragmentos de informação" (genericamente conhecidos como "conteúdos"), não sendo maiores que dois parágrafos" (BIAU GIL; PYM, 2006, p. 11). Fragmentos de informação, como os tópicos DITA, são montados de maneira dinâmica em documentos em resposta aos pedidos dos usuários, tipicamente via interface na *web*. Já no caso da autoria baseada em XML, o conteúdo armazenado em um sistema de gerenciamento de conteúdo (*Content Management System* – CMS) pode, geralmente, ser produzido em vários formatos.

Fontes únicas, autoria estruturada e "fragmentação" podem ser entendidas como aplicações dos conceitos de orientação de objetos e internacionalização no campo da autoria e publicação. Uma vez escrita, determinada informação do objeto pode ser reutilizada sistematicamente; uma vez traduzida, a versão da língua-alvo daquela mesma informação também pode ser sistematicamente reutilizada. Por ser separado da forma, o conteúdo pode ser processado utilizando-se quantas diretivas de estilo diferentes forem necessárias para que seja publicado nos formatos de produção desejados (*e.g.*, impressão, ajuda, *web* e dispositivos móveis), sem que se tenha que modificar o conteúdo. Assim como a orientação de objetos e a internacionalização facilitaram a modularização e reutilização do *software*, as fontes únicas e fragmentações facilitam a modularização e a reutilização do conteúdo.

A tradução do conteúdo XML e de informações fragmentadas não é "localização" da forma como tem sido tradicionalmente compreendida, pois não envolve a modificação das propriedades dos objetos na interface de usuário de um *software*. Entretanto, "projetos de tradução de conteúdo são agora considerados como projetos de localização devido aos ambientes complexos em que o conteúdo é escrito, gerenciado, armazenado e publicado", destaca Esselink (2003b, p. 7). A complexidade já foi uma característica dos projetos de localização, mas atualmente também caracteriza os projetos de tradução em larga escala.

Em um nível mais básico, a complexidade da localização de *software* e da tradução de seu conteúdo dá-se amplamente pelo fato de que os tradutores e localizadores não trabalham com textos lineares, mas sim com partes e *strings* descontextualizadas. Trabalhar um texto sem contexto não só complica o processo de decisão tradutória, como indiscutivelmente põe em questão a própria possibilidade de entender o texto como um todo e o ato pragmático de comunicação do qual é um artefato ostensivo. "Ao entender um texto, o leitor não deve somente ser capaz de integrar informação dentro de frases, mas também fazer conexões ao longo das frases para formar uma representação coerente de um discurso", observa Rayner e Sereno (1994, p. 73). Entretanto, não é sempre possível para o tradutor fazer conexões ao longo das frases ao trabalhar com *strings* de *software*. "Devido à sua estrutura não linear e à falta de um fluxo narrativo, *softwares* não podem ser 'lidos' da mesma forma que documentos tradicionais" (DUNNE, 2009, p. 197). Isso também se aplica ao conteúdo em XML e aos fragmentos de informações. Em projetos de fonte única, o "documento" não existe até que seja criado de maneira dinâmica em resposta ao pedido do usuário (tipicamente do usuário final). De forma geral, a tradução de *strings* e de parte da informação pode parecer tecnologicamente mais fácil que a localização convencional, pois os tradutores não precisam compilar ou testar arquivos-alvo. Entretanto, a tradução dos formatos mencionados anteriormente é cognitivamente mais complexa, pois ler e compreender textos sem o contexto e "textos sem fim" (BIAU GIL; PYM, 2006, p. 11) requer que o tradutor construa uma situação-modelo de texto que não existe ainda. Em outras palavras, a indústria não está mais tentando transformar os tradutores em semiengenheiros, mas os tradutores ainda precisam entender a arquitetura dos componentes em que os resultados dos projetos de localização de *software* são criados, como os arquivos-fonte, tópicos de ajuda, tabelas de conteúdos e índices. Como Esselink (2003b) bem observa, "parece provável que, embora os tradutores possam e esperem se concentrar cada vez mais em suas tarefas linguísticas [...], os entraves da complexidade técnica também aumentarão" (2003b).

