Universidad de Salamanca
Facultad de Traducción y Documentación
Departamento de Traducción e Interpretación

VNiVERSiDAD
D SALAMANCA

Localización e internacionalización de *software*: puntos de encuentro entre el localizador y el programador

Tesis doctoral realizada

por

Luis Alberto García Nevares

Bajo la dirección de

Dr. Jesús Torres del Rey
Dr. Adrián Fuentes Luque

Salamanca, 2016

UNIVERSIDAD DE SALAMANCA
FACULTAD DE TRADUCCIÓN Y DOCUMENTACIÓN
DEPARTAMENTO DE TRADUCCIÓN E INTERPRETACIÓN

VNiVERSiDAD
D SALAMANCA

TESIS DOCTORAL

TÍTULO: **Localización e internacionalización de *software*: puntos de encuentro entre el localizador y el programador**

AUTOR: Luis Alberto García Nevares

DIRECTORES: Dr. Jesús Torres del Rey

V^{O} B^{O}:

Dr. Adrián Fuentes Luque

V^{O} B^{O}:

SALAMANCA, 2016

Resumen

Por la manera en que se ha desarrollado la industria de localización de programas informáticos, el proceso de localización siempre ha funcionado como una caja negra que trabaja con su propio equipo de gestores y traductores, desvinculada de las metodologías y técnicas que se utilizan para desarrollar *software*. En muchos casos, y para muchas de las principales editoriales de *software* y desarrolladores de plataformas de programación, es, incluso, un proceso marginal que se invoca solo cuando hace falta traducir cadenas de texto. Esta inclusión tardía no solo genera enormes problemas en el proceso de la traducción de esas cadenas de texto, sino que, en muchas ocasiones, imposibilita que se pueda lanzar el producto en diversos mercados cuyos idiomas y culturas no encajan con las prestaciones que han sido tomadas en consideración durante el desarrollo y programadas al *software*.

En esta investigación haremos un resumen histórico del desarrollo de las computadoras y de cómo han surgido las tres principales plataformas de programación: las *mainframes*, las minicomputadoras y las computadoras personales. Veremos que, a la par con los equipos o *hardware*, han surgido una variedad de lenguajes de programación y de estrategias para desarrollar programas informáticos. Según la aplicación y uso de estos equipos se ha ido ampliando, la necesidad de dar orden a las estrategias y procesos que se llevan a cabo en el desarrollo de programas informáticos sirve de base para la creación y desarrollo de estrategias y metodologías de desarrollo de programas. Junto con estos desarrollos, el surgimiento de las computadoras personales favorece la creación de productos que no solo sirvan a los mercados (principalmente) estadounidenses, sino a otros mercados que se comunican en distintos idiomas y tienen necesidades particulares. Esta es la génesis de la industria de la localización y el punto en donde comienza a confluir la traducción con la informática.

Hasta este momento, todas las investigaciones que se han hecho sobre la localización de *software* toman como punto de partida el programador, un programador que investiga qué son programas multilingües y qué hay que tomar en consideración para crearlos. Nuestra propuesta nace del otro "lado", del lado del localizador que se acerca a la programación como experto en lenguas y en gestión multicultural. Este experto conoce los problemas que se

enfrenta el que está dentro de la caja negra de la localización, pero también está preparado para participar en los procesos anteriores, los que se llevan a cabo para crear programas nuevos. El "internacionalizador" posee los conocimientos y destrezas necesarias para poder formar parte del equipo de desarrollo de una aplicación desde sus comienzos hasta las etapas finales y ayudará a integrar en este proceso los requisitos necesarios para lograr que el *software* pueda ser localizado con facilidad llegado el momento de viabilizar su lanzamiento en otros mercados con necesidades lingüísticas, legales y culturales diversas. En el siglo de las comunicaciones es imposible pensar que se desarrolle *software* que no atienda las necesidades de más de un mercado. El internacionalizador puede ayudar al equipo de desarrollo de *software* a lograr esto.

Palabras clave: localización, internacionalización, desarrollo de software, programación, informática, traducción.

Abstract

The localization process has always been regarded as a black box that functions on its own, with a team of project managers and translators who never get involved in software development methodologies or technologies. This is mainly due to the way in which the software localization industry has developed. In many cases, and for many of the main software publishers and platform developers, localization is a peripheral process that is only invoked when text strings need translation. Including localization late in the development process not only brings enormous problems when translating strings, on many occasions it becomes impossible to launch the product in other markets which have the need to accommodate languages and cultures that do not fit within the features that were developed and included in the software.

This dissertation begins by making a historical account of the development of computers and describes how the three main programming platforms—namely mainframes, minicomputers, and personal computers—came into being. We shall see that, as hardware developed and new features were added, a variety of programming languages and strategies for developing software emerged. As features, application, and use of hardware further expanded, the need for organizing strategies and processes for creating programs became the basis for formulating and establishing software development strategies and methodologies. Along with these developments, the introduction of personal computers eventually promoted the need for creating products that serve not only markets in the United States, but other markets that communicate in different languages and exhibit particular needs of their own. Thus, the localization industry was born. It is at this point that translation and computers begin to come together and interact.

Up until now, research regarding software localization has had programmers as a starting point. These programmers have researched what multilingual programs are and what needs to be done to create them. Our proposal comes from the other "side," from the localizer's point of view; a localizer that approaches programming as an expert in languages and intercultural mediation. This expert knows the problems within localization's black box, but

is also prepared to participate in the processes that take place *before* translation, the processes that take place in order to create new software. These "internationalizers," as we refer to them, have all the necessary knowledge and skills to enable them to become part of a software development team from the beginning all the way through to the final stages. Their knowledge and presence will help integrate into this process the necessary requirements that will allow for smooth software localization, when the decision is made to launch the application into other markets that have diverse linguistic, legal, and cultural needs. In this century, mainly guided by communication, it is unthinkable to develop software that only attends to the needs of a single market. The internationalizer can help the software development team to accomplish this.

Keywords: Localization, internationalization, software developmente, programming, information systems, translation.

A Antonio. Sin ti esto no.

Agradecimientos

No iba a incluir una sección de agradecimientos. Pensaba que no era importante ni necesaria. Sin embargo, un repaso mental del camino recorrido me lleva desde que comencé con este proyecto hasta ahora, a este mismísimo instante, donde, según aprieto teclas en mi computadora, salen en la pantalla estas palabras, los momentos en que muevo el cursor hacia delante y hacia atrás, borro, corto, pego, escribo, reescribo. Este ahora no sería sin la ayuda o sin la presencia de muchas personas que conforman lo que para mí es una vida sobre todo, sobre todo, afortunada y feliz.

En primer lugar, agradezco a mis directores, el Dr. Jesús Torres del Rey y el Dr. Adrián Fuentes Luque. No solo por la sabiduría con la que im(com)parten su conocimiento, sino porque siempre tienen la palabra precisa que me anima a llegar hasta el final y a realizar un trabajo de excelencia. Un don precioso. Gracias.

A mis amigos, mi tesoro, los más olvidados, que han sido tan pacientes y comprensivos a lo largo de todo este proceso. Gracias.

A mis colegas del Programa Graduado de Traducción, por el ánimo y el apoyo que me brindan siempre. Gracias.

Niévelyn, gracias por prestarme tus ojos y tu conocimiento. Mú.

A mis padres, por enseñarme lo importante que es la educación y el trabajo. Gracias.

A mi marido Antonio, por el ánimo, el apoyo incondicional, la paciencia, la paciencia, la paciencia, la interminable paciencia que tienes. A Ana Beatriz y Paloma también. Gracias.

A todas y cada una de las personas que pasan por mi vida y dejan una huella, cualquiera que sea. Gracias.

Doctor, doctor, doctor, doctor, and doctor.

Índice

Lista de figuras

Lista de tablas

Acrónimos

ACM	*Association for Computing Machinery*
API	*Application Program Interface*
ARMMS	*Architectural Reference Model for Multilingual Software*
ARPANET	*Advanced Research Projects Agency Network*
ASCII	*American Standard Code for Information Interexchange*
BAN	*Body Area Network*
BD	bases de datos
bidi	bidireccional
BIOS	*Basic Input/Output System*
BSD	*Berkeley Software Distribution*
CLI	*Common Language Infrastructure*
CMS	*Content Management System*
COTS	*Commercial Off-The-Shelf*
DB	*Databases*
DSDM	*Dynamic Systems Development Methodology*
EBCDIC	*Extended Binary Coded Decimal Interchange Code*

FOSS	*Free and Open Source Software*
g11n	globalización
GALA	*Globalization and Localization Association*
GILT	*Globalization, Internalization, Localization, and Translation*
GNT	*Guidelines Navigation Tools*
GNT-DV	*Guidelines Navigation Tools - Development*
GNT-FS	*Guidelines Navigation Tools - Feasibility Study*
GNT-PP	*Guidelines Navigation Tools - Project Properties*
GNT-PR	*Guidelines Navigation Tools - Post-implementation*
GNU	*GNU is Not Unix*
GUI	*Graphical User Interface*
HTML	*Hyper Text Markup Language*
HTTP	*Hyper Text Transfer Protocol*
I/O	*Input-Output*
i18n	internacionalización
ICU	*International Components for Unicode*
IDE	*Integrated Development Environment*

IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things*
ISO/IEC	*International Organization for Standardization/International Electrotechnical Commission*
IT	*Information Technology*
ITS	*Information Technology Services*
IU	interfaz del usuario
JSON	*JavaScript Object Notation*
LAN	*Local Area Network*
L10n	localización
LISA	*Localization Industry Standards Association*
LTR	*Left-to-right*
MoSCoW	*Must have, Should have, Could have, Would not have*
MSDN	*Microsoft Developer Network*
NLSO	*National Language Support Objects*
OLIF	*Open Lexicon Interchange Format*
OS	*Operating System*
PARC	*Palo Alto Research Center*

QA	*Quality Assurance*
QC	*Quality Check*
RAD	*Rapid Application Development*
RTL	*Right-to-Left*
SDLC	*Software Development Life Cycle* o *Systems Development Life Cycle*
SO	sistema operativo
TAC	traducción asistida por computadora
TAO	traducción asistida por ordenador
TBX	*Term Base eXchange*
TI	tecnología de la información
TIC	tecnologías de la información y comunicación
TM	*Translation Memory*
TMX	*Translation Memory eXchange (format)*
UI	*User Interface*
VWAN	*Very Wide Area Network*
WAN	*Wide Area Network*
WIMP	*Windows, Icons, Menus, Pointing devices*

WYSIWYG	*What You See Is What You Get*
XLIFF	*XML Localization Interchange File Format*
XML	*Extensible Markup Language*

If you think about it, language is the purest form of data transmission that exists.

Brain Games (Nigro 2014)

Introducción

No hay más. Un solo camino
que se quisiera tomar,
mas la suerte del andar
maltrata y confunde el tino.

Juan Antonio Corretjer

Motivación

Esta investigación busca conciliar mi formación académica con mi experiencia profesional que, a lo largo de los años, ha dado giros inesperados, no buscados y muchas veces de profunda satisfacción y (sobre todo) disfrute personal. Luego de más de un decenio trabajando como programador y gestor de informática para varias empresas, y de una decisión tajante, me topé con la oportunidad de abordar un campo laboral cuya producción me parecía más natural que académica: la traducción. Aun así, no me sorprendió que, según pasaron los años y tuve más contacto (y experiencia) con esta profesión, primero la intuición, luego el sentido común y finalmente el descubrimiento de la existencia de programas de formación académica me ayudaron a comprender que era (casi) todo lo contrario, que para traducir hace falta más que tener lo que me parecía natural, que hacía falta formarse. Ello fue lo que me motivó a completar una Maestría en Traducción y, ahora, a ampliar aún más mi conocimiento en este campo por medio de estudios doctorales.

Llegado el momento de decidir el ámbito de mi investigación, resulta muy natural que escogiera abordar la localización, un área de estudio que combina la traducción, campo en el que también cuento con más de un decenio de experiencia laboral y una amplia formación a nivel graduado, con mi formación empresarial especializada en informática y la experiencia laboral como programador y gestor de informática que mencionara. En un principio, me propuse integrar dentro del proceso de desarrollo de *software* lo que entiendo es un paso ineludible en nuestro presente: la internacionalización. Observé que, como bien señala Mata Pastor,

> […] desde el punto de vista de la internacionalización lingüística y cultural, los contenidos y productos que se publican o salen al mercado cada día adolecen

> de una manifiesta falta de conciencia de que existen otros usos y costumbres allende las fronteras nacionales y continentales […]
>
> (2016, p. 312)

Esta falta de conciencia se traduce en una limitación en el acceso a los servicios informáticos que tienen otros mercados, culturas y países, pero no se limita a ello, pues puede muy bien situarse dentro del contexto del mismo país que los produce, ya que es posible que un mismo país tenga necesidades multilingües diversas:

> […] in Pondicherry, an [sic] Union Territory in India, the official languages are Tamil, Telugu, Malayalam and English. Hence, a multilingual working environment is needed for the software in order to provide effective IT services for the people living in Pondicherry, which is an example for multilingual community.
>
> (Venkatesan 2008, p. 13)

Según avancé en la investigación, descubrí no solo que ya se habían propuesto estrategias para lograr esta integración y reducir esta limitación, sino también que ya los autores Murtaza y Shwan (2012) se habían planteado cuestionamientos similares y habían publicado una serie de directrices para facilitar la internacionalización de los programas informáticos.

Posición de la industria de la localización de cara a la industria del desarrollo de *software*

A lo largo de la lectura de esta investigación descubriremos que, por la manera en que ha devenido su desarrollo histórico, la industria de la localización se encuentra en posición de desventaja en relación con la industria de desarrollo de *software*, pues la génesis de aquella la propicia el mercado de la industria de las computadoras y sus necesidades de expansión. La localización queda reducida a un subproceso desligado por completo del desarrollo de los programas informáticos, con su propio grupo de gestores y traductores, que trabaja aparte y no participa a lo largo de este desarrollo, salvo al final, cuando llega el momento de traducir cadenas de texto. A pesar de ello, aunque en apariencia este (sub)proceso se trata como algo

independiente del desarrollo de *software*, es la editorial de *software*[1] la que controla y dirige todo lo relacionado con el producto final de la localización. La editorial no entiende que las decisiones que se toman a lo largo del proceso afectan la calidad de la traducción, algo que mejoraría con la participación activa de quienes forman parte del proceso de localización. Así, limitan la posibilidad de que este equipo logre una buena mediación intercultural; su fin es satisfacer las necesidades inmediatas del mercado de la industria de las computadoras. La localización no es más que otro proceso industrial que se enlaza indirectamente a la línea de producción.

> Localization came into being owing to **the market needs of the computer industry**, requiring software applications to be usable in different locales[2]. With this background the domain has developed as an **industrial process** […].
>
> (O'Hagan y Mangiron 2013, p. 100, énfasis añadido)

La visión de la localización como proceso industrial produce aislamiento y la encajona dentro de un compartimiento reducido. Este aislamiento, añadido a la carencia de conocimiento, destrezas y falta de interés hacia el ámbito de las tecnologías y su desarrollo técnico por parte de muchos de los investigadores y los alumnos en los estudios de la traducción, no ha permitido que esta rama de estudios se involucre de manera significativa en los procesos que conforman el desarrollo de programas informáticos. Los esfuerzos por mejorar esta situación se ven fácilmente tronchados pues, aunque el sentido común dice que la localización debe estar dirigida a satisfacer al mercado meta, no es así. Las localizaciones responden a la influencia de sus patrocinadores, los desarrolladores de *software*.

Carbonell Cortés nos recuerda que ninguna producción discursiva está exenta de carga ideológica, y donde está presente una ideología también están presentes los intereses y la influencia de quienes la patrocinan.

[1] Las editoriales de *software* o *software publishers* son empresas que se dedican a crear y vender programas informáticos, como por ejemplo, Microsoft.

[2] Véase nota 92.

> Es muy importante que los traductores, y sobre todo los futuros traductores, reconozcan que las cuestiones ideológicas no se dan sólo en textos muy específicos o en situaciones muy específicas, sino que permean toda producción discursiva.
>
> (1997, p. 67)

Así, los localizadores, siendo traductores de cadenas de texto, también se encuentran en posición vulnerable y están sometidos a los intereses del patrocinador. A pesar de que las relaciones de poder son un asunto ampliamente discutido en los estudios de traducción, no han tenido el mismo impacto en la práctica de la localización.

> […P]ractice-led approaches dominant in localization seem to have glossed over issues such as the power relationship between the software developer/publisher and the translator and the influence of the former on certain translation decisions which are imposed on the latter. […A] localized software product may in fact most strongly resonate with the publisher's (the commissioner's) values despite the fact that it is ostensibly presented as "target-oriented". Such a perspective seems to have remained so far largely unexplored in localization research despite the fact that […] concepts such as "patronage" […] have been explored in Translation Studies to acknowledge the influence of patrons (translation clients, publishers, etc.) on translation. Such concerns, well argued in the discipline, have so far not been applied in the domain of localization, which instead has been dominated but the industry's concern over more immediate practical issues […] relating, for example, to how to deliver localized products most efficiently on time and within budget.
>
> (O'Hagan y Mangiron 2013, p. 100)

Las mismas autoras señalan que la *estructura* de la industria y la *manera* en que las partes implicadas[3] afectan la localización y su nivel de calidad han sido muy poco estudiadas:

> […] Broader, less immediate questions such as the industry structure and the different stakeholders with their various influences on localization practices and

[3] Véase nota 63.

> quality have so far attracted little attention within the industry or in Translation Studies […].
>
> (*Ibidem*, p. 101)

Más adelante, en el capítulo 2, veremos que los procesos que están relacionados con la localización sí han sido objeto de estudio, pero sus investigadores son programadores que buscan entender qué hay que hacer para que un programa sea multilingüe. La motivación de estos autores es variada. Por poner un ejemplo de cada uno, para Young, "[…] there was a lack of practical advice on integrating internationalization into the standard procedure for software development […]" (2001, p. 28). Abufardeh encontró que había una "[…] lack of an effective internationalization strategy, as well as [… a] lack of [a] comprehensive framework for the integration of internationalization and localization activities into the software development lifecycle […]" (2008, p. 12). Para Venkatesan el fenómeno de la globalización requiere que lo que él llama el aparato "clave" del humano en la actualidad, la computadora, dé apoyo a múltiples idiomas y, entonces, "[… i]n order to make computer multilingual, appropriate software has to be designed" (2008, p. 3). Murtaza y Shwan han encontrado que "[…] multilingualism of the software isn't part of the requirements and design documents […]" (2012, p. 17) y esto dificulta grandemente el lanzamiento de los programas en otros idiomas.

Pensamos que no solo las razones que exponen en sus respectivas investigaciones han motivado a estos autores a profundizar sobre el tema de la localización y la internacionalización de *software*. Por el contenido de su tesis, podemos intuir que Erica Young tiene cierto interés en idiomas y suficiente conocimiento de alemán como para considerarlo como un posible idioma en su proyecto de prueba[4]. También por el contenido de su tesis y la manera en que aborda el tema, Abufardeh demuestra especial (y posiblemente profundo) conocimiento del árabe, y lo hace un tema importante en su investigación dedicándole dos capítulos a temas relacionados directamente con este idioma. Venkatesan, de su parte, desarrolla programas en

[4] "I chose English (US) and German (Standard) as the two initial locales, not because they showed the most promise as a market for this application, but because those are the two locales with which I am the most familiar" (2001, p. 68).

inglés, tamil, malayo e hindi para poner a prueba los modelos que propone en su investigación. Murtaza y Shwan no mencionan explícitamente su conocimiento e interés por el árabe, pero a lo largo de toda su tesis prestan especial atención a los temas relacionados con la arabización y el apoyo a lenguajes arábigos y similares. También mencionan que Murtaza trabajó como gestor de proyecto y programador principal en un proyecto monolingüe en árabe, con lo cual podemos presumir que, al menos, se siente cómodo manejando el idioma (*Ibidem*, p. 78). Las investigaciones que han realizado estos programadores tienen como denominador común los idiomas y las culturas que de una u otra manera forman parte de sus vidas.

La industria de la localización tiene muchísimo más que aportar a la informática que una serie de cadenas de texto traducidas. Proponemos que la participación de localizadores expertos en programas multilingües a lo largo de todo el proceso de desarrollo de *software* puede ayudar a crear un producto más útil y de mejor calidad para beneficio de la editorial y, sobre todo, del usuario final. Es un buen momento para que la balanza de poder se nivele.

El estancamiento de los modelos de localización actuales

La localización se percibe y se trata como una caja negra que recibe cadenas de texto en un idioma fuente y produce cadenas de texto en el idioma meta.