DUNNE, Keiran J. Localiação. *Belas Infiéis*, Brasília, v. 9, n. 4, p. 249-270, jul./set., 2020. e-ISSN: 2316-6614.
DOI: 10.26512/belasinfieis.v9.n4.2020.26401

REFERÊNCIAS

BARTLETT, P.G. The benefits of structured XML authoring for content management. *In*: Graphic Communications Association (U.S.); Organization For The Advancement Of Structured Information Systems (OASIS) (Ed.). *XML 98 Conference Proceedings*. Chicago, IL/New York: Graphic Communications Association, 1998. p. 15-18.

BELLAMY, Laura; CAREY, Michelle, SCHLOTFELDT, Jenifer. *DITA best practices*: A roadmap for writing, editing, and architecting in DITA. Upper Saddle River, NJ: IBM Press, 2012.

BIAU GIL, José R.; PYM, Anthony. Technology and translation (A pedagogical overview). *In*: PYM, Anthony, PEREKRESTENKO, Alexander, STARINK, Bram (Ed.). *Translation technology and its teaching*. Tarragona, Spain: Intercultural Studies Group, 2006. p. 5-19.

BOEHM, Barry W. *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

CADIEUX, Pierre; ESSELINK, Bert. GILT: Globalization, Internationalization, Localization, Translation. *Globalization Insider*, n. 11, [s.p.], 2002.

CLARK, Dan. *Beginning C# object-oriented programming*. [s.e.]: Apress, 2013. doi: https://doi.org/10.1007/978-1-4302-4936-8

DEITSCH, Andy; CZARNECKI, David. *Java internationalization*. Sebastopol, CA: O'Reilly, 2001.

DUNNE, Keiran J. Putting the Cart behind the Horse: Rethinking localization quality management. *In*: DUNNE, Keiran J. (Ed.). *Perspectives on localization*. Amsterdam and Philadelphia: John Benjamins, 2006. p.1-11. doi: https://doi.org/10.1075/ata.xiii.08dun

DUNNE, Keiran J. Assessing software localization: For a valid approach. *In:* ANGELLI, Claudia V.; JACOBSON, Holly E. (Ed.). *Testing and assessment in translation and interpreting studies*. Amsterdam and Philadelphia: John Benjamins, 2009. p. 185-222. doi: https://doi.org/10.1075/ata.xiv.10dun

DUNNE, Keiran J.; DUNNE, Elena S. *Translation and localization project management*: The art of the possible. Amsterdam and Philadelphia: John Benjamins, 2011. https://doi.org/10.1075/ata.xvi

DUNNE, Keiran J. Localization. *In*: SIN-WAI, Chan. *The Routledge encyclopedia of translation technology*, 2015. p. 550-562.

ESSELINK, Bert. *A Practical guide to localization*. Amsterdam and Philadelphia: John Benjamins, 2000a.

ESSELINK, Bert. Translation versus localization. *Tranfree*, v. 10, [s.p.], 2000b.

ESSELINK, Bert. Localization engineering: The Dream Job? *Revista Tradumàtica*, n. 1, p. 2-5, 2002.

ESSELINK, Bert. Localisation and translation. *In*: SOMERS, Harold L. (Ed.*). Computers and translation*: A translator's guide. Amsterdam and Philadelphia: John Benjamins, 2003a. p. 67-86. doi: https://doi.org/10.1075/btl.35.08ess

ESSELINK, Bert. The evolution of localization. The guide to localization. *Supplement to MultiLingual Computing and Technology*, v. 14, n. 5, p. 4-7, 2003b.