> […] T]he scope of translation within the localization paradigm can be illustrated in the core working unit of text, typically referred to in the localization industry as a "string", in contrast to a paragraph, a whole document or even a larger unit of culture. This notion of translation in the localization industry, based mainly on short fragments of decontextualized strings, may be partly responsible for the restricted and reduced scope of translation. In short, in the field of localization, translation has been condemned to be the conversion of these strings from the SL into the TL, allegedly without any cultural implications or other challenging issues, which are treated separately outside "translation" […].
>
> (O'Hagan y Mangiron 2013, p. 102)

Esta visión limitada de la localización, con un alcance restringido y reducido, es la que ha fijado el modelo de la localización actual y lo mantiene inamovible.

Dunne (2006) nos arroja luz sobre lo que puede ser la principal razón que explica por qué seguimos usando modelos para la localización que no han evolucionado desde que se comenzó a localizar. Por un lado, en la compleja madeja corporativa de una empresa inciden una variada gama de actores que tira en múltiples direcciones. Cada uno ejerce presión en una dirección que es afín a sus propios intereses y metas. En las empresas, los desarrolladores o autores de las aplicaciones se enfocan principalmente en las funcionalidades del dominio[5], con lo cual prestan poca o ninguna atención a asuntos lingüísticos o culturales. También, para los programadores, añadir apoyo a múltiples idiomas a un programa añade trabajo y esfuerzo, algo que es preferible evitar de no ser un requerimiento expreso.

> […] The people who build the products – programmers – are usually also the people who design them. Programmers are often required to choose between ease of coding and ease of use[6]. Because programmers' performance is typically judged by their ability to code efficiently and meet incredibly tight deadlines, it isn't difficult to figure out what direction most software-enabled products take […].
>
> (Cooper, Reimann y Cronin 2007, p. 8)

Ventas, mercadeo y la división de asuntos legales, con su conocimiento sobre las necesidades del mercado, las oportunidades económicas que presentan los distintos *locales* y las posibles implicaciones legales regidas por regulaciones locales o nacionales, pueden resultar buenos aliados para fomentar la localización. Sin embargo, los gestores de la empresa la perciben como un "back-office spending rather than as a wise investment", algo que se debe minimizar a toda costa.

Estas perspectivas fragmentadas sobre la localización, continúa argumentando Dunne (2006), se ven reafirmadas por los currículos educativos pues "[…] foreign languages and translation, computer science, graphic design, as well as business and management tend to be

[5] Véase nota 128.

[6] Los autores hacen referencia a esto en el contexto de la usabilidad, pero aplica de igual manera a la internacionalización de *software*.

mutually exclusive areas of study […]". En este ámbito, los desarrolladores o creadores de *software* carecen, por lo general, de la experiencia práctica y el conocimiento de idiomas y culturas, incluso la propia, para entender las implicaciones económicas de prácticas y costumbres en la programación que inciden en elementos lingüísticos y culturales. En el mundo profesional y empresarial las personas relacionadas con mercadeo, ventas y gestión desconocen la forma en que las dimensiones lingüísticas, legales y culturales afectan la usabilidad, la imagen del producto e incluso si es legal o no lo que están haciendo. Inmersos en este desconocimiento se comprometen a incluir (o dejan de incluir) funciones o prestaciones en el *software* sin entender las repercusiones que tienen estos (no-)compromisos a nivel global.

Por otro lado, los traductores, que poseen las destrezas necesarias para señalar, discutir y solventar este tipo de problemas, carecen del conocimiento técnico que les permitiría aportar sus destrezas lingüístico-culturales a un proyecto de este tipo. Tampoco tienen el conocimiento necesario que les ayude a entender la manera en que actúan las fuerzas de los mercados y a saber en qué consisten las exigencias comerciales que rigen la decisión de localizar.

Los educadores en el ámbito de la traducción y de las lenguas extranjeras, según Dunne, tienden a tener una formación humanística que enfatiza en literatura y análisis literario, con lo cual, a menudo no tienen el conocimiento profundo necesario para entender los proceso ni las herramientas relacionadas con la localización, ni las realidades que rigen a la industria en la actualidad. Los elementos que conforman la industria de la localización (es decir los clientes externos e internos, proveedores, subcontratistas, asociaciones, expertos de la industria, educadores y organizaciones estandarizadoras) tampoco aportan unidad pues cada cual tiene una perspectiva distinta sobre la localización. La situación se agrava ante la falta de comunicación que existe entre todos estos componentes. "[… L]ocalization simply does not lend itself well to being perceived *globally* […]", sentencia Dunne. (*Ibidem*, pp. 2-3).

Nuevos perfiles en la localización

Nuestra investigación busca proponer un modelo integrador que combine el conocimiento humanístico con el conocimiento tecnológico por medio de la participación de expertos que se sientan cómodos trabajando en ambos ámbitos, proveyéndoles las herramientas

que les permitirán participar activamente en el desarrollo de aplicaciones multilingües y con sensibilidad multicultural. La gestión de proyectos de localización ya incluye equipos multidisciplinares, multilingües y multiculturales.

> […] L]ocalization teams would typically be coordinated by a project manager overseeing schedules and budgets, a linguist to monitor any linguistic issues, an engineer to compile and test localized software and on-line help and a desktop publisher to produce translated printed or on-line manuals.
>
> (Esselink 2006, p. 25)

Solo necesitamos entender los beneficios de la expansión de la participación de este equipo (o más bien de una parte de este) hacia las demás fases y etapas del desarrollo de *software*, sobre todo en las etapas tempranas. Esperamos que la aplicación práctica de un proyecto participativo como el que proponemos ayudará no solo a mejorar la calidad del producto final, el *software* multilingüe, sino que también facilitará y optimizará el proceso de localización.

Partiendo de las investigaciones publicadas que hemos mencionado, de la observación y de mi experiencia profesional es que planteo los objetivos, hipótesis y preguntas de investigación a continuación.

Objetivos, hipótesis y preguntas de investigación

1. Objetivos
    A. Investigar las maneras (metodologías, estrategias, diagramas, *frameworks*, lenguajes de programación usados) en que se desarrolla *software* a fin de determinar el conocimiento y destrezas que un localizador necesita para que pueda servir como mediador durante todo el proceso de desarrollo de *software* y no solo durante la fase final.
    B. Investigar la manera en que interactúa la industria de desarrollo de *software* con la industria de la localización.
    C. Proponer una revisión del concepto del localizador y su papel en el proceso de internacionalización del *software* multilingüe.

   D. Describir las competencias del "nuevo" localizador, es decir, el que conoce las maneras antes mencionadas, desde el punto de vista de la traducción especializada y desde la programación.
   E. Proponer una serie de recomendaciones profesionales y formativas basadas en el análisis y descripción de las competencias mencionadas.

2. Hipótesis – Incluir a un localizador a lo largo de todo el proceso de desarrollo de software es factible y deseable para facilitar la labor de localización en el futuro. Su participación también propicia mayor rentabilidad a largo plazo en el proyecto pues su labor mediadora le da presencia constante al usuario final, mejora la calidad del producto en su versión "original" y simplifica el desarrollo en cuanto a las cuestiones relacionadas con el idioma y la cultura que inciden en la programación. El localizador está presente en doble función: usuario y mediador, con lo cual ofrece una visión completa y actualizada en todo momento de las cuestiones lingüísticas, culturales y técnicas que deben tenerse en cuenta en el proceso de desarrollo, localización e internacionalización de los programas y productos.
3. Preguntas de investigación
   A. ¿Qué necesita saber un traductor que quiera profundizar más en este campo?
   B. ¿Cómo interactúa una editorial de *software* (*software publisher*) institucional, privado o individual con la actual industria de la localización?
   C. ¿Qué ventajas supone la posibilidad de involucrar más profundamente en este campo a un localizador?

Metodología

Se llevará a cabo una revisión de literatura a fin de establecer una contextualización histórica de la interacción entre los ámbitos de la localización y la informática, y además se revisarán las investigaciones previas relacionadas con el tema. La contextualización histórica nos ayudará no solo a entender la evolución de ambos campos y cómo se interrelacionan, sino que, además, permitirá apreciar desde una mejor perspectiva la importancia y la necesidad ineludible de desarrollar *software* multilingüe en nuestra realidad presente. Según progrese nuestra investigación, veremos las aportaciones que puede hacer un localizador a lo largo de las etapas que comprenden el desarrollo de *software*. El lector podrá entender cuáles son las

competencias del "nuevo" localizador, el localizador que es capaz de integrarse a lo largo de todo el proceso de desarrollo y aportar sus conocimientos lingüísticos y de gestión intercultural (y hablar el idioma del programador) para lograr una mejor internacionalización de aplicaciones informáticas. Siendo este también un usuario en potencia[7], el localizador tiene la posibilidad de mejorar la experiencia del usuario aportando al proyecto el punto de vista del usuario. Como consecuencia de la inclusión de este nuevo localizador a lo largo de este proceso, se reducirán los riesgos inherentes al proceso de desarrollo de *software* multilingüe y se habilitará la posibilidad de ampliar los idiomas y culturas hacia los cuales se puede localizar el programa. Terminaremos con una serie de conclusiones y recomendaciones.

Organización de la tesis

La investigación que presentamos se divide en tres capítulos e incluye esta introducción, un capítulo de conclusiones y recomendaciones (alineado con el objetivo E de nuestra investigación y con las preguntas de investigación A, B y C), y una bibliografía que comprende los textos citados y las referencias consultadas a lo largo de nuestra investigación.

El primer capítulo hace un recorrido histórico del desarrollo de las computadoras en cuanto a *hardware* y *software* desde sus inicios hasta la actualidad. Incluye el desarrollo de las tres gamas de equipos que han persistido a lo largo de la historia de las computadoras (las *mainframes*, las minicomputadoras y las microcomputadoras) y describe el desarrollo de los primeros lenguajes de programación. De la mano de estos desarrollos surgen las estrategias de análisis y diseño, las cuales abordamos y describimos siguiendo los trabajos de Avison y Fitzgerald (2006a, 2006b, 2003). Este capítulo se alinea con el objetivo A de nuestra investigación y con la pregunta de investigación A.

En el capítulo dos se establece la relación que existe entre el desarrollo de *software* para microcomputadoras con los inicios del campo de la localización y se presentan las definiciones de términos fundamentales en el desarrollo de *software* multilingüe: localización,

[7] Véase la hipótesis en la página 11.

internacionalización y globalización. Continuamos con una breve descripción de la relación entre *localización* y *traductología*, y se presenta un resumen crítico de las cuatro investigaciones principales que abordan el tema de la localización, siempre desde el punto de vista de la informática. Este capítulo se alinea con los objetivos A y B de nuestra investigación y con las preguntas de investigación A y B.

El tercer y último capítulo presenta y describe la figura del internacionalizador, un experto en traducción que domina el lenguaje informático y es capaz de participar en todas las etapas del desarrollo de *software* a fin de lograr una internacionalización adecuada del nuevo *software*. Aquí describimos las competencias que posee este internacionalizador, señalamos algunas de las herramientas visuales que usan los programadores al desarrollar *software* y explicamos las consideraciones relevantes en el desarrollo de programas informáticos que tienen que ver con la codificación de caracteres. Incluimos una lista de lenguajes de programación en uso en la actualidad, según la plataforma en que se pueden usar, una breve descripción de cada lenguaje y las prestaciones relacionadas con internacionalización que oferta cada uno. Luego mencionamos y describimos brevemente algunos entornos de desarrollo de *software*, a fin de que el internacionalizador pueda tener idea de las prestaciones que ofrecen. Discutimos algunas directrices de redacción que proponen las grandes plataformas de *software*. Este capítulo se alinea con los objetivos C y D de nuestra investigación y con las preguntas de investigación A, y C.

En el capítulo titulado "Conclusiones y recomendaciones" hacemos un resumen de nuestra investigación y una serie de recomendaciones profesionales y formativas para el internacionalizador.

¿Descriptivismo o prescriptivismo?

Muchas de nuestras propuestas podrán recibir como crítica que tienen un cariz prescriptivista. Los lenguajes de programación son una serie de reglas semánticas y sintácticas, un lenguaje controlado, que tiene como fin gestionar los cálculos y procesos que la computadora ejecuta. Estas reglas semánticas y sintácticas son, por lo general, estrictas y, para que un programa pueda compilarse, tienen que seguirse al pie de la letra. Incluso, aunque un

programa compile[8], la interpretación de estas reglas puede dar pie a un error de lógica, un error de programación que en muchos casos es sumamente difícil de detectar. La naturaleza de la programación es, en su más pura esencia, prescriptivista.

No solo es la esencia de la programación lo que hace que nuestras propuestas suenen a prescriptivismo. Dunne (2006, p. 5) explica que la concienciación diacrónica de cuáles son los procesos para traducir un programa y la importancia relativa de cada uno de ellos ha evolucionado desde adentro hacia afuera, desde el programa en sí hacia los demás participantes que se involucran en el proceso. En los inicios, se intentó "transformar" los programas y su documentación de una versión en un idioma a otro. La localización surge de la conciencia de que este proceso no era simplemente traducir, que involucraba más aspectos. Los subsiguientes esfuerzos por localizar programas desvelaron que, al realizar ciertas actividades antes de localizar, se podía facilitar esta labor. Así nació el proceso de internacionalización. Cuando surge la necesidad de lanzar programas en varios idiomas de forma simultánea, las editoriales de *software* se dieron cuenta de que al adoptar estrategias de globalización a nivel empresarial se facilita todo el proceso de desarrollo de programas informáticos. Así esto, la concienciación de los procesos de GILT[9] y la interrelación entre sus componentes se ha desarrollado de manera paulatina en las décadas pasadas. Cada capa ha sido explorada como si fuera una *matrioska* pero que, opuesto a lo esperado, el interior abarca ámbitos cada vez mayores. Este desarrollo histórico ha conformado la manera en que se ha abordado el tema de la localización y explica su naturaleza descriptiva:

> […] approaches to localization have historically been largely *descriptive* and characterized by incremental improvements, as companies have successively

[8] Los programas basados en lenguajes no interpretados necesitan estar compilados para que una computadora los pueda ejecutar. Véase la explicación sobre los compiladores y los lenguajes compilados en la página 26 y subsiguientes.

[9] Véase la explicación sobre GILT en la página 65.

> grappled with and resolved issues of translation, then localization, on to internationalization and finally globalization […].
>
> (*Ibidem*, p. 6, énfasis en el original)

Sin embargo, el autor piensa que este enfoque no favorece la gestión del proceso de GILT como un todo.

> […] the effective management of GILT processes requires a *prescriptive* approach, beginning with globalization and progressing downstream through the sequential processes of internationalization to localization and finally to translation. This fact has powerful ramifications for practice, process, workflow and even enterprise structure. [… T]here is a chain of dependencies from globalization, to internationalization on through to localization and translation. Working upstream against the flow of these dependencies imposes a point of diminishing returns at each step of the ladder. The relative success of translation and localization efforts depends to a greater extent on the successful implementation of internationalization strategies. However, these in turn depend on enterprise-level commitment to globalization strategies […].
>
> (*Idem*, énfasis en el original)

Trabajar en contra de la corriente, sobre todo, minimiza los beneficios y la rentabilidad de lo que se produce en cada uno de los pasos. El autor termina su argumentación haciendo un llamamiento para cambiar la perspectiva de la localización para que la estrategia funcione desde el punto de vista global, es decir, a nivel empresarial, y a que se evalúen las ventajas de una implementación prescriptiva de las estrategias de globalización e internacionalización.

> It is for precisely this reason that global perspective is so critical in localization. The failure to adopt effective enterprise-wide globalization strategies and to implement appropriate internationalization strategies at the level of product design means that localization processes will be condemned to remain relatively descriptive and reactive, and gains in productivity and cost containment will be incremental at best. Conversely, the prescriptive implementation of globalization and internationalization strategies, given their proactive focus on avoiding problems, will facilitate localization and enable exponential savings by eliminating costly problems *before localization even begins*. To the extent that different participants lack global perspective, they risk being unaware of the downstream impact that their decisions may have on the other participants in the process and on the project as a whole. Likewise, when localization is driven by opportunistic sales, as opposed to defined international sales and marketing

> strategies, and when it is carried out after product development is complete instead of being integrated into the development cycle as a key component of corporate globalization strategies, localization professionals are forced to retroactively fix problems and address the consequences of defective planning after the fact. The bottom line is that in the absence of global perspective, localization is invariably more difficult and more costly than need be the case.
>
> (*Ibidem*, pp. 6-7, énfasis en el original)

Pensamos que es, sobre todo, la reactividad que ocasiona un abordamiento descriptivista lo que añade más costos y destruye la posibilidad de que un programa informático pueda ser localizado adecuadamente para que funcione bien en otros mercados. Como veremos más adelante cuando estudiemos los procesos que se llevan a cabo al desarrollar *software* y las diferentes propuestas existentes para crear programas multilingües (capítulos 1 y 2), la consecuencia de que se señale un problema vinculado a la localización en la parte más "baja" de la cadena (es decir, cuando un traductor señala algo que afecta la calidad de la localización y que no se puede resolver por causa de las decisiones tomadas en etapas anteriores al proceso de traducción) es que se crea un efecto dominó en todo el proyecto. Esta situación involucra costos de reelaboración y ello podría forzar, incluso, la cancelación de la creación de versiones localizadas (o del proyecto). Una perspectiva prescriptiva es *proactiva* y evita que se den situaciones en que se imposibilite localizar un producto o se desmejore su calidad, y si no las puede evitar del todo, al menos reduce su impacto.

En consonancia con la explicación de Dunne, buscamos resaltar la importancia de que una perspectiva más globalizada que permee todo el proceso de desarrollo de *software* permite ampliar el acceso a otras culturas y a otros mercados, por medio del idioma, y posibilita la expansión económica de la inversión en tiempo y dinero de los autores de una aplicación. Siguiendo los objetivos y preguntas de investigación antes mencionados, nuestra propuesta también tiene un fin didáctico (que según el tema y el enfoque puede ser muy prescriptivo) en donde aportamos claves y competencias para el futuro localizador.

Lenguaje utilizado y variantes regionales

Dado que el autor y los directores pertenecen a distintas comunidades lingüísticas, se ha respetado el lenguaje, los conceptos y las connotaciones regionales originarios del doctorando, salvo en aquellos casos en los que una expresión pueda resultar malsonante o confusa para el lector de España, país en el que se presenta y defiende esta tesis doctoral. A la vez, debido a la naturaleza traductológica del trabajo, se ha intentado limitar el uso de anglicismos, más natural en la variante del español puertorriqueño, y proponer equivalentes plausibles en español.

Capítulo 1: Contextualización histórica: desarrollo de las computadoras, de los lenguajes de programación y de las metodologías de desarrollo de *software*

1.1 Trasfondo histórico de los sistemas informáticos

Desde un punto de vista histórico, los lenguajes de programación que están disponibles para desarrollar programas informáticos han condicionado el modo en que se codifican dichos programas. También han influido la forma en que se estructura el desarrollo de sistemas informáticos[10]. De igual manera, la tecnología disponible dicta las limitaciones técnicas que tienen dichos lenguajes, pero según evoluciona la tecnología, y con ella los lenguajes, comienzan a desarrollarse estrategias para darle coherencia al proceso de desarrollo de sistemas informáticos[11] (Coad y Yourdon 1991, p. 2). Gordon B. Davis nos recuerda que existe un vínculo importante entre la evolución histórica de los equipos y los sistemas informáticos: "[w]ithout the development of computing devices, information systems would not have become a field of study and research […]" (2006, p. 17). Para entender estas relaciones y la importancia y el posicionamiento de las metodologías de desarrollo de software en el campo de la informática actual, y, por ende, para nuestro internacionalizador propuesto, debemos echar una mirada al desarrollo de las computadoras electrónicas y de los lenguajes de programación, y a cómo, a partir del desarrollo evolutivo de estos, surgen las estrategias de análisis y diseño de sistemas informáticos. Veremos que el desarrollo de los lenguajes de programación va de la mano con el desarrollo de las computadoras, pero que las estrategias de análisis y diseño, y el estudio de los sistemas de información llegan años más tarde.

Según avance en este capítulo, esperamos que el lector pueda tener una perspectiva histórica de cómo ha evolucionado el campo de la informática que le ayude a comprender la manera en que dicho desarrollo histórico ha determinado la manera en que se crea *software*.

[10] Para las organizaciones, los sistemas informáticos, sistemas de información o *information systems (IS)* son "[…] the systems that deliver information and communication services to an organization" o "the organization function that plans, develops, operates, and manages the information systems […]" (Davis 2006, p. 12). Esta definición abarca, por un lado, el ámbito de los programas y equipos y, por otro, el de la entidad dentro de la organización que gestiona estos equipos y los servicios informáticos.

[11] En específico, el proceso de análisis y diseño de sistemas informáticos.

Las metodologías de desarrollo de *software* ayudarán al localizador a entender el proceso mental implícito (incluso el no-proceso, como veremos en el apartado 1.11) que se lleva a cabo a lo largo de un proyecto de desarrollo de *software* y a derivar de este entendimiento competencias y conocimientos que le permitirán ser un participante activo durante el mismo (objetivo A, página 11).

Un traductor que desee profundizar más en este campo se verá beneficiado de conocer este trasfondo histórico, qué circunstancias impulsan el surgimiento de las metodologías de desarrollo y el estado de dichas metodologías en la actualidad. El trinomio *mainframe*/mini/pc no queda del todo visible para muchas personas, aun cuando, en su diario vivir, es muy posible que interactúen con este de una manera o de otra. Por esta invisibilidad, consideramos que es importante que el localizador conozca de dónde surgen estas plataformas informáticas y hacia dónde se dirigen, ya que, como dijimos, casi todas ellas forman parte esencial de nuestra cotidianeidad y no son ajenas a la industria de la localización. Esto está en consonancia con la pregunta de investigación A en la página 12.

1.2 De tarjetas perforadas a *mainframes*

En torno a 1890 el inventor estadounidense Herman Hollerith crea la tarjeta perforada y una serie de equipos accesorios para ayudar a la oficina del Censo de los EE. UU. a llevar a cabo cálculos simples y recuentos basados en la información que era cifrada en estas tarjetas. A fin de comercializar su invento, Hollerith funda en 1896 una empresa llamada la Tabulating Machine Corporation, la cual luego de varias adquisiciones y cambios de nombre, se convierte en 1924 en la International Business Machines Corporation —IBM en nuestros tiempos—, empresa clave en el desarrollo de equipo de computadoras a nivel mundial. A la vez que proliferaba la venta de equipos de tarjetas perforadas y se ampliaban sus prestaciones, otras empresas e inventores comenzaron a crear y a vender calculadoras mecánicas para llevar a cabo sumas —algunas más complejas y costosas podían multiplicar y hasta dividir— que encontraron uso en ámbitos variados: en las empresas, en el campo de la ingeniería, en las ciencias y, sobre todo, en la astronomía. Las personas diestras en el uso de estos aparatos podían hacer cálculos a gran velocidad, y el gobierno y los laboratorios de investigación comenzaron a

crear grupos de personas, que por lo general eran mujeres[12], para trabajar con estos equipos. Las "computadoras" tenían la responsabilidad de hacer operaciones matemáticas sobre los datos que se les proporcionaban. Así fue como este término pasó a referirse también a las máquinas que eran capaces de hacer el trabajo del grupo de computadoras (Swartzlander 2015).

La Segunda Guerra Mundial dio paso a la primera generación de equipos de computación especializados. Por años los estadounidenses se han preciado de haber sido los creadores de la primera computadora electrónica: la ENIAC, que entró en operaciones el 14 de febrero de 1946, y aunque en la actualidad tienen el cuidado de llamarla "la primera computadora electrónica **polivalente**" (University of Pennsylvania SEAS 2013, énfasis añadido), por años se pensó que había sido no solo la primera computadora electrónica, sino también el primer aparato electrónico en usar lenguaje binario[13] para realizar las computaciones matemáticas internamente. Poco a poco la historia se nos ha ido desvelando y ya sabemos que esas primeras computadoras electrónicas fueron la serie de Colosos creados por Tommy Flowers en Bletchley Park[14], computadoras cuya única función era desencriptar lo que los ingleses bautizaron "Tunnys", las comunicaciones entre Hitler y su alto comando que estaban encriptadas usando las encriptadoras Lorenz (The National Museum of Computing 2015).

[12] Desde el punto de vista histórico, bajo condiciones normales habrían sido los hombres los encargados de esta faena, pero mucho de la historia de las computadoras coincide con la Segunda Guerra Mundial. La mayor parte de los hombres formaban parte de las defensas nacionales, así que las mujeres estaban más disponibles.

[13] La idea de usar un lenguaje binario en las computadoras la concibió Atanasoff (véase nota 18).

[14] Interesante por demás es el relato del científico convertido en historiador Brian Randell sobre cómo logró averiguar sobre la existencia y los detalles de este proyecto secreto (véase The National Museum of Computing 2013), que al terminar la Segunda Guerra Mundial fue destruido por órdenes de Winston Churchill a fin de evitar que cayera en manos enemigas (Mail Online 2007; Criado 2014). Bob Bemer, el padre del código ASCII, nos describe en su crónica de la ponencia del Dr. Randell (que también lo comenta en el vídeo «Uncovering Colossus», véase The National Museum of Computing 2013) en junio de 1976 la cara de absoluto asombro de Maulchy al conocer la noticia de que no había sido él el inventor de la primera computadora (Bemer 2000).

Pasada la Segunda Guerra Mundial comienzan a crearse aparatos que cumplen con algunas pero no con todas las características que definen una computadora moderna[15]. No fue sino hasta principios de los años 50 que se lanza en Estados Unidos la primera computadora de uso comercial que tuvo cierto éxito, la UNIVAC I[16], creación de los mismos inventores de la ENIAC, J. Presper Eckert y John Mauchly. También durante esta época IBM lanzó la IBM 700 Series. Ambos equipos estaban dirigidos principalmente a las grandes empresas y a organismos públicos y se les empezó a llamar *mainframe computers*[17] (Swartzlander 2015; Grad 2015).

[15] Las nuevas "computadoras", es decir, las que reemplazan a las personas en la tarea de hacer cómputos, pueden *hacer cálculos* y *almacenar datos*, pueden *llevar a cabo una secuencia de operaciones* predeterminada, las cuales pueden depender de los resultados de cálculos anteriores (es decir, que sea *programable*), y *trabajan a velocidades electrónicas*. Anteriormente las operaciones que dependen del resultado de cómputos realizados en pasos previos las hacían los seres humanos a mano: bien fuera llevando una estiba de tarjetas perforadas de una máquina a otra para realizar cierto cálculo, o que el operador de la calculadora mecánica realizara un cálculo u otro dependiendo de si el resultado del cálculo anterior arrojaba un número negativo o positivo, por ejemplo. Según avanzó la tecnología se crearon memorias de almacenaje que operan a grandes velocidades, y la definición de uno de estos cuatro elementos que define lo que es una computadora se expandió: *programable* incluye también la noción de que la secuencia de operaciones que la computadora ejecuta, el programa, se almacena en estas memorias de gran velocidad, lo que facilita la alteración de los programas (la ENIAC, por ejemplo, se programaba usando cables y tomaba días reprogramarla). Esta posibilidad también sienta las bases de los lenguajes de alto nivel que discutiremos más adelante (Swartzlander 2015).

[16] "Historians might quibble, but a credible case can be made that the 'computer age' we live in today began at 8:30 Eastern Standard Time on the evening of November 4, 1952" (Boyer 2004, p. 8). La Universal Automatic Computer, de cuyo nombre se crea el acrónimo UNIVAC, estaba concebida como un aparato polivalente que funcionaría con cualquier aplicación que se le programara (de ahí el "universal"). "Automatic computer" subraya su capacidad de realizar cómputos de manera automática, sin intervención humana (Swartzlander 2015). La máquina debutó ante el público estadounidense durante las elecciones presidenciales de 1952, cuando predijo una apabullante victoria para Dwight D. Eisenhower, una probabilidad 100 a 1, a eso de las 8:30 de la noche. Sus ingenieros no se sentían confiados en los resultados que arrojaba la máquina (las encuestas habían indicado una votación muy cerrada) y no fue sino hasta pasada la medianoche, cuando tenían más datos disponibles, que anunciaron la predicción. "It was right. We were wrong. Next year we'll believe it", se lamentaba Art Draper, el representante de Remington Rand, empresa fabricante de la computadora (Henn 2012).

[17] No se sabe exactamente por qué se les llama de esta manera. Es muy posible que se deba a que los *frames* (diseñados para las centralitas telefónicas), aparatos donde antiguamente se colocaban los circuitos de

1.3 Los primeros lenguajes de programación

Todas las computadoras electrónicas, desde las más antiguas hasta las más modernas, se comunican internamente por medio de impulsos eléctricos que, para fines de programación, se convierten en lenguaje binario representado por ceros y unos: cero significa que no hay corriente, uno que sí[18]. En los primeros equipos electrónicos programables disponibles la codificación se hacía usando un lenguaje muy cercano a este sistema binario, mediante códigos breves para representar la secuencia de acciones que debía llevar a cabo el programa. Este lenguaje se conoce como lenguaje de máquina y, por su cercanía al lenguaje binario de las computadoras, se clasifica como lenguaje de bajo nivel. Al otro extremo, están los lenguajes de alto nivel o lenguajes más cercanos al lenguaje humano, que discutiremos más adelante.

Es preciso comprender que, por la proximidad de los lenguajes de bajo nivel al lenguaje de máquina, programar en lenguajes de bajo nivel era extremadamente complejo y presentaba todo tipo de limitaciones: el programador tenía que ser un experto, era responsable de mantener

procesamiento y memoria *centrales* del equipo (de ahí *main frame*), se colocaban aparte del resto de los componentes. Con el tiempo se convirtió en sinónimo de "computadoras grandes" (Computer History Museum 2011k).

[18] Inicialmente los equipos de computación se valían de medios mecánicos y físicos para llevar a cabo los cálculos matemáticos. La conceptualización de la idea del uso de sistemas electrónicos y de un lenguaje binario para fines del proceso de computación en una computadora está rodeada de controversias. En 1939, el Dr. John Vincent Atanasoff solicitó una beca a la universidad donde enseñaba, Iowa State College, para obtener fondos para materiales y para contratar un asistente graduado a fin de experimentar en la creación de una idea que había tenido en el invierno de 1937 sobre una máquina que usara sistemas electrónicos para realizar con rapidez y exactitud cálculos científicos. Así es como contrata a un alumno graduado, Clifford Berry, con quien trabaja en la creación de la primera computadora electrónica digital de lenguaje binario conocida como la Atanasoff Berry Computer o ABC. Debido a circunstancias históricas se decía que la primera computadora electrónica era la Electronic Numeric Integrator and Computer, mejor conocida como ENIAC, creación de John W. Mauchly y J. Presper Eckert. Luego de una complicada batalla en corte, en octubre de 1973 el tribunal de distrito de Minnesota dictó sentencia: "Eckert and Mauchly did not themselves first invent the automatic electronic computer, but instead derived that subject matter from one Dr. John Vincent Atanasoff" (JVA Initiative Committee y Iowa State University 2011; Parker y Duffin 1990; Computer History Museum 2011b).

los programas lo más compacto posible[19]; debía saber en qué espacio de la memoria colocaba qué dato, indicarle a la computadora cómo hacer los cálculos y dónde poner los resultados, tener en mente la longitud de los datos y los punteros que indican la ubicación de estos datos, entre muchas otras consideraciones[20]. De igual forma, como la configuración interna de cada máquina era particular a cada aparato[21], la codificación en lenguaje de máquina era específica de la máquina en donde se codificaba: el código no era "transportable" a otro equipo.

[19] Las limitaciones en capacidad de almacenamiento de programas y de datos en memoria de trabajo, donde se ejecutan los programas, presentaban un problema a la hora de ejecutar un programa. "En esa época las computadoras tenían memoria [de trabajo] muy pequeña (cerca de 15Kb [kilobits], era común medir la capacidad en bits), eran lentas y tenían sistemas operativos muy primitivos (si tenían alguno)" (Agay y Vajhoej 1998, traducción nuestra). Actualmente la memoria de las computadoras se mide en *bytes*, que son conjuntos de ocho *bits*, acrónimo de *BInary digiTS*. Compárese la capacidad de memoria en ese momento con las medidas de capacidad de memoria de las computadoras actuales, que se miden en gigabytes (GB). La computadora portátil que estoy usando para preparar este documento tiene 8GB de memoria, es decir, 67 108 864 kilobits, unos 4,4 millones de veces más grande que las de antes. Para que se ejecute, un programa debe estar cargado en la memoria de trabajo y necesita espacio adicional en ella para colocar datos y hacer cálculos. De ahí la importancia del tamaño de esta memoria para los programadores de la época.

Un sistema operativo es "[a]n integrated collection of routines that service the sequencing and processing of programs by a computer". Se encarga de proveer servicios básicos como asignación de los recursos disponibles en el sistema, programación de los procesos (*scheduling*), control de *input/output* y gestión de datos (Alliance for Telecommunications Industry Solutions 2011, s. v. operating system). En otras palabras, los sistemas operativos coordinan los procesos internos de las computadoras, específicamente, la lectura, el orden de procesamiento y la presentación de datos. En los comienzos del desarrollo de las plataformas computadorizadas no existían estos tipos de servicios internos en los equipos. Quizás el sistema operativo que tuvo más resonancia en el ámbito de las computadoras es DOS (*Disk Operating System*), el sistema operativo que usaban las primeras computadoras personales manufacturadas por IBM.

[20] Para tener idea de la complejidad de este proceso y una visión histórica del desarrollo de las computadoras, véase el maravilloso discurso de apertura que ofrece Grace Murray Hopper, pionera en el campo de la programación, durante el primer congreso de *History of Programming Languages*, celebrado en Los Ángeles, California del 1 al 3 de junio de 1978 (Hopper 1981).

[21] Como vimos antes, muchas máquinas se diseñaban específicamente para una empresa o para ejecutar una tarea específica.

Debido a la necesidad de hacer más accesible la programación y por tener la posibilidad de crear programas que se pudieran ejecutar en distintas máquinas, comienza a desarrollarse la idea de los lenguajes compilados[22]. El concepto era bastante simple: crear un metalenguaje específico, partiendo del inglés, que recoja las estructuras semánticas que controlan la lógica de los procesos que se ejecutan en un programa, y que este metalenguaje, llamado *código fuente* y que tiene su propia sintaxis y vocabulario, genere el código de instrucciones reconocibles por el equipo. Los compiladores toman el código fuente y crean un código objeto optimizado usando este metalenguaje. El compilador y optimizador, creado para generar código objeto para una máquina específica, se encargará de verter el código en el lenguaje binario que pueda ser ejecutado en la computadora para la cual fue creado. Para que el programa se pueda transportar a otra máquina no hace falta (en principio) modificar el código fuente del programa, sino que lo único que se necesita es que el compilador, que en efecto funciona como traductor[23], pueda convertir el código fuente al código de lenguaje binario específico para esa nueva máquina. Los compiladores se convierten en el puente entre el código fuente y el código objeto para una máquina en particular (véase Figura 1.1). Su escritura la rigen la sintaxis del lenguaje de alto nivel y el vocabulario que se ha creado, y es estándar, pero el código objeto que genera está particularmente destinado a ejecutarse en el equipo para el cual fue desarrollado (Computer History Museum 2011h, 2011j; International Business Machines 2014a).

[22] Grace Murray Hopper fue la creadora del primer compilador, llamado A-0.

[23] Traductor en este caso, en contraste con los lenguajes interpretados, como lo es el *hyper text markup language* o HTML. Los navegadores de Internet, por ejemplo, son intérpretes de HTML.

El primer lenguaje de alto nivel que se crea se conoce como FORTRAN y lo desarrolló un grupo de programadores de la empresa IBM liderado por John Backus[24] en 1959. FORTRAN, acrónimo de *FORmula TRANslating [System]*, está dirigido principalmente a la creación de programas para científicos e ingenieros. Su diseño estaba optimizado para generar un código objeto eficiente y para reducir el tiempo de programación, puesto que para esa época el costo de pagar a programadores era tan alto como el costo del equipo. También se buscaba que el lenguaje de programación fuera similar al proceso de pensamiento que usan los científicos, a fin de que pudieran codificar de manera más directa la notación matemática de sus calculaciones (Irani y Brennan 2001).

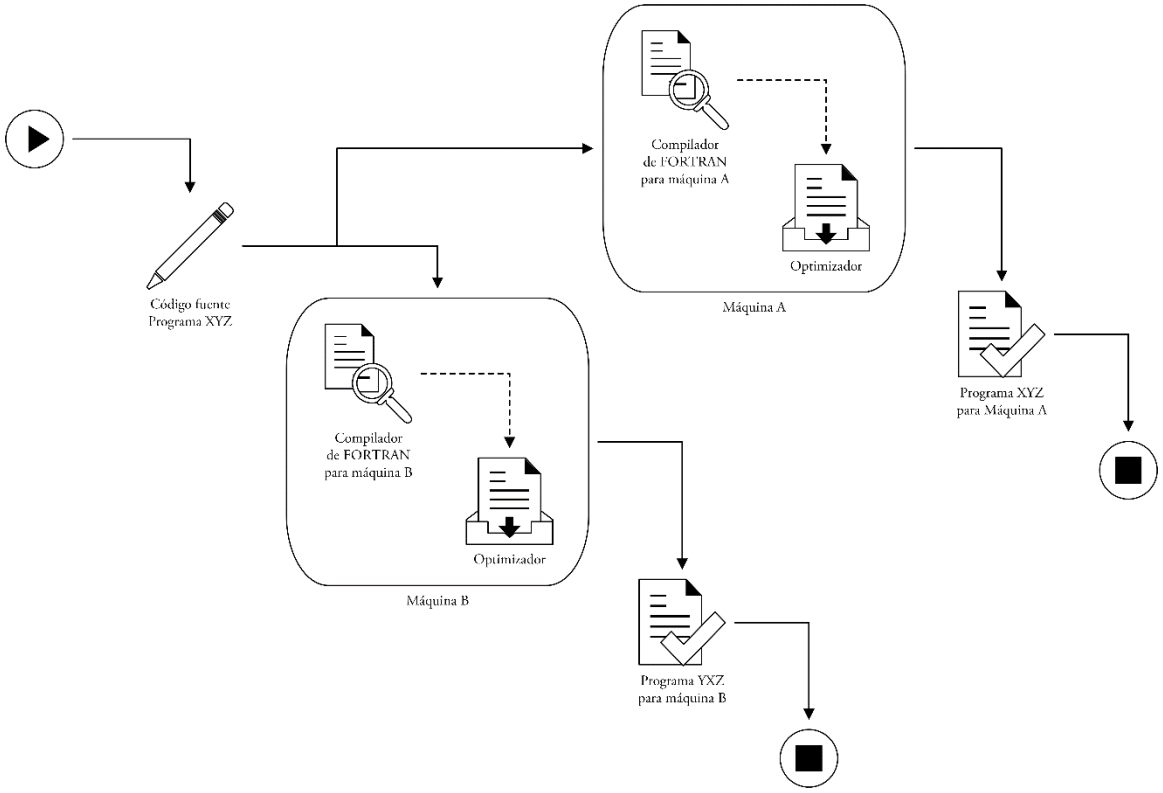

Figura 1.1 - Los compiladores
Los compiladores están creados para cada máquina en específico, pero deben generar programas que arrojen los mismos resultados en todas ellas. En principio, un programador crea un solo código fuente en FORTRAN llamado «Programa XYZ» que, si se compila en la «Máquina A» crea el «Programa XYZ para Máquina A» y si se compila en la «Máquina B» creará un «Programa XYZ para Máquina B». Aunque el resultado de la ejecución de ambos programas debe ser el mismo, el programa creado para la Máquina B no funciona en la Máquina A ni viceversa.

[24] Backus ofrece un recuento histórico del desarrollo de este lenguaje durante su ponencia en el primer congreso de *History of Programming Languages* que mencionamos en la nota 20. Véase (Backus 1981).

Casi al mismo tiempo, en marzo de 1959, Mary K. Hawes, de la *ElectroData Division* de la empresa Burroughs Corporation, convocó a una reunión entre manufactureros y usuarios para desarrollar las especificaciones de un "*common business language for automatic digital computers*" (citado en Sammet 1981, p. 200). Este lenguaje estaría dirigido a programar aplicaciones relacionadas con la contabilidad[25], como por ejemplo nómina, control de inventario y registro de débitos y créditos, que además pudieran ejecutarse en distintas marcas de computadoras. Del comité de trabajo que se creó a partir de esta convocatoria surgieron las especificaciones de COBOL (*COmmon Business-Oriented Language*), basado en AIMACO[26],

[25] El concepto de aplicaciones en el ámbito de desarrollo de sistemas informáticos no se relaciona en nada con el sentido de *apps* que se usa en la actualidad. *Aplicaciones* se refiere a un conjunto de programas que llevan a cabo una o más funciones en una empresa. Por ejemplo, la aplicación de pago de nómina incluye programas para ingresar datos de horas trabajadas, programas de cálculos de ingresos, retenciones y deducciones, acumulación de días por vacaciones y enfermedad, e impresión de cheques, entre otros. También puede interactuar con otros sistemas, como por ejemplo, con el sistema de contabilidad y el sistema de gestión de recursos humanos.