GIAMMARRESI, Salvatore. Strategic views on localization project management: The importance of global product management and portfolio management. *In*: DUNNE, Keiran J.; DUNNE, Elena S. (Ed.). *Translation and localization project management*: The art of the possible. Amsterdam and Philadelphia: John Benjamins, 2011. p. 17-49. doi: https://doi.org/10.1075/ata.xvi.03gia

HALL, Patrick A.V. Software internationalization architectures. *In*: KERSTEN, Gregory E., MIKOLAJUK, Zbigniew; GAR-ON YEH, Anthony. (Ed.). *Decision support systems for sustainable development in developing countries*: A resource book of methods and applications. Boston, MA: Kluwer Academic Publishers, 1999. p. 291-304. doi: https://doi.org/10.1007/0-306-47542-1_16

LIEU, Tina. Software localization: The art of turning Japanese. *Computing Japan*, v. 4, n. 12, [s.p.], 1997.

LOMMEL, Arle. *The globalization industry primer*. Romainmôtier, Switzerland: Localization Industry Standards Association, 2007.

LUONG, Tuoc V.; LOK, James S. H.; TAYLOR, David J.; DRISCOLL, Kevin. **Internationalization**: Developing software for global markets. New York: John Wiley & Sons, 1995.

MARGULIES, Benson I. Your passport to proper internationalization. *Dr Dobb's*, [s.p.], 1 maio 2000.

RAYNER, Keith; SERENO, Sara C. Eye movements in reading: Psycholinguistic studies. *In*: GERNSBACHER, Morton A. (Ed.). *Handbook of psycholinguistics*. San Diego, CA: Academic Press, 1994. p. 57-81.

ROCKLEY, Ann; KOSTUR, Pamela; MANNING, Steve. *Managing enterprise content*: A unified content strategy. Indianapolis: New Riders, 2003.

SAVOUREL, Yves. *XML internationalization and localization*. Indianapolis: Sams, 2001. doi: https://doi.org/10.1162/109966202760152149

SCHMITZ, Klaus-Dirk. Indeterminacy of terms and icons in software localization. *In*: ANTIA, Bassey E. (Ed.). *Indeterminacy in terminology and LSP*. Amsterdam and Philadelphia: John Benjamins, 2007. p. 49-58. doi: https://doi.org/10.1075/tlrp.8.07sch

SMITH-FERRIER, Guy. *NET internationalization*: The developer's guide to building global windows and web applications. Upper Saddle River, NJ: Addison-Wesley, 2007.

UREN, Emmanuel; HOWARD, Robert; PERINOTTI, Tiziana. *Software internationalization and localization*: An introduction. New York: Van Nostrand Reinhold, 1993.

VAN DER MEER, Jaap. The fate of the localization industry and a call to action. *The LISA [Localization Industry Standards Association] Forum Newsletter*, v. 4, n. 4, p. 14–17, 1995.

W3C (WORLD WIDE WEB CONSORTIUM). W3C issues XML1.0 as a proposed recommendation. *World Wide Web Consortium Press Release*, Washington, [s.p.], 8 dez. 1997. Disponível em: http://www.w3.org/Press/ XML-PR. Acesso em: 10 mar. 2020.

* Direitos de tradução cedidos pela editora Routledge / Taylor & Francis Group

Referência completa do capítulo:
DUNNE, Keiran J. Localization. *In*: SIN-WAI, Chan. *The Routledge encyclopedia of translation technology*. Routledge, 2015. p. 550-562. Disponível em: https://www.routledge.com/Routledge-Encyclopedia-of-Translation-Technology-1st-Edition/Chan/p/book/9780415524841

A elaboração da tradução deste artigo é oriunda das atividades do projeto "Ferramentas tecnológicas para o ensino e a pesquisa na área de Tradução" (Edital nº 2/2017; Projeto nº 495), coordenado por Marileide Dias Esqueda e financiado pelo Programa de Bolsas de Graduação da Pró-Reitoria de Graduação e Divisão Discente da Universidade Federal de Uberlândia (UFU). Agradecimentos devem ser feitos à bolsista do projeto, Isabel Borges Ribeiro de Assunção Machado, então aluna do Curso de Sistemas de Informação e hoje aluna do Curso de Bacharelado em Tradução da UFU, especialmente pela revisão da tradução e por seu auxílio no que tange à tradução da terminologia pertencente à área de Computação.