[26] AIMACO es un acrónimo de *Air MAterial Command* y era un lenguaje para negocios que estaba en proceso de desarrollo por la *US Air Force* (Sammet 1981, pp. 200, 201).

FLOW-MATIC[27] y el *Commercial Translator*[28] de IBM. Ya para diciembre de 1960 el compilador se había probado con éxito en dos máquinas distintas, la UNIVAC II y la RCA 501 (Smithsonian Institute 2014). COBOL introdujo varios principios novedosos: es un lenguaje legible porque sigue la sintaxis del inglés; es modular, por lo tanto, transportable a distintas máquinas con modificaciones mínimas; y "entiende" objetos implícitos de manera parecida a como los humanos entendemos el idioma[29] (Irani y Brennan 2001). Durante la década de 1970 se convirtió en el lenguaje *de facto* de las empresas y del gobierno estadounidense, y aún en la actualidad millones de transacciones bancarias se procesan usando COBOL (Smithsonian Institute 2014; Irani y Brennan 2001).

[27] FLOW-MATIC es un lenguaje de programación diseñado para que fuera parecido al inglés y fue desarrollado por Hooper (véase nota 22). Estaba ya en uso en la computadora UNIVAC, fabricada por la empresa Remington Rand (véase nota 16). Hooper estaba convencida de que los programas de computadora deberían ser fácilmente comprensibles y que deberían usar comandos en inglés (o en cualquier otro idioma) puesto que las personas que trabajaban en el campo de *data processing*, como se le llamaba al campo de la informática empresarial en esa época, no entendían bien de símbolos ni de matemáticas (Irani y Brennan 2001). "Working with the people that I had worked with in data processing, I found that very few of them were symbol-oriented; very few of them were mathematically trained. They were business trained, and they were word-manipulators rather than mathematics people. There was something else that was true of the data processing world: whereas in the mathematical-engineering world, we had a well-defined, well-known language; everybody knew exactly what—if you said "SIN" everybody knew it meant "sine"—they knew exactly what you meant, whether you were in Berlin or Tokyo, or anywhere in between. It was fully defined, already there. Everybody knew it. But there wasn't any such language for data processing" (Hopper 1981, p. 16).

[28] El *Commercial Translator* de IBM también usaba un vocabulario y sintaxis parecidos a los del inglés, pero aún no se había implementado (Smithsonian Institute 2014).

[29] En este trozo de código podemos ver como el lenguaje "entiende" de manera implícita la referencia a `X` en `IF X=Y MOVE A TO B; IF GREATER ADD A TO Z; OTHERWISE MOVE C TO D`. En el código anterior se infiere lo que hemos añadido entre paréntesis: `IF X=Y` (then) `MOVE A TO B;` (else) `IF` (`X` is) `GREATER` (than `Y` then) `ADD A TO Z; OTHERWISE` (if `X` is neither equal to `Y` nor greater than `Y` [en otras palabras, si `X` es menor que `Y`] then) `MOVE C TO D` (Irani y Brennan 2001).

1.4 Revoluciones

Impulsadas por la investigación y por las inversiones que se hicieron luego de la Segunda Guerra Mundial y durante la Guerra Fría, aparecieron varias empresas que entraron a competir en el mercado de las computadoras. La empresa principal que sentó las pautas en la industria fue IBM y su liderazgo en el ámbito de las *mainframes* continúa hasta el día de hoy[30]. Las minicomputadoras han dejado prácticamente de existir[31], pues han sido reemplazadas por los servidores[32], pero en su momento satisficieron las necesidades de varios mercados que estaban desatendidos. Las computadoras personales han sido la gran puerta a la era de las comunicaciones y han permitido la democratización de esta tecnología. Todas estas tecnologías representan diversas plataformas sobre las que se desarrollan los programas. Veamos cuáles son, de dónde surgieron y hacia dónde se dirigen.

1.4.1 La revolución de las *mainframes*

Para 1963 las empresas que competían en el mercado de la época, Burroughs, Honeywell, Remington Rand, Control Data, General Electric e IBM, tenían varias líneas de productos disponibles para toda una gama de clientes. El problema más grave era que, a pesar

[30] "[…] the major companies of the computer industry were often referred to as Snow White and the Seven Dwarfs. The Seven Dwarfs were Burroughs, Control Data, General Electric, Honeywell, NCR, RCA, and Sperry Rand. Of the estimated $10 billion worldwide inventory of installed computers in 1964, the Dwarfs had produced about 30 percent, and IBM had produced the rest. Five years later IBM's worldwide inventory had increased more than threefold to $24 billion and that of the Dwarfs had increased by about the same ratio to $9 billion" (Pugh 2015).

[31] Posiblemente en parte por causa de la soberbia del mismo que vio la oportunidad de crear las minicomputadoras, el presidente de DEC, Kenneth Olsen (véase el apartado "La mini(falda/)computadora se cuela en la informática" en la página 35), que prohibió el uso del término *personal computer* en la empresa y dijo a finales de la década de 1970 que "the personal computer will fall flat on its face in business" (Saxenian 1996, p. 100).

[32] Que no son más que computadoras personales de gran capacidad de procesamiento, almacenaje y memoria.

de que eran máquinas excelentes, presentaban incompatibilidad entre los distintos modelos que ofrecía una misma empresa. "IT customers who wanted to move up from a small to a larger system had to invest in a new computer, new printers and storage devices, and entirely new software, most of which had to be written from scratch", lo que representaba una carga económica onerosa para cualquier negocio en crecimiento que se viera en la necesidad de ampliar sus capacidades de computación (Boyer 2004, p. 10).

El 7 de abril de 1964 IBM anunció la IBM System/360, una familia de computadoras que sentó pautas en la creciente industria de las computadoras. Desarrollar estos productos representó para la empresa un enorme riesgo económico[33], sin embargo, luego de dos años de tropiezos y atrasos, lograron lanzar al mercado una línea completa de modelos de *hardware* intercambiable con prestaciones desde las más básicas hasta las más avanzadas a los cuales se les podían conectar más de 40 equipos periféricos[34] que funcionaban con dos sistemas operativos[35].

[33] "Fortune magazine famously called the System/360 a […] US$30 Billion gamble (in 2005 US$). It was an extraordinary expression of confidence. The task merely involved concurrently designing a comprehensive, enduring, and extendable architecture; designing and building factories to produce and test state-of-the-art components; designing and building factories to produce and test a family of processors and related I/O products; designing, creating, and testing a new operating system, along with key components and utilities; plus hiring and training a new sales and support organization and doing it all while coordinating activities and resources on a worldwide basis in a time before the easy availability of cheap, high bandwidth communications" (International Business Machines 2008, sec. 1960s). Fred Brooks, Jr., quien estuvo a cargo de desarrollar el proyecto, lo cuenta desde el punto de vista de la experiencia: "[…] You kill six product lines, all you have. You leave them open to the depredation of competitors […]. And you trust that this one that you are putting out that is totally untried is going to hold all that customer base and continue to grow on an exponential basis. That takes executive boldness of the first order and we had it in Tom Watson, Jr. [el presidente de IBM en ese momento] And that, I think, is an impressive testimony. [Riéndose dice:] He later said he'd never do it again" (Computer History Museum 2011d).

[34] Los equipos periféricos (*peripheral equipment*) son las impresoras, los discos y cintas de almacenaje magnético, los terminales, los lectores de tarjetas perforadas, el equipo de comunicación, etc. que se conectan al *mainframe* y expanden su funcionalidad.

[35] La idea original era crear un solo sistema operativo (para la definición de sistema operativo véase nota 19), Operating System/360 (OS/360), para toda la gama de equipos, pero la memoria limitada que las máquinas de las gamas inferiores tenían no permitía que este sistema se pudiera almacenar en ellas. Por ello se diseña el Disk

Los modelos de la gama más baja también eran capaces de emular el modelo 1401, uno de los equipos antiguos de la empresa más exitosos que se estaban viendo amenazados por la competencia[36] (Pugh 2015). La jugada le salió bien a IBM y en la actualidad la *mainframe* sigue siendo el equipo de computadora preferido por las grandes empresas para aplicaciones de misión crítica (*mission-critical applications*)[37] debido a las cuatro fortalezas fundamentales que ofrece esta plataforma: a) fiabilidad, disponibilidad de servicio y facilidad de mantenimiento[38];

Operating System/360 (DOS/360, que no tiene nada que ver ni se debe confundir con el DOS de las computadoras personales, véase nota 19), un sistema operativo más compacto que IBM esperaba desapareciera en pocos años. Sin embargo, no fue así. DOS/360 se convirtió a principio de los 1970 en DOS/VS (Disk Operating System/Virtual Storage) y ya para finales se ampliaban sus prestaciones en la versión DOS/VSE (Extended). Para los 1980 perdió su prefijo y ganó un nuevo sufijo, VSE/SP (Systems Package). Durante los siguientes 30 años este sistema vio más cambios de nombre y ampliación de prestaciones. Resulta asombroso que la versión 6.1 de z/VSE, como se conoce este sistema operativo en la actualidad, está disponible desde noviembre de 2015. Todavía, bajo este ambiente, se ejecutan las mismas aplicaciones que se desarrollaron en los primeros tiempos de las *mainframes*. La estrategia que IBM adoptara en 1964 todavía tiene validez en la actualidad. "These newest releases confirm our successful z/VSE "**PIE**" strategy: **P**rotect existing investments, **I**ntegrate with other systems, and **E**xtend for new workloads" (International Business Machines 2008).

[36] En diciembre de 1963, Honeywell anunció el "Liberator" un traductor de programas que permitía que los programas del IBM 1401 ejecutaran en su modelo H-200, una opción mucho más barata que el 1401. En dos meses sus vendedores reportaban que 196 de sus clientes se habían cambiado a esta máquina. Hacía ya un tiempo que los ingenieros de IBM estaban experimentando con simuladores, unos programas que hacían creer a otros programas que estaban ejecutándose dentro de un modelo de máquina antiguo, cuando en realidad estaban ejecutándose en un equipo más moderno. Usando circuitos especialmente diseñados, lograron, sin la necesidad de un programa, crear un ambiente que "emulaba" este modelo más antiguo. No solo los programas viejos podían ejecutarse sin tener que modificarlos al portarlos a máquinas nuevas, sino que ejecutaban a mayor velocidad. De esa manera, facilitaban el cambio a los clientes de un sistema más viejo a uno más nuevo y les dieron a sus vendedores una poderosa herramienta de venta (Pugh 2015, 1995, pp. 273-275).

[37] "A mission critical application is any software application that is essential to the organisation to perform its key business functions. […] The loss or stoppage of a mission critical application may have a negative impact on the organisation, together with possible legal or regulatory repercussions. […It's] vital to take care of any mission critical applications with the utmost care, and ensure that data is adequately protected and available at all times" (Thejendra 2014, p. 89).

[38] Estos tres factores siempre han sido importantes en el ámbito del procesamiento de datos y garantizan el funcionamiento continuo del equipo, es decir, que el servicio que ofrece la máquina esté disponible siempre y

b) seguridad[39]; c) escalabilidad[40] y d) compatibilidad continua[41] (International Business Machines 2013f). En la actualidad hay muy pocas empresas (Fujitsu y Unisys, por ejemplo) que compiten en la plataforma *mainframe*, donde IBM es el líder indiscutible del mercado[42].

IBM lanzó su último modelo de *mainframe* en enero de 2015, el modelo z13, cuyas prestaciones dan indicio de una redefinición de lo que significa *mainframe* en la actualidad. La imagen de la pantalla verde[43] donde había que introducir comandos crípticos para ejecutar

sin interrupción. Se conocen en la industria como RAS (*Reliability, Availability, and Serviceability*) (International Business Machines 2013f).

[39] La seguridad garantiza que se restringe el acceso a datos por parte de usuarios no autorizados, pero de igual manera los hace disponibles a quienes sí están autorizados. El equipo permite compartir datos entre gran cantidad de usuarios y, a la vez, los protege (*Idem*).

[40] La escalabilidad es la capacidad de adaptar el sistema según crece la demanda de los servicios que ofrece (*Extend for new workloads* de la estrategia PIE, véase nota 35 y la importancia de la emulación en la nota 36). El diseño modular de estos equipos permite que, de manera continua y sin interrumpir los servicios al negocio, se añadan o reemplacen prestaciones como procesadores, memorias y discos (*Idem*).

[41] La capacidad de poder seguir usando los programas que fueron creados en los modelos anteriores (IBM la llama *continuing compatibility*, que es lo mismo que la retrocompatibilidad) es importantísima. Las empresas han invertido grandes cantidades de dinero en crear las aplicaciones que se ejecutan en estas máquinas y no pueden darse el lujo de descartarlas solo porque necesitan cambiar de equipo. La idea de la compatibilidad continua comenzó con la IBM System/360 y siempre ha sido parte de la filosofía de desarrollo y *marketing* de la empresa (*Protect existing investments* de la estrategia PIE, véase nota 35) (*Idem*).

[42] Dice Shields (2014) que Vox reporta que el 90% de las aplicaciones para *mainframe* usan sistemas operativos nativos (que no son clonados, los que vende la empresa) IBM y que dichas aplicaciones se ejecutan en equipos marca IBM. También explica que BMC Software, un proveedor de aplicaciones para *mainframe*, encontró que el 90% de los usuarios de *mainframes* piensan que IBM es la solución acertada a largo plazo.

[43] Las pantallas *green screen* se asocian con aplicaciones que se ejecutan en *mainframes* por los terminales que se usaban para acceder estos equipos, que presentaban los datos en una pantalla generalmente color verde (o, en ocasiones, color ámbar). Todavía se siguen usando, pero con programas de emulación, que son programas que simulan a estos terminales de manera virtual y se ejecutan en una computadora personal. Es posible que mucha información que accedemos por medio de páginas web o teléfonos móviles en realidad sea un programa que se ejecuta en una pantalla *green screen* a la cual se le ha añadido una interfaz que la "maquilla" y la hace parecer al

programas y acceder a datos está casi en el olvido. Esta línea de *mainframes* acepta múltiples sistemas operativos dirigidos a preservar la inversión en programas desarrollados para ejecutar en equipos anteriores[44] y a ofrecer las más modernas prestaciones que ayudan a integrar funcionalidad indispensable en la actualidad, como por ejemplo, capacidad para procesar transacciones móviles, a estos antiguos programas. También añaden prestaciones modernas como la capacidad de crear decenas de servidores virtuales[45] en Linux[46], encriptación de datos

usuario como una moderna aplicación. Es el modelo de *wrapper* que explicamos más adelante en la nota 107 y en la página 104.

[44] Véase notas 35 y 36.

[45] La virtualización comenzó en la década de 1960 cuando se introdujo el concepto a fin de repartir los recursos que las computadoras tenían disponibles de una manera más eficiente. En esa época los programas funcionaban en modo *batch*, es decir, por tandas o lotes: uno colocaba una estiba de tarjetas perforadas (el programa) en el lector y esperaba a que la máquina lo ejecutara. El problema era que, a medida que las máquinas trabajaban más rápido, la velocidad de lectura de los métodos de insumo de la información (cintas de papel y magnéticas, tarjetas perforadas) que el programa procesaba no había mejorado, por lo que este equipo tardaba en hacer disponible la información al procesador. El resultado era que el procesador se quedaba sin trabajar esperando hasta que esa información "entrara". Muchas personas necesitaban usar las computadoras, con lo cual el tiempo de uso de máquina era como el oro y esto resultaba ineficiente. Para solucionar este problema, se creó un programa que creaba un espacio virtual para cada usuario, independiente de los espacios de los demás, que permitía a cada usuario interactuar (de ahí el "modo interactivo") con lo que parecía ser su propia máquina. Cada usuario tenía la impresión de que tenía disponible la máquina para sí, pero en realidad estaban todos compartiendo los recursos de la computadora (memoria, procesador). Así, cuando este programa, llamado "hipervisor", detectaba que cierto programa de cierto usuario estaba esperando por información para seguir procesando, le daba paso a otro programa para que ejecutara. De esa manera ese usuario aprovechaba el tiempo de espera del otro (Koziris 2015, pp. 3-4). Con la virtualización nace lo que llamamos servicios en línea (*on-line service*, probablemente por las líneas de cable que conectan el terminal con el equipo), un ambiente donde la computadora atiende al usuario al momento. La virtualización en esta época también se conocía como *time sharing*, véase nota 51.

[46] Linux es el sistema operativo FOSS (*Free and Open Source Software*) *par excellence* de los servidores de las páginas web en Internet. Forma parte de lo que se conoce como el paquete (*bundle*) LAMP: Linux, Apache (el servidor HTTP que se encarga de atender las solicitudes de páginas en internet), MySQL o MariaDB (el sistema de gestión de bases de datos de estos servidores y donde se almacena y se recupera la información) y PHP, Perl o Python (tres lenguajes de programación que se usan para expandir la funcionalidad de las páginas web).

estáticos y dinámicos[47], análisis y adaptación de volumen de trabajo[48] y manejo de *big data*[49].

1.4.2 La mini(falda/)computadora se cuela en la informática

Mientras IBM lidiaba con su éxito y desarrollaba estrategias para adelantarle el paso a la competencia, una pequeña empresa iba aprovechándose de las oportunidades de negocio que había en la gama inferior de las *mainframes*. Kenneth H. Olsen había formado parte del

[47] La encriptación de datos es un tema de extrema importancia por el tipo de datos que procesan y almacenan estos equipos (datos financieros, personales, médicos, de inteligencia empresarial o gubernamental). Este sistema posee procesadores diseñados para únicamente atender los procesos de encriptación de datos y puede encriptar y desencriptar a alta velocidad datos estáticos, es decir los que están almacenados dentro de la máquina, y datos dinámicos, que son los que se están transmitiendo desde y hacia la máquina (International Business Machines 2015).

[48] El volumen de trabajo que atienden estos sistemas es muy variable debido a que cada tipo de trabajo que procesan (*batch*, en línea, servidores virtuales, interfaces entre sistemas) tiene picos y valles variados que suceden en distintos momentos e independientemente uno del otro. Por ello, hace falta poder gestionar con facilidad y eficacia los recursos que dispone la máquina. Este *mainframe* tiene herramientas que monitorizan y gestionan automáticamente la distribución de los recursos a fin de optimizar su uso y dar prioridad a los procesos que la necesitan (*Idem*).

[49] Definir este concepto resulta complicado puesto que su uso ha ganado popularidad a partir de 2011, sobre todo por causa de la promoción de IBM y otras firmas que mercadean tecnología, y todavía las empresas no se ponen de acuerdo en qué exactamente quiere decir. *Big* hace que el tamaño resalte como característica principal, pero muchos autores (y según la fuente) tienden a definirlo como datos cuyas características son: volumen, variedad, velocidad, veracidad, variabilidad (y complejidad) y valor. Volumen se refiere a la magnitud de los datos, que se reportan en unidades como terabytes y petabytes; variedad se refiere a la heterogeneidad de los conjuntos de datos, que pueden ser estructurados (datos en bases de datos y hojas de cálculo, por ejemplo), no estructurados (texto, imágenes, audio, vídeo, medios sociales, datos de sensores) y semiestructurados (datos que no se conforman a estándares específicos como XML y JSON); velocidad está relacionada con la razón en que se producen y se pueden analizar dichos datos; veracidad (creación de IBM) tiene que ver con cómo se gestiona información poco precisa y que posee un grado de incertidumbre (por ejemplo, los sentimientos que expresa un cliente); variabilidad y complejidad (creación de SAS, empresa dedicada al análisis estadístico) hace referencia a que el flujo de estos datos está caracterizado por picos y valles y que estos proceden de un enorme abanico de fuentes que añade complejidad al proceso de análisis; valor (creación de Oracle) quiere decir que una característica del *big data* es que tiene en sí una densidad de valor baja, pero que una vez que se analiza, se convierte en datos con una densidad de valor alta (Gandomi y Haider 2015, pp. 138-139).

Proyecto SAGE[50], donde había aprendido los métodos de gestión y de manufactura que usaba IBM, y establece en 1958, junto con Harlan Anderson, la empresa Digital Equipment Corporation (DEC), que inicialmente se dedica a la manufactura de módulos de circuitos para otras empresas. En 1965 la empresa presenta la PDP-8 (*Programmable Data Processor*) a un precio sumamente atractivo, creación de sus ingenieros C. Gordon Bell y Edson de Castro que, comisionados por el Canadian Chalk River Nuclear River Lab, optaron por diseñarles una computadora polivalente programada para monitorizar uno de sus reactores en lugar de crear un controlador hecho a la medida para esos fines. Esta computadora servía para todo: para controlar las tiras de noticias en Times Square, para ofrecer un *time-sharing*[51] barato en la Universidad Carnegie Mellon, para hacer análisis de señales y hasta para controlar la iluminación del teatro que presentaba el musical *A Chorus Line*. Los modelos posteriores, con precios aún más atractivos, permitían que estos sistemas se incrustaran en máquinas más grandes, haciendo posible usarlas tanto en una oficina como procesadoras de datos, como en un hospital para monitorizar los equipos durante una operación o en una máquina para llevar a cabo cirugía del cerebro (Computer History Museum 2011c).

Cuenta Pugh que en esa época en que estaba de moda la minifalda, a alguien se le ocurrió llamar a estas máquinas minicomputadoras, y el nombre pegó. En sus comienzos, las

[50] El Proyecto SAGE (*Semi-Automatic Ground Environment*) fue uno de los primeros sistemas de defensa aérea de los EE.UU. y fue desarrollado por IBM en colaboración con el laboratorio Lincoln de MIT. Unas 23 localidades donde se analizaba la información recopilada y miles de estaciones que transmitían datos relacionados con el tráfico aéreo conformaron la primera gran red de computadoras que brindaba interacción con una máquina en tiempo real y usaba unas pantallas especialmente diseñadas que mostraban el resultado del análisis de dichos datos recopilados. El proyecto comenzó a principios de la década de 1950 y estaba operando a capacidad en 1965; algunas localidades aún estaban en servicio en 1982 cuando se descontinuó su uso (Computer History Museum 2011a; International Business Machines 2003e).

[51] *Time-sharing* es una forma de compartir el uso de una computadora. Múltiples usuarios se conectan a ella por medio de terminales y cada uno tiene la impresión de que tiene una máquina para su uso personal, pero en realidad lo que la máquina hace es compartir el tiempo de uso entre todos los usuarios atendiéndolos uno a uno. Es una forma de tener acceso a una computadora, que en esa época podría hasta considerarse un lujo. Véase virtualización en la nota 45.

minicomputadoras se usaron para registrar datos en experimentos científicos y controlarlos, y luego se usaron para controlar procesos y en las redes de telecomunicaciones (Pugh 1995, pp. 298-299). Muchas empresas entraron a competir en esta plataforma, pero la que se destacó sobre todas las demás fue Digital[52]. Sin embargo, cuando empiezan a salir al mercado las primeras computadoras personales, esta empresa y sus competidores fallan en reconocer la importancia de este desarrollo (véase nota 31), rehúsan adaptar sus sistemas y hacerlos abiertos para posibilitar la participación de terceros en la creación de programas[53], y, cuando deciden incursionar en este segmento, ya era demasiado tarde. DEC desaparece en 1998 cuando, irónicamente, la compra Compaq, empresa fabricante de computadoras personales (Saxenian 1996, p. 100).

Si las minicomputadoras atienden a los usuarios científicos y técnicos, las computadoras *midrange* —así las llama IBM— se crean para atender a las pymes de la época. Están dirigidas al mismo mercado de las minicomputadoras, pero en el ámbito empresarial. Comienzan con la línea System/3, lanzada en 1969, a la que le sigue la System/38 (1978), la System/36 (1986) y su más exitoso producto en este segmento del mercado: la Application Systems/400 (AS/400, lanzada en 1988). A la fecha de su lanzamiento contaba con más de 1000 aplicaciones y permitía ejecutar las que habían sido desarrolladas para los sistemas anteriores, facilitándoles a sus clientes la migración a la nueva plataforma (International Business Machines 2003b, 2003d, 2003c). En la actualidad, la AS/400 fue sustituida por los Power Systems, una gama de servidores diseñada para ofrecer prácticamente las mismas prestaciones que ofrecen los modelos

[52] Aparte de Digital, podemos mencionar las más destacadas, Data General y Hewlett Packard, la única sobreviviente en la actualidad.

[53] En los 1980, Digital, Hewlett-Packard, Data General, Perkin Elmer, General Automation y Computer Automation competían en el segmento de minicomputadoras. Estos manufactureros ofrecían para sus equipos su propio sistema operativo más FORTRAN y otros compiladores, lo que limitaba el crecimiento del mercado de desarrolladores de programas para esta plataforma (Grad 2015).

de *mainframes*. Funcionan con los sistemas operativos Linux, AIX e IBM i[54], todos en modo virtualizado.

1.4.3 Microrrevolución

A IBM se le pasó la revolución de las minicomputadoras, pero para la de las microcomputadoras o computadoras personales procuró ser el líder de la industria[55], sacando partido a su reputación como líder en servicio y calidad. El desarrollo de la tecnología de microcircuitos estaba muy adelantado ya para principios de la década de 1970. Intel había sido fundada en 1968 y se dedicaba a la producción de circuitos de memoria de acceso aleatorio (RAM). En 1969 la empresa japonesa Nippon Calculating Machine Corporation la contrata para que les diseñe y fabrique un chip[56] para una nueva línea de calculadoras electrónicas, pero en lugar de crear un chip para esos fines, la empresa decidió crear un conjunto de cuatro *microchips* polivalentes. Cada chip llevaba a cabo una función determinada: uno se encargaba de gestionar los sistemas de *input* y *output*, otro era para la RAM, otro era para la memoria de solo lectura (ROM) y el último era una unidad central de procesamiento, que se encargaba de realizar cálculos y coordinar las operaciones entre todos estos elementos electrónicos. Poco tiempo después, la empresa nipona quebró, pero Intel procuró comprar de vuelta los derechos

[54] *IBM i* es la última versión de lo que antiguamente era OS/400, el sistema operativo de la AS/400 original. Con él se facilita la migración de los clientes a la nueva plataforma. AIX es la implementación de IBM del sistema operativo UNIX.

[55] "We missed the minicomputer revolution completely. The sociological factors that said departments, when they could afford to get transistorized computers that you could afford on a department budget, wanted one that they could control instead of a glass house at the center of the corporation. We missed it clean. And so here came the Digitals, and the HPs and the Data Generals and so forth, and they had that field to themselves. IBM sputtered out a few things here and there but never really captured that. […] They were wildly successful [… but] I think with perhaps only with exception of HP, if that, they missed the microcomputer revolution. Now IBM having missed the minicomputer revolution, got back in on the microcomputer revolution. The hazards of success are very great and we've seen it in the computer field now with two revolutions" (Computer History Museum 2011d).

[56] "A chip, also known as a ***die*** or processor, is a microelectronic device that can process information in the form of electrical current traveling along a ***circuit***" (Intel Corporation 2012, p. 6, énfasis en el original).

de estos chips. El *Micro Computer Set* MCS-4, como se le conocía a este conjunto de cuatro componentes, fue el primer sistema de microprocesadores disponible para uso comercial y el que dio inicio a toda la familia de microprocesadores que sirvieron de base para desarrollar no solo la IBM-PC, sino también su contraparte, la Mac (Computer History Museum 2011f).

Sin embargo, las computadoras personales no nacen de las grandes empresas, sino de los grupos de entusiastas en electrónica que se reunían para compartir (o más bien alardear sobre) sus creaciones y aplicaciones de estos nuevos aparatos electrónicos que estaban siendo creados para grandes empresas, pero que las personas de a pie podían usar para construir sus propias computadoras. Luego de asistir a la primera reunión del *Homebrew Computer Club*, a Steve Wozniak se le ocurrió que podía usar el terminal que había creado para "hablarles" a otras computadoras de la Arpanet para "comunicarse" con el microprocesador y su memoria. ¿Por qué lo hizo? Porque, en primer lugar, le permitía tener una computadora; en segundo lugar, podía no solo realizar las simulaciones que necesitaba hacer para su trabajo, sino que también podía jugar a juegos; y, en tercer lugar, les podía mostrar a los compañeros del club que no era necesario gastar miles de dólares para tener una computadora, sino que con unos cuantos chips baratos podían construir una. Así surgió la Apple I, predecesora de la Apple II, la computadora con más éxito hasta la llegada de la IBM-PC (Computer History Museum 2011i). En el próximo capítulo veremos que el surgimiento de la localización está muy ligado a las computadoras personales y profundizaremos un poco más en la microrrevolución.

1.5 Actualidad

En cuanto a equipos, la actualidad nos presenta un panorama amalgamado donde se funden las divisiones que a finales del siglo pasado estaban bien demarcadas. Las *mainframes* podrían parecer PCs superdotadas y las PCs alcanzan capacidades de computación y almacenaje enormes. Aunque los lenguajes de programación han proliferado grandemente, si miramos las principales plataformas de *hardware* que se usan en la actualidad (sin considerar los equipos móviles), habrá poco más de veintitantos lenguajes de uso común.

1.5.1 *Hardware*

Las *mainframes* siguen siendo las plataformas más fiables para aplicaciones *mission-critical*[57] y continúan prestando sus servicios de la manera tradicional, es decir, por medio de aplicaciones *green screen,* aunque muchas de ellas adaptadas a aplicaciones cliente-servidor[58] para poder brindar al usuario una interfaz gráfica más *user-friendly*. Aun así, las *mainframes* ya no se limitan a ese tipo de aplicaciones y cuentan con prestaciones de integración avanzadas que permiten acoplar estos programas y sistemas altamente fiables a las nuevas tecnologías, en especial las tecnologías móviles y de internet. También son capaces de simular fincas de servidores autogestionables que se crean y se activan o se desactivan y desaparecen según la demanda de servicios informativos oscila entre picos y valles. Pueden gestionar, tomar decisiones y responder al *big data*, a fin de atender al momento las necesidades cambiantes de los clientes. Son capaces, además, de aceptar sistemas de código abierto como, por ejemplo, Linux. La gama intermedia, las minicomputadoras y *midrange*, ha desaparecido para todos los efectos y ha sido suplantada por macroservidores. Estos macroservidores, que no son más que los superhijos de las computadoras personales, están compuestos (si lo miramos de forma muy simplista) de los mismos equipos y prestaciones básicos que nos ofreciera Intel en su MCS-4 a principios de la década de 1970. La diferencia es la capacidad de computación que se ha multiplicado hiperexponencialmente. Las computadoras personales de principio de 1980 han sufrido esta misma transformación debido a la creación de *microchips* de alta

[57] Véase nota 37.

[58] Las aplicaciones cliente-servidor están diseñadas bajo la arquitectura (véase el apartado 2.3.3.1) con ese mismo nombre. Esta arquitectura es un modelo de comunicación distribuida donde el programa que requiere un servicio, es decir, el cliente, solicita por medio de una red cierta información al servidor. En este modelo no solo se distribuye la información entre el cliente y el servidor, sino que también se distribuye el trabajo de procesar dicha información. El protocolo de transferencia de hipertexto (HTTP), que contsntemente usamos al acceder a la web, es un ejemplo de arquitectura cliente-servidor (Janssen 2016, s. v. client-server model).

capacidad y velocidad de procesamiento, capaces de almacenar cada vez más circuitos de procesamiento de información dentro de espacios cada vez más pequeños[59].

La internet es el hilván que vincula todas estas plataformas y obliga a la convergencia entre ellas. Es generatriz de la plataforma más nueva y omnipresente de nuestros tiempos, la plataforma móvil, y añade presencia de aún más elementos si consideramos la internet de las cosas (IoT)[60]. Por un lado, la plataforma móvil permite democratizar el acceso a la internet, dada la facilidad que ofrece de interconectar aparatos a la red con relativa facilidad[61]. Por el otro, la internet de las cosas refuerza la importancia de brindar estos servicios informativos en formatos relevantes para las partes receptoras, es decir, usando elementos gráficos y textuales

[59] Para poner el desarrollo de la tecnología de microprocesadores en contexto, el microprocesador Intel 4004, creado en 1971, contenía 2300 transistores, un reloj interno que operaba a 108 kHz (el reloj interno es el elemento coordinador de los procesadores, funciona como si fuera un director de orquesta marcando el tempo; mientras más rápido va, más rápido se procesan los datos dentro del chip) y fue manufacturado usando tecnología en la escala de 10 micrones (micrometros). El Intel Core de tercera generación lanzado en 2012 contiene 1400 millones de transistores, opera a una velocidad interna de 2,9 GHz (2 900 000 kHz) y su tecnología de manufactura es en la escala de 22 nm (0,022 micrones) (Intel Corporation 2012, pp. 28-29). Cuando en 1965 Gordon Moore, uno de los cofundadores de Intel, predijo que "[…] the number of transistors on a piece of silicon would double every year […]" no estaba lejos de la realidad. Sin embargo, en 1975 "[…] Moore updated his prediction that the number of transistors that the industry would be able to place on a computer chip would double every couple of years. […]". El nivel de reducción de espacio en los circuitos está llegando a los electrones mismos, bordeando el límite físico. La nueva frontera son las supercomputadoras cuánticas (véase Chow 2015).

[60] La internet de las cosas (IoT, por sus siglas en inglés) busca conectar toda suerte de aparatos a la internet: "[…] everyday physical objects will be connected to the Internet and be able to identify themselves to other devices […]" (Janssen 2016, s. v. internet of things (IoT)) a fin de proveer un entorno que responda a la información que estos aparatos generan. Estas respuestas, que implican un flujo de datos e información continuo entre los aparatos mismos y lo que (o quienes) esté(n) en el entorno, pueden aplicarse a nivel de "body area network" (BAN; por ejemplo, una camisa inteligente, gafas inteligentes), "local area network" (LAN; por ejemplo, un contador eléctrico inteligente como parte de la interfaz en el hogar), "wide area network" (WAN; por ejemplo, sistemas telemáticos, ITS o servicios de tecnología de la información, el coche inteligente) y "very wide area network" (VWAN; la ciudad inteligente con servicios de gobierno-e omnipresentes y no atados a localidades específicas) (IoT Council 2015). El artículo "What is the internet of things?" (Kobie 2015) explica algunas posibilidades de la IoT.

[61] Google tiene un proyecto para desarrollar una red mundial de WiFi usando globos que flotan en la estratósfera y funcionan como portales a la red. Véase http://www.google.com/loon/

que se conformen a las necesidades lingüísticas de los receptores. Así, observamos cómo el futuro de la informática se conforma como un espacio necesariamente plurilingüístico y pluricultural.

1.5.2 *Software*

En cuanto a los lenguajes de programación, la evolución ha sido exponencial, y a pesar de que en la actualidad los lenguajes nuevos son pocos, sigue habiendo creaciones nuevas. De unos cuatro lenguajes que había a principios de la década de 1960, Wikipedia enumera un total de 695 lenguajes de programación (Wikipedia 2016), sin considerar toda la gama de versiones y variaciones. Por otro lado, Kinnersley ([sin fecha]) ha compilado una lista de aproximadamente 2500 lenguajes y variaciones en su sitio web. Aun habiendo esta enorme cantidad de lenguajes, si examinamos las plataformas principales de *hardware*, la cantidad de lenguajes utilizados, sin tomar en cuenta las versiones o variaciones, suman una veintena. En el capítulo 3 describiremos los lenguajes que están en uso en cada una de las plataformas que hemos discutido a lo largo de este capítulo, así como también las prestaciones o posibilidades de internacionalización que cada uno de estos lenguajes ofrece.

1.6 Surgimiento de las estrategias de análisis y diseño

Las estrategias de análisis y diseño de sistemas informáticos se desarrollaron en el ámbito académico y esto comenzó varias décadas después de que se iniciara el desarrollo de las computadoras. La Computación (Ciencia de Cómputos) "se desarrolló como área de investigación académica […] en el contexto de las ciencias y las matemáticas" (Davis 2006, p. 14, traducción nuestra). Los investigadores académicos en computación desarrollaron equipos computadorizados diseñados para desencriptar códigos a principios de 1940, y ya para finales de esta década muchas universidades estaban desarrollando sistemas a fin de probar diversas ideas de diseño. Para 1951 ya estaba disponible la UNIVAC I en los Estados Unidos, y la Lyons Tea Company y la Universidad de Cambridge habían hecho funcionar a LEO, la primera computadora que se usó con fines comerciales en el Reino Unido. Existían ya varias organizaciones científicas dedicadas a la computación, suficientes como para crear una

federación internacional de sociedades dedicadas a la computación. En 1960, con el auspicio de la Unesco, se crea la *International Federation for Information Processing* (IFIP) (*Idem*).

Davis estima que existen tres factores principales para que los sistemas de información no hayan sido considerados hasta bastante más tarde como una disciplina aparte de la computación. Primeramente, el procesamiento de datos no era un tema de investigación interesante para los académicos ni desde el punto de vista de la enseñanza ni desde el de la investigación, pero según se fue aplicando el uso de las computadoras en las organizaciones, el interés aumentó:

> Punched card data processing was not an interesting academic subject for teaching or research. Early use of computers focused on simple transaction processing, so it didn't look interesting. What was interesting was the possibility of improved analysis, improved managerial reporting, and improved decision making. As organizations developed and implemented computer-based data processing systems, they experienced many interesting methods problems such as requirements determination, development methodologies, implementation, design of work systems, and evaluation.
>
> (*Ibidem*, p. 15)

En segundo lugar, los primeros investigadores académicos que tenían interés en los sistemas de información tenían un trasfondo de experiencias diverso (gestión, contabilidad, ciencia de cómputos y ciencias de gestión), con lo cual había conflicto de lealtades hacia las organizaciones académicas y profesionales ya establecidas y a las que ya pertenecían. Tampoco existía la necesidad de establecer una nueva disciplina académica, puesto que los interesados en investigar el tema se doctoraban en las disciplinas antes mencionadas[62]. Finalmente, dado que estos primeros investigadores estaban afiliados a organizaciones y facultades particulares a su campo, tenían dónde publicar y presentar el resultado de sus investigaciones. Varias organizaciones formaron grupos de interés relacionados con el campo de la informática y auspiciaron enfoques relacionados con el tema dentro de sus propios congresos, pero como existían estos medios no

[62] Davis indica que el primer programa doctoral en sistemas de información de América del Norte se estableció en la Universidad de Minnesota en 1968 (2006, p. 15).

había urgencia de establecer la informática como campo de investigación aparte con sus propios congresos y revistas (*Ibidem*, pp. 15-16).

1.7 El concepto de la metodología en el área de la informática

Antes de entrar en la discusión de la evolución histórica de las metodologías de desarrollo de sistemas informáticos debemos detenernos en cómo se define el término *metodología* en el ámbito del desarrollo de sistemas de información. Avison y Fitzgerald (2006a, p. 567) citan la definición de *metodología* que da el *Information Systems Analysis and Design Working Group* de la *British Computer Society* (BCS): "a recommended collection of philosophies, phases, procedures, rules, techniques, tools, documentation, management, and training for developers of information systems". En un artículo publicado en las actas del tercer congreso de la *BCS Specialist Group on IS Methodologies* Susan Gasson recoge algunas definiciones que otros autores proponen para luego resumir su postura así:

> A methodology is a *holistic* approach: it embodies an analytical framework which is conveyed through intersubjective representational practices and operationalised through a 'toolbox' of analytical methods, tools and techniques. Underlying the analytical framework is a process-model which indicates the sequence and relative duration of development activities. Standardised organisational procedures […] are critical to a holistic definition of a methodology, so that development outputs may conform to pre-defined standards, may be governed by pre-defined expectations and so that there may be a unified system of control over all IS development projects which use the methodology. Underlying all of these elements is a philosophical basis, which justifies the need for each element of the methodology and ensures intersubjectivity and commitment between development team members.
>
> (Gasson 1995, p. 2, énfasis en el original)

Avison y Fitzgerald expanden la primera definición añadiendo varios principios que ellos consideran fundamentales.

> A systems development methodology is a recommended means to achieve the development, or part of the development, of information systems based on a set of rationales and an underlying philosophy that supports, justifies and

> makes coherent such a recommendation for a particular context. The recommended means usually includes the identification of phases, procedures, tasks, rules, techniques, guidelines, documentation and tools. They might also include recommendations concerning the management and organization of the approach and the identification and training of the participants.
>
> (Avison y Fitzgerald 2006a, p. 568)

Estas definiciones comprenden a las partes implicadas (*stakeholders*[63]) que se involucran en el proceso de desarrollo de *software*, al igual que identifican una serie de fases y procedimientos durante las cuales se ejecutan tareas, se aplican técnicas y se siguen guías para producir un sistema de información en particular, su documentación y, de ser necesarios, hacer ajustes a la estructura organizacional de la empresa. Pensamos que, para profundizar en el campo, el localizador, como una de las partes implicadas en este proceso, debe conocer cómo se desarrollaron estas estrategias desde el punto de vista histórico y cuál es el estado de estas en la actualidad. Esto está en consonancia con el objetivo A y la pregunta de investigación A.