[1] Nota das tradutoras: O termo "publicadora(s) de *software*" está sendo aqui adotado como tradução de "*software publisher(s)*", embora os termos "distribuidora(s) de *software*" sejam também usuais. Vale ressaltar que empresas publicadoras podem ser também desenvolvedoras de *software*.
[2] Para mais informações sobre requisitos de mercado exterior, cf. Giammarresi (2011), especialmente as páginas 39-40.
[3] N.T.: Uma localidade é definida como a área circunscrita de um país, região, estado, cidade. Disponível em: http://www.aulete.com.br/localidade. Acesso em: 5 mar. 2020.
[4] Para um estudo de caso que ilustra alguns dos problemas que podem ocorrer na ausência de uma abordagem estruturada de internacionalização, cf. Margulies (2000).
[5] Esses recursos são derivados de uma aplicação de amostras de arquivos chamada Scribble, desenvolvido pelo autor usando o Visual Studio 2010 C++. As amostras MSDN Archive, Visual C++ para Visual Studio 2010, estão disponíveis em http://archive.msdn.microsoft.com/vcsamplesmfc, acesso em: 8 set. 2012. [N.T.: o *link* se encontra inativo.]
[6] Nota das tradutoras: Os termos "autoria e publicação" estão sendo aqui adotados como tradução de *authoring and publishing*, embora os termos "editoração e publicação" sejam também usuais.

NOTA DO AUTOR

Keiran J. DUNNE – Professor da Kent State University. Doutor em Civilização Francesa pela The Pennsylvania State University e mestre pela Université de Haute-Bretagne/Rennes II, França. Tem ampla experiência em localização. Seus interesses de pesquisa são localização de *software* e *website*. É editor dos pediódicos *Perspectives on localization* e *translation and localization project management: The art of the possible*. Kent State University, *Deparment of Modern and Classical Language Studies*. Kent, Ohio, Estados Unidos da América.
ORCID: https://orcid.org/0000-0002-1892-1161
Página institucional: https://www.kent.edu/mcls/profile/keiran-dunne
E-mail: kdunne@kent.edu

NOTAS DAS TRADUTORAS

Carolina Flávia de HENRIQUE – Bacharel em Tradução (2017) pela Universidade Federal de Uberlândia. Pesquisadora autônoma. Uberlândia, Minas Gerais, Brasil.

ORCID: https://orcid.org/0000-0002-1892-1161
Currículo Lattes: http://lattes.cnpq.br/0827528609987168
E-mail: carolflavia97@hotmail.com

Jahynne Martins SALVADOR – Bacharel em Tradução (2017) pela Universidade Federal de Uberlândia. Pesquisadora autônoma. Uberlândia, Minas Gerais, Brasil.
ORCID: https://orcid.org/0000-0002-7537-2891
Currículo Lattes: http://lattes.cnpq.br/2738390247473423
E-mail: jahynne@gmail.com

Marileide Dias ESQUEDA – Doutora (2005) e Mestre (1999) em Linguística Aplicada pela Universidade Estadual de Campinas. Bacharel em Tradução (1995) pela Universidade Sagrado Coração. Realizou estágio de pós-doutorado na *Université de Montréal*, Canadá. Professora associada da Universidade Federal de Uberlândia. Universidade Federal de Uberlândia, Instituto de Letras e Linguística, Bacharelado em Tradução. Uberlândia, Minas Gerais, Brasil.
ORCID: https://orcid.org/0000-0002-6941-7926
Currículo Lattes: http://lattes.cnpq.br/3341029625579574
E-mail: marileide.esqueda@ufu.br

NOTAS DA REVISORA TÉCNICA

Isabel Borges Ribeiro de Assunção MACHADO – Graduanda do Bacharelado em Tradução da Universidade Federal de Uberlândia. Universidade Federal de Uberlândia, Instituto de Letras e Linguística, Bacharelado em Tradução. Uberlândia, Minas Gerais, Brasil.
ORCID: https://orcid.org/0000-0002-0042-7878
Currículo Lattes: http://lattes.cnpq.br/8411756171615618
E-mail: isabelmachado97@outlook.com.br