A continuación, veremos que las estrategias de análisis y diseño de sistemas informáticos se fueron desarrollando gradualmente hasta formar un abanico de metodologías. Esta división en "eras" está tomada de Avison y Fitzgerald (2006a, 2006b, 2003).

1.8 La era premetodología

Como vimos anteriormente, los programadores[64] de esta época se concentraban en resolver problemas técnicos y en la programación en sí. Avison y Fitzgerald llaman este período

[63] "Un *stakeholder* es una parte que tiene algún interés en el éxito del sistema: puede ser el cliente, los usuarios finales, los desarrolladores, el gestor de proyecto, quienes le dan mantenimiento al sistema y hasta quienes mercadean el sistema […]" (Bass, Clements y Kazman 2013, p. 52). En castellano se puede traducir de varias maneras (accionistas, involucrados, interesados, partes implicadas, partes interesadas, participantes), dependiendo del contexto.

[64] El concepto de programador no existía en los Estados Unidos y se les llamaba *coders* o codificadores. Todavía el término no había llegado de Inglaterra (Hopper 1981, p. 7).

la *era premetodología*[65]. Los programadores no estaban relacionados con las necesidades e interacciones características de los negocios y desarrollaban programas de forma individual, por lo que rara vez servían a la necesidad de los clientes.

> In the early days of computing, programmers did not understand the need for formal and well-planned life cycle methodologies. They tended to move directly from a very simple planning phase right into the construction step of the implementation phase—in other words, from a very fuzzy, not-well-thought-out system request into writing code.
>
> (Dennis, Wixom y Tegarden 2009, p. 8)

Esto limitaba grandemente las posibilidades de este nuevo campo y, según la tecnología cambiaba, se dieron cuenta de que hacía falta un proceso más estructurado y estandarizado para desarrollar los sistemas informáticos; de ahí surgen las primeras metodologías de desarrollo de sistemas basadas en el *ciclo de vida de desarrollo de sistemas* o SDLC[66]. El desarrollo de esta primera metodología marca el comienzo de la *era de metodología antigua*.

1.9 La era de metodología antigua[67]

"[… SDLC] is the process of understanding how an information system […] can support business needs, designing the system, building it, and delivering it to users" (*Ibidem*, p. 1). El analista de sistemas es la persona clave en este proceso y su función principal no es crear un sistema espectacular ni una herramienta para el negocio. Su función es proveer valor[68],

[65] "Pre-Methodology Era", que comprende desde los inicios de las computadoras hasta bien entrados en la década de 1960 (Avison y Fitzgerald 2003, p. 79, 2006a, pp. 576-577, 2006b, p. 28).

[66] *Software Development Life Cycle*. También *Systems Development Life Cycle*.

[67] "Early Methodology Era", que comprende la década de 1970 y principios de los 1980 (Avison y Fitzgerald 2003, pp. 79-80, 2006a, pp. 577-578, 2006b, pp. 28-29).

[68] Para la mayor parte de las empresas valor se traduce en aumentar los beneficios económicos (Dennis, Wixom y Tegarden 2009, p. 2). "Information systems often have significant consequences for the content and the shape of work in organizations. These consequences are not only visible in terms of money but also in changing

sobre todo, valor sobre la inversión en los sistemas informáticos. Este *ciclo de vida* está compuesto por fases, cada fase dividida en pasos que dependen de ciertas técnicas para producir unos resultados esperados específicos (*Ibidem*, p. 3).

Durante la fase de planificación el primer paso es hacer un estudio de viabilidad que debe contestar las siguientes preguntas: ¿se puede crear?; ¿añadirá valor al negocio?; si lo creamos, ¿lo usarán? Una vez se aprueba el proyecto, se comienza con la etapa de gestión de proyecto. El plan de gestión de proyecto indica cómo y en cuánto tiempo se llevará a cabo el proyecto.

La segunda fase es la fase de análisis y en esta se determina quién usará el sistema, qué hará dicho sistema, y dónde y cuándo se usará. La *estrategia de análisis*, que da dirección al equipo del proyecto, es el primer paso de esta fase, da una visión del estado actual del sistema y estudia posibles opciones sobre cómo puede ser el nuevo sistema. El paso siguiente recoge los requisitos del sistema, cuyo análisis generará el concepto del sistema que se usa para desarrollar modelos de análisis del negocio. Estos sirven como un mapa para describir la manera en que el negocio operará en función del nuevo sistema, describiendo los datos y procesos necesarios para brindar apoyo a las operaciones del negocio. El último paso de esta fase es la creación de una propuesta que debe ser aprobada antes de continuar con el desarrollo del proyecto (*Ibidem*, pp. 4-5).

Cuando se entra a la fase de diseño del sistema, se traduce lo que se ha propuesto en la fase anterior en términos físicos. El primer paso, conocido como la *estrategia de diseño*, establece cómo se va a desarrollar la propuesta: si se va a desarrollar usando programadores internos, si se va a contratar a un proveedor de servicios o si se va a usar un paquete de programas existente.

---

conditions of work, new authorities, and, so on. […] Financial consequences are the consequences which can be expressed in monetary terms. Non-financial consequences cannot be expressed in monetary terms […]. Financial and non-financial consequences together determine the **value** of an information system. **Benefits** refer to all positive consequences of an IS investment and **sacrifices** to all negative consequences" (Renkema y Berghout 1997, p. 2, énfasis en el original).

El diseño de arquitectura, el segundo paso, describe la infraestructura del sistema en cuanto a *hardware* o equipo, *software* y arquitectura de red[69]. El diseño de la interfaz especifica la navegación de los usuarios a través del sistema. Luego se especifica el diseño de las bases de datos que almacenarán la información, de qué manera se almacenará y dónde. El cuarto y último paso de esta fase es el diseño de programas, donde se describe qué programas hace falta crear y qué hacen estos programas exactamente. Al producto de esta fase, que incluye la información recogida en estos cuatro pasos, se lo conoce como *especificación del sistema*.

Durante la cuarta y última fase, la fase de implementación del sistema, se crea el nuevo sistema y se somete a pruebas de control de calidad para verificar que su desempeño sea el esperado. Este primer paso es el más importante y crítico, sobre todo lo referido a las pruebas de control de calidad, por lo cual en muchas ocasiones se dedica más tiempo a estas pruebas que a la programación. Luego de que está aprobado, el segundo paso de esta fase es la instalación del sistema y el adiestramiento de los usuarios. Esto se puede hacer de tres maneras: con una fecha de corte, donde se cambia de sistema en una fecha en específico; usando los sistemas de forma paralela (el nuevo y el viejo sistema, ambos a la vez); o haciendo un cambio de sistema por fases. El último paso de la fase de implementación es crear un plan de apoyo que incluye una revisión del sistema luego de terminar la implementación y una forma sistemática para identificar cambios de mayor y menor envergadura que haga falta hacer en el sistema[70] (*Ibidem*, pp. 5-6).

[69] Esto no se debe confundir con la *arquitectura de software*, de la cual hablaremos en el apartado dedicado a la investigación de Venkatesan sobre este tema en la página 98 del capítulo 2.

[70] Algunos distinguen este plan de apoyo como una fase adicional llamada *fase de mantenimiento*.

Como hemos visto, la metodología SDLC está delimitada por fases específicas, cada una de las cuales se debe completar antes de pasar a la fase siguiente. De esta estructura rígida que la compone surge el otro nombre con que se conoce: metodología de cascada (véase Figura 1.2). Ha sido utilizada ampliamente durante la década de 1970 y a principios de los 1980, y sigue siendo el fundamento de muchas de las metodologías que se usan en la actualidad. La gran cantidad de documentación que genera asegura una buena comunicación entre etapa y etapa, ayuda a controlar el tiempo y costo del proyecto, y establece controles al dividir el proyecto en etapas. A pesar de esto, Avison y Fitzgerald indican que esta metodología tiene "graves limitaciones", que se agudizan por la manera en que se pone en práctica, y nos explican las posibles críticas al usarla y aplicarla.

En primer lugar, la metodología "falla en atender las necesidades de los gestores de la empresa" y "en atender las necesidades reales de la empresa". Al aplicar esta metodología muchos desarrolladores se enfocan en crear aplicaciones informáticas únicas a nivel de las operaciones que se llevan a cabo en la empresa en lugar de mirar el desarrollo del sistema informático como un proceso integrador que atienda las necesidades del gestor y las de la

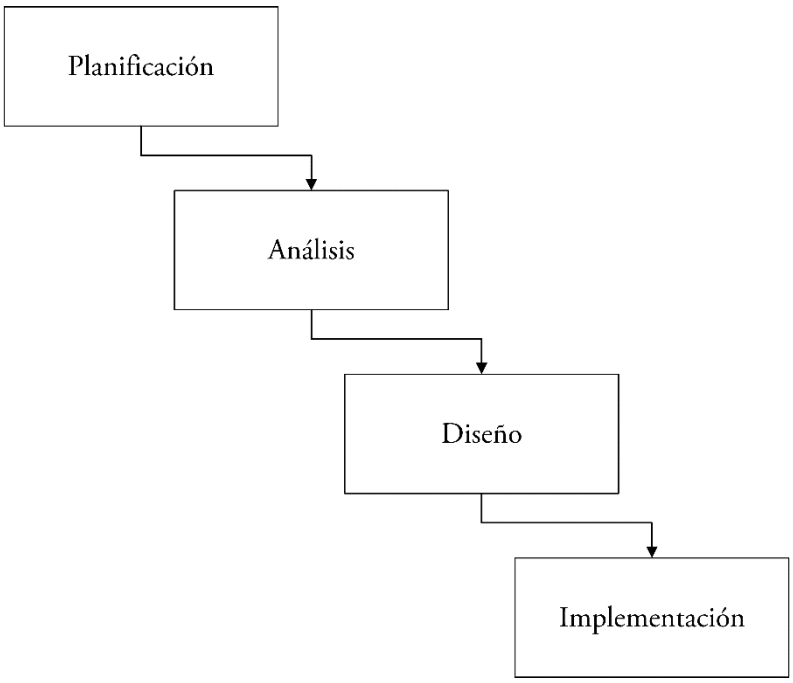

Figura 1.2 - El modelo de cascada

empresa que dirige[71]. También los desarrolladores tienden a "diseñ[ar] de manera poco ambiciosa" y a limitar su nivel de creatividad durante el proceso de desarrollo porque toman como base un proceso manual existente y lo mecanizan a nivel informático[72]. Por otro lado, esta metodología es rígida e inflexible; exige que cada etapa se complete antes de pasar a la próxima y, una vez una etapa se ha completado, no hay vuelta atrás. Por consecuencia, la rapidez con que cambian el mercado y las necesidades del negocio causa inestabilidad en la modelación de los procesos del negocio y hace que, una vez completado el proyecto, exista la posibilidad de que ya no responda a la finalidad para la cual fue desarrollado. Esta inflexibilidad y rigidez inherente a la metodología hace difícil y costoso incorporar cambios. Otra crítica es la insatisfacción de los usuarios, que no pueden "ver" el sistema antes de que esté funcionando y que tampoco tienen acceso a la documentación del sistema[73]. Finalmente explican que la estructura rígida de la metodología hace que las peticiones de cambios[74] se acumulen, con lo

[71] La SDLC hace hincapié en crear aplicaciones (para el sentido de "aplicaciones" véase nota 25) que automaticen a nivel informático el área operativa de la empresa. Si miramos una empresa como estructura piramidal, encontramos el área operativa en los niveles más bajos de la pirámide, con los gestores y oficiales en los niveles más altos. La SDLC busca implementar un cambio a nivel tecnológico de un problema único bien definido. Susan Gasson, en la introducción de su artículo sobre el uso práctico de las metodologías en el desarrollo de los sistemas de información, explica indirectamente la limitación inicial que existía en cuanto a la visión de la función de las metodologías de análisis y diseños de sistemas: "[i]n a narrow sense, information systems development may be seen as technical change. A single problem is seen as well-defined, a technical solution is proposed, evaluated and implemented" (Gasson 1995, p. 1).

[72] Los autores lo llaman "computeriz[e]", es decir, convertir un proceso manual a un proceso que usa una computadora.

[73] No solo que no tienen acceso a ella, sino que, además, el enfoque de la documentación es hacia la aplicación y no hacia el usuario, y en raras ocasiones se mantiene actualizada. Como contraste a esta metodología están las metodologías de desarrollo rápido y evolutivo, por ejemplo, el RAD (véase nota 111), que incluyen modelos prototípicos con los que el usuario puede interactuar.

[74] Estas peticiones surgen cuando hace falta modificar las aplicaciones según cambian las necesidades del usuario, del mercado y de la empresa. Como la etapa de atender estas peticiones corresponde a la fase de análisis y esta fase ya ha pasado, cualquier cambio debe hacerse luego de que haya concluido la fase de implementación, es decir, una vez haya terminado el proyecto en su totalidad.

cual no queda más remedio que dejarlas para que sean atendidas más adelante[75]. Así, cuando se entrega el producto, este no está atemperado a las necesidades de la actualidad y habrá que esperar a que estos cambios se incorporen en el futuro (Avison y Fitzgerald 2006b, p. 29, 2003, pp. 79-80).

Otra importante desventaja que presenta esta metodología la señala Winograd (1995, p. 118) cuando cita a Goguen, que dice que la mayor falla (y arrogancia) del modelo de cascada es que no da espacio a considerar el error y ni a incluir ningún tipo de retroalimentación a lo largo de la aplicación de la metodología. Los errores que se hayan cometido en las etapas de análisis y diseño se hacen sobre todo evidentes durante la fase de implementación, pero la estructura rígida que conforma esta metodología no permite dar vuelta atrás para corregirlos. Esto contribuye a que aumente la probabilidad de que el proyecto fracase y limita las posibilidades del proyecto a lo ya analizado y diseñado.[76]

Aprovechamos este momento para adelantar un ejemplo de los problemas que afronta la localización de software que se hace evidente durante la etapa de implementación y que, por causa de la rigidez de este modelo, imposibilitaría llevar a cabo el proceso de localización. Cuando se comienza a traducir, aproximadamente a mitad de camino en esta etapa (cuando hay suficientes cadenas de texto creadas y comienzan a traducirlas), los localizadores empiezan a toparse con problemas de apoyo a caracteres relacionados con idiomas que usan alfabetos no

[75] Durante la *fase de mantenimiento*. Véase nota 70.

[76] Winograd cita a Goguen en el contexto de la conferencia "Software Development and Reality Construction: An Invited Working Conference Focusing on Epistemological Foundations for Development and Use of Computer Based Systems" celebrada en 1988 en Alemania. Luego, en el artículo titulado "The Denial of Error" (Goguen 1991), Goguen critica la obsesión por el control que manifiesta la cultura occidental y su arrogancia al proponer metodologías que garantizan la ausencia de errores. Ambos citan la postura de Heidegger en cuanto a experimentar y reconocer el poder del error: "[…a]ppearance, deception, illusion, error stand in definite essential and dynamic relations which have long been misinterpreted by psychology and epistemology and which consequently, in our daily lives, we have wellneigh ceased to experience and recognize as powers" (*Ibidem*, p. 194; Winograd 1995, pp. 118-119).

latinos, y formatos de fechas y números. Un poco después, al hacer pruebas de control de calidad, comienzan a encontrar problemas para desplegar correctamente o ingresar ciertos caracteres que no están disponibles en teclados latinos. En ese momento también se dan cuenta de que la base de datos donde se almacena la información no admite caracteres fuera del repertorio ASCII[77]. Este tipo de problema, que se debió haber resuelto en etapas anteriores, no permitiría continuar con la traducción del *software* a esos idiomas que necesitan desplegar estos caracteres no latinos o usar teclados que den apoyo a estos caracteres. En un modelo de cascada puro, se tendría que esperar a terminar la implementación completa para empezar a decidir qué hacer con estas limitaciones que no se consideraron en etapas anteriores. La respuesta, muy posiblemente será no hacer nada, debido al alto costo de alterar los programas. Incluir a un localizador a lo largo de todo el proceso de desarrollo de programas ayudaría a reducir la posibilidad de que esto sucediera (todos los objetivos de nuestra investigación, preguntas de investigación A, página 12 y C, página 12). El localizador, en su función de usuario y analista experto en mediación intercultural, señalaría durante la etapa de análisis y la de diseño que este sería un problema que atender con urgencia para posibilitar la localización más adelante.

1.10 La era de las metodologías

Las limitaciones y críticas de la SDLC abren las puertas a la *era de las metodologías*[78]. Avison y Fitzgerald explican en el artículo *Where Now?* que probablemente es durante este período que se comienza a hablar sobre una metodología y la definen como "un conjunto de fases, procedimientos, reglas, técnicas, herramientas, documentación, gestión y adiestramiento que se usa para desarrollar un sistema"[79]. Estas nuevas metodologías que surgen durante esta

[77] Véase el apartado "Codificación de caracteres, Unicode" en la página 170.

[78] "Methodology Era", que comprende las décadas de 1980 y 1990 (Avison y Fitzgerald 2003, p. 80, 2006a, pp. 578-583, 2006b, pp. 29-32).

[79] "A methodology is a recommended collection of phases, procedures, rules, techniques, tools, documentation, management, and training used to develop a system […]" (Avison y Fitzgerald 2003, p. 80) Para una definición más abarcadora véase el apartado "El concepto de la metodología en el área de la informática" en la página 43.

época tienen la particularidad de que están basadas en una filosofía subyacente, no siempre explicitada por sus autores, que expresa los principios y presunciones que conforman la clave de su funcionalidad. La motivación para adoptar una u otra metodología puede variar, pero su finalidad se resume en: lograr obtener un mejor producto, lograr obtener un mejor proceso de desarrollo o llegar a un proceso estandarizado. La primera está relacionada directamente con satisfacer las necesidades del usuario, la segunda con mejorar la productividad y el control del desarrollador, y la tercera con beneficiar la integración de los sistemas y beneficiar a la empresa con un esfuerzo en común (Avison y Fitzgerald 2003, p. 80).

Esta época se puede dividir en dos movimientos principales, uno que busca mejorar y ampliar el modelo tradicional de cascada y otro que propone nuevas metodologías que se distancian en gran manera de este modelo. Los fundamentos en los que se basan algunas de estas nuevas metodologías propuestas incluso muestran características y principios que las hacen ser particularmente distintas entre sí.

Para mejorar el modelo de cascada se introducen nuevas técnicas y herramientas[80]: *entity-relationship modelling,* normalización, diagramas de flujo de datos, inglés estructurado, diagramas de acción, diagramas de estructura y ciclo de vida de entidades, entre otras. Entre las herramientas podemos destacar programas para la gestión de proyectos, programas para la gestión de los diccionarios de datos, repositorios de sistemas, herramientas para crear diagramas y sistemas de ingeniería de *software* asistidos por computadora (herramientas CASE, *computer-assisted software engineering*) (Avison y Fitzgerald 2006b, p. 30).

[80] Véase el apartado “Herramientas visuales para la modelación de sistemas” en la página 155.

El distanciamiento del modelo de cascada se puede percibir a partir de la década de 1980. Avison y Fitzgerald (*Idem*) clasifican las nuevas metodologías de esta época en seis enfoques: de sistemas, estratégicos, participativos, de prototipos, estructurados y de datos (véase Tabla 1.1). El enfoque de sistemas se basa en la teoría general de sistemas y trata a la empresa como un sistema abierto que se ve afectado por el medioambiente que la rodea. El enfoque estratégico basa el desarrollo de sistemas informáticos en la estrategia que los gestores de alto nivel establecen para cumplir con las metas y objetivos de la empresa. El extremo opuesto se manifiesta en el enfoque participativo, que enfatiza la aportación de todos los usuarios de los sistemas informáticos e incluso hasta relega a niveles inferiores la participación de las partes implicadas del área de la informática. Cuando se involucra de esta manera a los usuarios el resultado esperado es que se comprometan por completo con el proyecto para así garantizar

Enfoque	Metodología
sistemas	*Soft Systems Methodology* (SSM)
estratégico	*Business Systems Planning* de IBM
participativo	ETHICS
prototipos	*Rapid Application Development* (RAD)
estructurado	Aplicación de técnicas como árbol de decisiones, diagrama de flujo de datos (DFD), diagrama de estructura de datos, inglés estructurado, repositorios de sistemas
análisis de datos	*Information Engineering*

Tabla 1.1 - Enfoques y metodologías de la era de las metodologías según Avison y Fitzgerald (Idem)

que este sea aceptado en toda la empresa.[81] El enfoque de prototipos busca resolver, principalmente, el problema de "ver" el producto antes de terminado. En lugar de diagramas abstractos y técnicos se les presenta a los usuarios una implementación prototípica con las prestaciones principales del sistema, para que puedan ver el insumo (*input*), los procesos y el producto de la aplicación que se está desarrollando. Los enfoques estructurados están centrados en los procesos y buscan descomponer problemas complejos en unidades más pequeñas usando una técnica de descomposición funcional, mientras que los de análisis de datos enfatizan el entendimiento, la documentación y la validación de los datos relevantes a todas las partes implicadas de la empresa (*Ibidem*, pp. 30-31)[82].

En la década de 1990 comienza una segunda ola de metodologías, las llamadas *metodologías orientadas a objetos*[83]. Estas metodologías, argumenta Yourdon (*Ibidem*, p. 31),

[81] En su libro *Work-Oriented Design of Computer Artifacts* (1988), Ehn presenta su experiencia como diseñador de artefactos de computadora (o computarizados) y propone un diseño fundado en interpretaciones pragmáticas de la filosofía de la fenomenología existencial, la cual contrasta con los modelos cartesianos que han regido el pensamiento de los desarrolladores del ámbito informático. Su propuesta busca la democratización del trabajo y la creación de herramientas que, en lugar de fomentar la desespecializacón o *de-skilling* (algo que considera deshumanizador), propulsen el enriquecimiento de los trabajadores a fin de que generen productos y servicios de gran utilidad y calidad. Esta alternativa enfatiza los "social systems design methods, a new theoretical foundation of design, and the new potential for design in the use of prototyping software and hardware" (*Ibidem*, v. Abstract), que conforman el fundamento de los sistemas participativos. Es un punto de vista más humanista y que toma en consideración los aspectos sociales como parte fundamental de la mecanización de las empresas:

> […W]ork-oriented design of computer artifacts is, as I see it, not only a strategy to include users and their trade union activities in the design process, but more fundamentally to include a cultural and anthropological understanding of human design and use of artifacts, to rethink the dominating objectivistic and rationalistic conception of design […](*Ibidem*, p. 5)

[82] Más adelante presentaremos algunas de las herramientas visuales usadas para documentar y diseñar sistemas basados en estos enfoques. Véase los apartados "Modelación de procesos" y "Modelación de datos" en las páginas 156 y 159 respectivamente. Ahí mismo, en el apartado "Modelación de objetos" (página 161), también discutimos herramientas visuales para el enfoque orientado a objetos que veremos enseguida.

[83] Las metodologías orientadas a objetos usan varios elementos encapsulados en módulos para construir un sistema informático: clases, objetos, métodos y mensajes. "Object-oriented systems focus on capturing the structure and behavior of information systems in little modules that encompass both data and process. These little

enfatizan la reutilización de código por medio de la modularización y son más naturales que los acercamientos enfocados en datos o en procesos. La "naturalidad" que se les atribuye a estas metodologías tiene que ver con la manera de conceptualizar la interacción entre las partes implicadas que participan en el proceso de desarrollo de *software*:

> […] The primary difference between a traditional approach like structured design and an object-oriented approach is how a problem is decomposed. In traditional approaches, the problem decomposition process is either process centric or data centric. However, processes and data are so closely related that it is difficult to pick one or the other as the primary focus. […] "[O]bject-think" is a much more realistic way to think about the real world. Users typically do not think in terms of data or process; instead, they see their business as a collection of logical units that contain both—so communicating in terms of objects improves the interaction between a user and an analyst or developer.
>
> (Dennis, Wixom y Tegarden 2009, pp. 17, 19)

Esta naturalidad de la programación orientada a objetos se manifiesta en las metáforas de agrupación, clasificación, ordenamiento y herencia que usa. Coad y Yourdon (en Avison y Fitzgerald 2006b, p. 31) mencionan las ventajas y motivaciones para utilizar estas metodologías: que son mejores para abordar situaciones más complejas dado el entendimiento más profundo que brindan sobre el problema analizado; que la relación entre el analista y el usuario es mejor, pues no están orientadas directamente hacia la computarización[84] de los sistemas; que genera resultados más uniformes y consecuentes, pues el modelaje se genera tomando en consideración y de igual manera todos los aspectos del problema; y que permite

modules are known as objects" (Dennis, Wixom y Tegarden 2009, v. Apéndice, p. 1). "The basic building block of the system is the *object*. Objects are *instances* of *classes* that we use as templates to define objects. A class defines both the data and processes that each object contains. Each object has *attributes* that describe data about the object. Objects have *state*, which is defined by the value of its attributes and its relationships with other objects at a particular point in time. And, each object has *methods*, which specify what processes the object can perform. From our perspective, methods are used to implement the *operations* that specified the *behavior* of the objects […]. In order to get an object to perform a method (e.g., to delete itself), a *message* is sent to the object. A message is essentially a function or procedure call from one object to another object" (*Ibidem*, p. 320).

[84] *Computerize*. Véase nota 72.

representar factores cambiantes dentro del modelo, lo que lo hace un modelo más elástico y flexible.

También durante la década de 1990 surgen las metodologías incrementales o de desarrollo evolutivo, mediante las cuales no se genera un sistema completo de una sola vez, sino que se dividen los sistemas en módulos prototípicos más pequeños y sobre los cuales se van iterando o desarrollando de manera incremental dichos módulos. Este modelo permite crear un sistema básico al cual se le van añadiendo prestaciones nuevas, y se modifican y mejoran las que ya se incluyeron en las versiones anteriores, lo que da paso a la incorporación de cambios y mejoras en cada versión. De esta forma se reduce el tiempo de desarrollo de un sistema y se toma en consideración el problema de la necesidad de cambios en los requerimientos que surgen como resultado del aprendizaje inherente al proceso de desarrollo y que ocurren a lo largo del tiempo que toma desarrollar sistemas usando un modelo tradicional (por ejemplo, el modelo de cascada) (*Idem*).

1.11 La era posmetodología[85]

Avison y Fitzgerald caracterizan el presente como una "era de reflexión" en la que las empresas tienden a alejarse de las metodologías puras y reflexionan sobre las ventajas y desventajas que estas pueden presentar (*Ibidem*, p. 33). Incluso hay empresas que rechazan del todo el uso de metodologías y prefieren adoptar estilos menos formales y más flexibles; otras continúan en la búsqueda de metodologías alternas que solucionen la falta de adecuación de la metodología que usan, pero no porque estén en desacuerdo con el uso de metodologías (Avison y Fitzgerald 2006a, p. 586). Ya para 1998 un estudio empírico que analiza hasta qué punto se aplican en la práctica las metodologías de desarrollo de sistemas, cuánto contribuyen al proceso de desarrollo y las tendencias futuras sobre adoptar metodologías realizado por Fitzgerald indicaba que un 60 % de los participantes en el estudio no usaban metodología alguna para

[85] "Post-Methodology Era" (Avison y Fitzgerald 2003, pp. 80-82, 2006b, pp. 33-37). También "Era of methodology reassessment", que los autores identifican como la era presente, es decir, a partir del cambio de siglo (Avison y Fitzgerald 2006a, pp. 583-589).

desarrollar aplicaciones y solo el 6 % las usaban y aplicaban con rigor. El 79 % de los que no usan metodología alguna indicó que no tenían intención alguna de usarla (Fitzgerald 1998, p. 317). En ese mismo estudio el autor enumera los argumentos y presiones que se consideran a favor y en contra de usar algún tipo de metodología (véase Tabla 1.2).

Avison y Fitzgerald presentan una detallada enumeración de las críticas que hacen los

Arguments in favour of methodologies	Arguments against of methodologies
Systems development is a very complex process. Methodologies may provide a reductionist subdivision […] into plausible and coherent steps. [… M]ethodologies may facilitate project management and control of the development process, thus reducing risk and uncertainty. [… Methodologies] may provide a purposeful framework for the application of techniques and resources during the development process. There is an economic rationale: methodologies may allow skill specialisation and division of labour which can receive different remuneration rates. An epistemological rationale can be identified as methodologies may provide a structural framework for the acquisition of knowledge. […] Standardisation of the development process […, which] facilitates interchangeability among developers […, …] is possible [… and can also] lead to increased productivity and quality, as resource requirements can be predicted and made available as and when necessary.	Estimates suggest that more than a thousand brand-named methodologies exist. […] Generalisation has been made without adequate conceptual and empirical foundation […]. Systems development is not actually an orderly rational process but most […] treat it as such, with a major emphasis on technical rationality at the expense of social aspects. There may be some means-ends inversion, as developers proceed in slavish and blind adherence to methodologies, while losing sight of the fact that development of an actual system is the real objective. Often there is an assumption that methodologies are universally applicable across all development situations—the one size fits all presumption. […] There is an inadequate recognition of developer-embodied factors, as […these methodologies] do not address factors critical to successful development, such as individual creativity and intuition, or learning over time.

Tabla 1.2 - Argumentos a favor y en contra del uso de las metodologías según Fitzgerald (1998, pp. 317-319)

detractores de las metodologías: que no logran los beneficios de productividad esperados; que son excesivamente complejas; que para usarlas hace falta personas con destrezas especializadas

en su aplicación y uso, y que estas destrezas no son fáciles de adquirir; que las herramientas que usan estas metodologías son difíciles de usar; que son inflexibles; que parten de premisas inválidas o poco prácticas; que no enfatizan suficientemente la dimensión social ni los problemas de contexto; y que la metodología, por la razón que sea, no ha logrado producir mejores sistemas informáticos, entre varias otras críticas (Avison y Fitzgerald 2006a, pp. 583-586). Indican también que durante esta era de reflexión el rumbo que las empresas han tomado en cuanto al uso de metodologías ha variado. Por un lado, se ha observado el desarrollo *ad hoc* de sistemas informáticos, algo muy parecido al acercamiento durante la era premetodología. Por otro, los acercamientos estilo *buffet*, donde la empresa selecciona una metodología, pero usa o adapta las técnicas y herramientas a conveniencia. Del otro extremo hay empresas que recurren al desarrollo externo, es decir, a comprar paquetes de *software* a terceros o a pedir a terceros que desarrollen el *software* (*outsourcing*). Incluso, mencionan que hay otras empresas que todavía andan en busca del santo grial de las metodologías (*Ibidem*, pp. 586-588).

1.12 Un historial relativamente breve, pero de profundo impacto

En este capítulo presentamos la evolución de las computadoras como un elemento que lleva de la mano a los lenguajes de programación. Una vez llegado a un punto donde se comercializa el uso de las computadoras, surge la necesidad de estructurar los procesos de desarrollo de los programas. La evolución de las metodologías de desarrollo de *software* comienza por las no-metodologías, pasa por una etapa de múltiples metodologías y evoluciona hasta el estado actual: una zona híbrida entre las metodologías establecidas y las no-metodologías. El desarrollo de las plataformas donde se ejecutan los programas procede de un pasado de claras delimitaciones a un presente amalgamado, en el que las posibilidades oscilan entre los antiguos extremos, *mainframe* ↔ microcomputador, pero las fronteras entre sus elementos son cada día más borrosas. La internet difumina cada vez más los estratos de esta amalgama, puesto que la plataforma móvil hace uso de todos estos equipos, aunque esto sea invisible para el usuario final.

Para sobrevivir ante este presente, pensamos que es imperativo que el localizador conozca estas plataformas, las metodologías de desarrollo, los lenguajes de programación y los medios que están disponibles en la actualidad a fin de posibilitar su participación durante el proceso de desarrollo de *software*, no solo como mediador intercultural, sino también como mediador interplataforma e interlenguaje. La (no-)aplicación de las metodologías en el desarrollo de *software* pinta un panorama borroso, empero es importante que el localizador pueda abordar este ámbito entendiendo qué se toma en consideración y qué se evalúa durante cada etapa del desarrollo para que pueda aplicar sus destrezas no solo como traductor y gestor cultural, sino también como conocedor del lenguaje y del metalenguaje con el que se comunica el programador a lo largo de estos procesos. Conocer las posibilidades y prestaciones para el desarrollo de programas multilingües existentes le ayudarán a lograr que el producto quede adecuadamente internacionalizado, a fin de que pueda ser localizado durante el proceso de desarrollo mismo o en algún momento futuro. Como hemos ido resaltando a lo largo de este capítulo, la información que presentamos es consonante con el objetivo A en la página 11 y la pregunta de investigación A en la página 12.

Capítulo 2: El desarrollo de *software* y su relación con la localización: definiciones e investigaciones previas

2.1 Globalización, internacionalización, localización y traducción

El surgimiento de la localización está muy relacionado con la llegada de las computadoras personales a principio de los años 80. En 1981 IBM anunció la computadora más pequeña y más barata que jamás haya producido, la IBM Personal Computer. Su introducción dio inicio al mercado de más rápido crecimiento en la historia de la informática: el de las computadoras personales (International Business Machines 2014b, p. 14; Computer History Museum 2006). Aunque Steve Wozniak y Steve Jobs habían formado Apple Computer, Inc. y lanzado la Apple I en 1976, no fue sino hasta 1984 que apareció la primera computadora personal con interfaz gráfica (WYSIWYG[86]) y ratón que tuvo éxito. La interfaz gráfica de la nueva Apple Macintosh estaba basada en WIMP[87], una colección de conceptos que hablaba un lenguaje absolutamente nuevo e innovador en el ámbito de las computadoras personales[88]. Microsoft vino a presentar su primera interfaz gráfica, el Microsoft Windows

[86] Siglas que corresponden a "What you see is what you get". La IBM PC usaba como sistema operativo MS DOS, desarrollado por Microsoft, un sistema operativo basado en comandos que había que memorizar y un complicado sistema para organizar ficheros.

[87] Siglas para "windows, icons, menus, and pointing device".

[88] La interfaz tipo WIMP era ya conocida en ámbito investigativo y académico. En 1970 la empresa Xerox estableció el famoso centro de investigaciones de Palo Alto, California (*Palo Alto Research Center*, PARC) donde en 1976 se estaba experimentando con la computadora Alto, la primera en usar este tipo de interfaz. Esta computadora era de vanguardia no solo en este aspecto: incluía una pantalla gráfica en blanco y negro de 606 x 808 pixeles, teclado, ratón, un procesador razonablemente poderoso, disco duro e interfaz *ethernet* que la conectaba a un servidor de ficheros, una impresora láser de alto rendimiento, a otras computadoras de la empresa y a ARPANET, la precursora de la Internet. Las prestaciones de este equipo las consideramos comunes y quizás algo *passé* en la actualidad pero, en ese momento, eran absolutamente revolucionarias. Paul McJones explica: "The Alto software included Bravo, the first WYSIWYG (What You See Is What You Get) word processor, Markup, a pixel-oriented drawing editor, and Draw, a vector-oriented drawing editor. That fall day I had my first taste of the world that we now take for granted: the ability to create and modify digital documents containing text and graphics, store them, and transmit them 'at blinding electronic speed' […]" Se produjeron unas 1500 unidades a un costo aproximado de $12 000 USD que se instalaron en varias oficinas de la empresa, universidades y otros lugares, y nunca se comercializaron (Computer History Museum 2011e; McJones 2014).

Operating Environment, Versión 1.0 dos años más tarde, sin poder emular la elegancia visual y eficiencia en ejecutoria de la Macintosh y su nuevo sistema operativo (Computer History Museum 2011g, secs. 10, 12).

Es común asociar las interfaces gráficas con la localización, pero ello no significa que no se comenzara a adaptar el idioma de la interfaz de los programas que se usaban en las primeras computadoras personales (con interfaz de solo texto) que aparecen en el mercado. Aunque "[…t]he IT industry was primarily developed in the US and therefore centered on American English and the initial awareness of the linguistic requirements of the international market was low […]" (O'Hagan y Mangiron 2013, p. 87), las empresas poco a poco se fueron dando cuenta de la importancia de atender estos mercados usando un idioma que les fuera familiar. Es lógico pensar que las primeras aplicaciones que requieren dar apoyo al uso de otros idiomas sean los ahora conocidos procesadores de texto, pero es difícil encontrar documentación alguna que arrojara una fecha, aunque fuera aproximada. En el mercado de procesadores de palabras, Apple Computer llevaba la delantera pues ya desde 1979 ofrecía "Apple Writer" para su Apple II. En sus inicios, la oferta de programas de Microsoft se limitó principalmente a compiladores[89], a desarrollar BIOS[90] y sistemas operativos para una amplia gama de plataformas. No fue sino hasta 1980 que anunciaron su primera aplicación, "Typing Tutor". El primer procesador de palabras que ofreció Microsoft, inicialmente llamado "Multi-Tool Word" y luego renombrado "Microsoft Word" como parte de la estrategia de imagen de la empresa en ese momento, no vino a lanzarse sino hasta 1983. Prestaciones como el recuento de palabras y la corrección ortográfica para este programa no aparecieron sino hasta 1985 (Allan 2001, p. 7/7, 12/25-26; Microsoft 2009, años 1975-1980). Lo que sí es cierto es que, según el mercado de las computadoras se amplió y el público en general comenzó a formar parte de los usuarios de las aplicaciones, las prestaciones y funcionalidades cambiaron. Las computadoras no solo tenían que ayudar a facilitar el trabajo de las personas, sino que también

[89] Véase la explicación sobre los compiladores y los lenguajes compilados en la página 26 y subsiguientes.

[90] Siglas de "Basic Input/Output System". Es lo primero que se ejecuta cuando se inicia cualquier computadora y lo que se encarga de activar el sistema operativo de la máquina.

tenían que hacerlo usando el idioma y tomando en consideración los hábitos y estándares de las localidades particulares, pues el mercado se había expandido. "[…] The international expansion of software and hardware developers automatically triggered the need to localize the products for international markets […]" (Esselink 2006, p. 22). A partir de esta necesidad las empresas de computadoras y *software* comienzan a establecer departamentos internos de traducción o a comisionar a traductores autónomos la traducción de los programas y la documentación. Entonces surge lo que a principio de los años 1990 se convertirá en la industria de la localización (Esselink 2000, p. 5).

En la industria de la localización se hace referencia a las siglas GILT para describir las etapas del proceso de localización: globalización, internacionalización, localización y traducción. Este acrónimo "[…] is used to stress how the globalization of modern technological platforms needs to be considered from the beginning with localization in mind, which in turn will be determined by companies' overall globalization strategies" (O'Hagan y Mangiron 2013, p. 89). Dunne sugiere que sería más lógico llamarlo TLIG puesto que refleja mejor la evolución histórica de la industria y la manera en que los estrategas de las empresas han tomado conciencia de la importancia de incorporar estos procesos no solo en el desarrollo de productos informáticos, sino también a lo largo de toda la estructura empresarial (Dunne 2006, p. 4).

2.1.1 Traducción y localización

Aunque el término *localización*[91] está relacionado principalmente con la traducción, es lo suficientemente abarcador como para incluir ámbitos que trascienden la traducción. "[… L]ocalization revolves around combining language and technology to produce a product that can cross cultural and language barriers. No more, no less […]" dice Esselink (2006, p. 21), sin más. Para efectos prácticos, la traducción se entremezcla con los procesos de localización de

[91] A la localización también se le conoce por su numerónimo L10n, una contracción del término original compuesta por un diez flanqueado por la primera y última letra de la palabra. El número diez sustituye a las letras que van entre la ele inicial (en mayúsculas para que se distinga claramente que es una ele) y la ene final. Se usa porque funciona no solo en castellano, sino también en inglés y en francés. Más adelante abordaremos la internacionalización y la globalización, i18n y g11n (creados y usados de igual manera).

manera tan íntima que no merece la pena distinguir uno del otro. LISA, que hasta marzo de 2011 fue la asociación de estándares de la industria de la localización, explica que la "[l]ocalization is the process of modifying products or services to account for differences in distinct markets" (Lommel 2007, p. 11). En esta definición se incluyen los servicios además de los productos, lo cual amplía los lindes que por lo general se asocian con este tema. Más adelante explica que la localización aborda cuestiones lingüísticas, físicas, comerciales y culturales, y técnicas. Las cuestiones lingüísticas (y culturales, en sentido general) están directamente relacionadas con la traducción; las físicas tienen que ver con el diseño físico del producto, por ejemplo, la adaptabilidad de voltaje de un producto electrónico o la ubicación del volante de un automóvil que se vaya a vender en mercados como Japón o Inglaterra, donde se conduce por el lado izquierdo de la carretera. Las cuestiones culturales (en relación con su mecanismo técnico de expresión o funcionamiento) y de negocio están ligadas a consideraciones en el diseño del producto relacionadas con moneda, reglamentación en la contabilidad y legislación del país o región, formatos de las direcciones y los nombres, la selección de gráficos y colores, incluso cuestiones políticas y de expectativas culturales en el mercado meta, entre otras. Los aspectos técnicos tienen que ver con la ordenación de letras y palabras, la capacidad de usar sistemas de caracteres expandidos y de idiomas que van de derecha a izquierda (RTL, por sus siglas en inglés), el (re)diseño de la interfaz, y los medios de insumo de datos, por mencionar algunos (*Ibidem*, pp. 12-15).

Según The Localization Institute, "localizar es el proceso de crear o adaptar un producto a un *locale*[92] específico, es decir, al idioma, al contexto cultural, a las convenciones y

[92] En su colección de terminología de su portal lingüístico Microsoft traduce el término *locale* como configuración regional (Microsoft Developer Network 2014b) y define esta configuración regional como "una colección de información sobre las preferencias del usuario que están relacionadas con el idioma". Ahí mismo explica que, en este caso, el idioma es el indicador de "un conjunto de propiedades que se usan en la comunicación oral y escrita", que cada idioma tiene un nombre particular y que está relacionado a una *code page* o página de códigos específica que se usa para representar la ubicación geográfica del idioma (Microsoft Developer Network 2014c).

requisitos de un mercado meta específico" (2012, traducción nuestra). A pesar de que esta definición es menos abarcadora, pues se limita solo a productos, recoge en esencia los mismos elementos de la definición anterior. Si bien dijimos que la localización es adaptar un producto a un *locale* específico, en esta definición no queda claro qué sucede con los aspectos culturales y del mercado meta (en su expresión técnica o de uso). Esto se relaciona con otros dos aspectos que van de mano con la localización: la internacionalización y la globalización, que veremos más adelante.

2.1.2 El perfil del localizador y sus competencias en la actualidad

En su obra seminal en el campo de la localización, Esselink (2000) describe quiénes son las personas involucradas a lo largo del proceso de localización. Su modelo, a partir de las prácticas iniciales características de la década de 1990, persiste aun en la actualidad. El autor indica que el grupo de personas que por lo regular se involucran en un proyecto de localización incluyen administradores de cuenta, gestores de proyecto, especialistas en localización/traductores *senior*/traductores, revisores/especialistas en QA[93], ingenieros de localización/ingenieros de pruebas o QC, especialistas en herramientas TAO y operadores de programas de maquetación. Los ingenieros de localización/ingenieros de pruebas o QC, quienes son los responsables de los aspectos técnicos de los proyectos de traducción, tienen a

El uso de las páginas de códigos ha ido cayendo en desuso puesto que la mayor parte de las aplicaciones y programas se escriben usando Unicode (véase el apartado "Codificación de caracteres, Unicode" en la página 170), pero incluso las más nuevas aplicaciones pueden tener la necesidad de seguir usando las anticuadas páginas de código para poder comunicarse con aplicaciones heredadas (*legacy applications*, que son programas desarrollados sobre arquitecturas de *hardware* antiguas pero que, de una manera o de otra, continúan usándose en la actualidad sobre las nuevas arquitecturas), con sistemas de correo electrónico que carezcan de apoyo a Unicode y con otras aplicaciones que no den apoyo a este tipo de codificación de datos más moderno (Microsoft Developer Network 2014a).

[93] *Quality Assurance* - Control de calidad.

su cargo la preparación del proyecto, la ingeniería de *software* y de la ayuda en línea, compilar los programas y realizar pruebas (*Ibidem*, p. 16). Algunas de las destrezas que debe tener son:

- Conocimiento práctico de la instalación y la gestión de la configuración de los sistemas operativos plataformas más comunes.
- Experiencia en entornos de desarrollo y programación estándar en la industria.
- Entendimiento de los problemas relacionados con la internacionalización y con los conjuntos de caracteres.
- Conocimiento de sistemas de bases de datos, lenguajes de guiones[94] y desarrollo de interfaces gráficas para usuarios.
- Una perspectiva profesional y creativa en cuanto a la resolución de problemas y de *bugs*.
- Ser consciente de los modelos de localización y flujo de trabajo, y de los problemas inherentes a la coordinación de tareas en los proyectos.
- Experiencia en seleccionar y usar herramientas de TAO.
- Conocimiento de la tecnología relacionada con la Internet.
- Destrezas de comunicación oral y escrita excelentes.
- Se prefiere que tenga destrezas en lenguas extranjeras (*Ibidem*, p. 98).

Según lo describe el autor, este participante es el experto que funciona como vínculo principal entre la localización y el desarrollo de *software*, pues su repertorio de conocimientos y destrezas, y las funciones que realiza abarcan dos ámbitos del conocimiento: el humanístico y el informático. Aquí lo vemos más claro:

> […] The practice of localization emerged in order to accommodate the specialized process required to extract and integrate fragments of text (strings)

[94] *Scripting languages*, como por ejemplo JavaScript, Perl, Ruby, entre otros. Véase (Collado 2013).

> embedded in the software, in addition to the translation task of converting these strings to a given language […].
>
> (O'Hagan y Mangiron 2013, p. 89)

De un lado, el localizador necesita llevar a cabo un ejercicio *técnico* que consiste en extraer los fragmentos de texto que están dentro del *software* y luego colocarlos de vuelta. Del otro, el localizador tiene que llevar a cabo un ejercicio de *traducción* y traducir dichas cadenas de texto al idioma correspondiente, tomando en consideración las adaptaciones culturales necesarias.

El perfil más actual de esta misma figura en el desarrollo de *software* lo describen Murtaza y Shwan (2012). Los ingenieros de localización "are in essence software translators". Los autores toman prestado de Esselink las competencias que debe exhibir, pero las reducen a las cinco más relevantes en la actualidad: competente en cuanto al apoyo que dan los distintos sistemas operativos a los *locales*; deben mostrar competencia en técnicas y herramientas de localización e internacionalización; conocer los problemas que surgen a partir de las distintas maneras de codificar caracteres; deben mostrar competencia en el uso y aplicación de las herramientas de TAO; y deben ser competentes en varios idiomas extranjeros, especialmente los que son ajenos al equipo de desarrollo del *software* (*Ibidem*, p. 30).

Jiménez-Crespo (2013) propone que el localizador profesional posee un perfil doble. De un lado tiene una serie de competencias traductoras avanzadas y del otro posee *destrezas* en tecnología y gestión:

> […T]he two main components of any professional localizer are an advanced translation competence and a degree of technological and management skills […]. Professional localization can then be conceived as an extension or addition to general translation competence […].
>
> (*Ibidem*, p. 165)

También presenta al localizador como un actor con funciones distintas a otros participantes del proceso localizador y que es necesario definir "[…] what the role is of the professional 'localizer' as opposed to a 'localization engineer', 'technical translator', 'localization manager' or 'technical translator'. […]" (*Ibidem*, p. 162). Es importante subrayar que lo que el autor

intenta es definir los perfiles formativos de los localizadores y que estas distinciones son necesarias para poder describir las competencias adicionales que necesitaría un traductor.

Aun cuando la función y la participación del localizador en el proceso de desarrollo de programas informáticos no ha cambiado, la expectativa es que cada vez más incluya en su repertorio de conocimiento destrezas técnicas:

> [… I]t looks likely that while translators will be able and expected to increasingly focus on their linguistic tasks in localization, the bar of technical complexity will be raised considerably as well. This applies not just to software localization, but also to the wider context of content localization.
>
> (Esselink 2006, p. 29)

El localizador de nuestro presente no se puede conformar con las destrezas de toda la vida. Tiene que evolucionar al mismo ritmo que la tecnología evoluciona y sumar valor añadido a lo que oferta como participante de un proyecto de localización.

2.1.3 Internacionalización

Veamos varias definiciones de internacionalización. Esselink enfatiza tres aspectos importantes en la internacionalización. El primer aspecto es dar apoyo a conjuntos de caracteres internacionales, que incluye no solo posibilitar la escritura y almacenamiento de caracteres diversos que den apoyo a idiomas con alfabetos no latinos, sino, además su despliegue y manejo adecuado en la interfaz. El segundo aspecto que presenta es la separación del código de las cadenas de texto traducibles, que facilita el trabajo del localizador a la hora de traducir y evita problemas dentro del programa pues el localizador no tiene que "meterse" dentro del programa a buscar las cadenas de texto. El último aspecto posibilita la adaptación del código a las necesidades específicas del mercado. Estas necesidades específicas pueden estar relacionadas con requisitos legales que aplican a ciertos mercados, o con funcionalidades específicas al usuario o a la empresa que la usa (personalización), por ejemplo.

> Internationalization refers to the adaptation of products to support or enable localization for international markets. Key features of internationalization have always been the support of international natural language character sets,

> separation of locale-specific features such as translatable strings from the software code base and the addition of functionality or features specific to foreign markets. Without internationalization, localizing a product can be very challenging.
>
> (*Ibidem*, p. 23)

Las siguientes definiciones enfatizan, sobre todo la extracción de los elementos culturales o la abstracción de estos elementos aparte del producto. Esta abstracción está en consonancia con el segundo aspecto presentado por Esselink.

> **Internationalization is the process of enabling a product at a technical level for localization.** [… It] primarily consists of abstracting the functionality of a product away from any particular culture, language or market so that support for specific markets and languages can be integrated easily.
>
> (Lommel 2007, p. 17, énfasis en el original)

> Internationalization is a way of designing and producing products that can be easily adapted to different locales. This requires extracting all language, country/regional and culturally dependent elements from a product. In other words, the process of developing an application whose feature design and code design do not make assumptions based on a single locale, and whose source code simplifies the creation of different local editions of a program, is called internationalization.
>
> (The Localization Institute 2012)

Apple tiene un acercamiento bastante simplista que refleja su postura de que la internacionalización no es más que una traducción de cadenas de textos. Aunque reconocemos que sus sistemas operativos ofrecen una amplia funcionalidad para facilitar la internacionalización de sus aplicaciones, sigue siendo un acercamiento *naïf* en cuanto a cuán complicado puede ser crear una aplicación que se adapte adecuadamente a su mercado meta.

> Internationalization is the process of making your app able to adapt to different languages, regions, and cultures.
>
> (Apple Inc. 2015a)

GALA ofrece una definición bastante simple también, pero de ella nos interesa más resaltar el aspecto dinámico del diseño cuando dice que es un producto que, sin necesidad de rediseñarlo, puede dar apoyo a múltiples idiomas y culturas. La definición también vincula este proceso con el de localización.

> Internationalization is a design process that ensures a product (usually a software application) can be adapted to various languages and regions without requiring engineering changes to the source code. Think of internationalization as *readiness* for localization. Internationalization can save significant expense, time, and headaches for everyone involved […].
>
> (GALA 2016b)

Estas definiciones subrayan la facilidad de adaptación, la generalización y la abstracción en el proceso de la internacionalización, que permiten que el proceso de localización sea más fluido tanto para quien realiza la localización como para quien la solicita (Esselink 2000, p. 26) y evitan que durante el proceso de localización surja la necesidad de adaptar las funcionalidades del producto para poder suplir las necesidades específicas de otros mercados. Por ejemplo, si se modularizan[95] todos los textos y otros recursos (las imágenes, por ejemplo) que puedan depender de la cultura de un programa (o aplicación), en principio no hace falta adaptar ninguna funcionalidad. Bastará sustituir el módulo de un lenguaje por otro. Por otro lado, la forma en que se muestra una fecha cambia según cada *locale*, incluso los tipos de calendarios que se usan. Un programa que está internacionalizado no realiza cálculos de fecha ni gestiona

[95] Modularizar significa convertir en módulos. Los programas que constituyen una aplicación pueden estar divididos en distintos módulos especializados. El efecto de modularizar es que se crean contenedores independientes que segmentan cada parte del programa. Estos contenedores se comunican por medio de mensajes que se envían el uno al otro o por medio de parámetros que se intercambian al momento de ejecutar un módulo, por ejemplo.

la presentación de una fecha directamente. Se vale de bibliotecas[96] externas, que son *grosso modo* colecciones de programas externos, que se encargan de calcular diferencias entre fechas y del formato de presentación de la fecha según el *locale* que el usuario haya seleccionado.

Bass, John y Kates (2001) explican las ventajas que ofrecen los sistemas que dan apoyo a la internacionalización desde el punto de vista comunicativo o del usuario. Primero, un sistema que da apoyo a la internacionalización permite que el usuario se comunique con dicho sistema usando el idioma que mejor conoce. En segundo lugar, el hecho de que un sistema incluya funcionalidades para admitir la internacionalización permite que se localicen los avisos que emite el sistema, algo que facilita al usuario la resolución de problemas, pues, cuando ocurre un error, cualquier aviso se presenta en un idioma que este usuario entiende bien. Un mensaje de error en un idioma incomprensible agrava el estado de confusión que este tipo de mensajes genera en un usuario. Por otro lado, el apoyo a la internacionalización también propicia el aprendizaje del usuario porque la retroalimentación que recibe es en un idioma que conoce bien: al facilitarle la comprensión propicia el aprendizaje. También, "[…b]eing able to communicate with a system in the language that a user knows best reduces frustration and increases user satisfaction by affirming the importance of the user's national or cultural identify […]" (*Ibidem*, pp. 28-29). Pero la internacionalización no solo facilita el aspecto comunicativo de la aplicación, sino que reduce costos a largo plazo. "[…] When a program is not internationalized, it may still be possible to have it localized, but the localization process will be difficult and costly. […]" (Roturier 2015, p. 49).

O'Hagan y Mangiron (2013) sugieren que la internalización de un producto mejora el proceso de desarrollo de *software* pues le da preeminencia al contenido y permite que la

[96] "A software library generally consists of pre-written code, classes, procedures, scripts, configuration data and more. Typically, a developer might manually add a software library to a program to achieve more functionality or to automate a process without writing code for it. For example, when developing a mathematical program or application, a developer may add a mathematics software library to the program to eliminate the need for writing complex functions. All of the available functions within a software library can just be called/used within the program body without defining them explicitly. Similarly, a compiler might automatically add a related software library to a program on run time." (Janssen 2016, s. v. software library)

localización se posicione en un primer plano principal durante el proceso de diseño del contenido original. Este posicionamiento ahorra tiempo y dinero en el desarrollo.

> […] Internationalization in effect pushes the localization process upstream so that localization can be foregrounded within the design of the original source content. [… It] was developed as a means of heading off at an early stage of product development any mayor localization challenge and is a logical approach to avoiding costly and time-consuming reengineering […].
>
> (*Ibidem*, p. 90)

Estas investigadoras trabajan con la localización de videojuegos. Esta industria, por la naturaleza del producto que vende, se ha visto obligada a incluir al localizador y a otros conocedores del impacto que tiene la cultura en los mercados a lo largo del desarrollo del proyecto. Las autoras hacen referencia a los comentarios de un experto de la industria que ha trabajado en proyectos con equipos multidisciplinares.

> […T]ranslators, editors, and localization producers work alongside the development team […]. Such an arrangement reflects recognition by the company of the critical link between game development and game localization, conductive to the creation of culturally appropriate target-language versions that respect the original intention of the game designer […].
>
> (*Ibidem*, p. 181)

Adoptar un equipo en que trabajen los desarrolladores junto con los especialistas en la localización a lo largo de todo el proyecto refleja la importancia que tiene para la empresa el hecho de crear un producto que sea aceptable en las versiones que se crearan para cada mercado.

2.1.4 Globalización

Es importante señalar que la internacionalización debe mirarse desde la perspectiva de los negocios, considerando el posicionamiento del producto o servicio en el futuro. Asimismo, requiere que la empresa se involucre a todos los niveles en el desarrollo del producto o servicio (Lommel 2007, p. 18). Esta integración vertical en la empresa se lleva a cabo por medio del proceso de globalización, que GALA define como:

> […] a broad range of processes necessary to prepare and launch products and activities internationally. […G]lobalization is an all-encompassing concept which applies to activities such as multilingual communication, global-readiness of products and services, and processes and policies related to international trade, commerce, education, and more […].
>
> (GALA 2016a)

Esselink nos explica que el término globalización se usa de distintas maneras, según el nivel empresarial al que nos estemos refiriendo. Puede estar relacionado con la globalización a nivel económico de la empresa, o con el establecimiento de oficinas o sucursales en el extranjero; puede estar relacionado con la creación de un sitio web multilingüe o con la adaptación multilingüe de los programas empresariales para dar apoyo a dichas operaciones en el extranjero (2000, p. 4). Cuando una empresa acoge una estrategia global, tiene la necesidad de localizar: "[…l]ocalization has gained recognition […] as an essential industrial process required by businesses for the efficient globalization of products in electronic form […]" señalan O'Hagan y Mangiron (2013, p. 87). En cualquier caso, la globalización es más un asunto de filosofía empresarial para gestionar y dar apoyo a los procesos de internacionalización, localización y traducción, pero ello no lo hace menos importante puesto que tiene gran importancia a la hora de decidir y, sobre todo, de dar apoyo a estos procesos.

2.2 Localización y traductología

En *Translation and Web Localization*, Jiménez-Crespo (2013) dedica una breve discusión a este tema y nos explica el posicionamiento de la localización dentro de la traductología desde varios puntos de vista, según esta área de conocimiento ha ido creciendo. Partimos del texto seminal de Esselink (2000), desarrollado totalmente desde la perspectiva profesional, que cubre todo lo relacionado con (lo que en aquel momento era) el nuevo campo: definiciones, relaciones entre el campo de la traducción y la ingeniería de *software*, control de calidad, traducción de *software* y materiales relacionados, gráficos, tecnología, gestión de proyectos, *desktop publishing*, entre otros. Jiménez-Crespo indica que esta tendencia descriptivista la propician expertos de la industria que migraron a la academia y que contrasta con la de los académicos que insistían en que la localización no era distinta a la traducción de

toda la vida. El otro debate gira en torno al espacio que ocupa la localización dentro de la disciplina o dentro de qué rama se debe estudiar, si es un fenómeno que merece estudio aparte por su enraizamiento en la tecnología o si es una modalidad más de la traducción. En su sentencia final, el autor propone que la localización "[…] can be easily conceptualized as a technology-based translation modality that requires the collaboration of a number of agents in addition to translators" (2013, pp. 20-22).

La breve discusión de Jiménez-Crespo resume lo que pensamos que son las posturas que han regido el desarrollo de esta industria: un modelo que limita y delimita los trabajos y las funciones de todos los agentes que se supone laboren en colaboración. Por un lado, y en posición dominante, están los desarrolladores de *software* que solo necesitan que les traduzcan las palabras que aparecen en las interfaces, y por el otro, los localizadores, en posición de dominados, que se limitan a extraer textos de los programas, traducirlos y reincorporarlos a ellos. Esta postura persiste[97], con lo cual reduce y quizás elimina las posibilidades de entrar en un proceso de creación multidisciplinar e interdisciplinar de desarrolladores de programas y localizadores expertos en sistemas capaces de funcionar en múltiples idiomas cuando sea necesario.

Hemos visto de dónde surge la industria de la localización, cuáles son los conceptos relacionados con ella (GILT) y cómo se percibe esta área de estudios en los estudios de traducción. A continuación, haremos un resumen crítico de las investigaciones sobre localización de *software* que se han llevado a cabo. Esto está en consonancia con los objetivos A y B, y con las preguntas de investigación A y B.

[97] Esselink piensa que la tendencia es que los traductores sigan siendo traductores y que las tecnologías lo que han hecho es distraerlos. "[…] Throughout the 1990s, the localization industry tried to turn translators into semi-engineers. Is it now expecting them to just translate again? It certainly looks that way. For the past decades, content authors and translators may simply have been "distracted" by the possibilities and the features the new technologies had to offer—all those file formats, all those compilers, all these new tools, all the output formats, all those cool graphics and layout features! […]" (2006, p. 28)

2.3 Investigaciones previas

La localización de *software* como área de estudio aplicada ha sido poco investigada y siempre desde el punto de vista de la programación. Las investigaciones que estudiaremos a continuación consideran la traducción un proceso subordinado al desarrollo de *software*. A fin de entender cuál es el espacio que ocupa el localizador en la actualidad y qué lo conforma, veamos un resumen de las investigaciones realizadas.

2.3.1 El primer *framework*: Young (2001)

La primera investigadora del tema fue Erica Young, quien en su tesis "A Framework for the Integration of Internationalization into the Software Development Process" (2001) comienza presentando un breve trasfondo histórico de la localización. La autora explica que Alvin Yeo separa las áreas en que la localización afecta a un programa y las divide en dos: los factores manifiestos (*overt*) y los no manifiestos (*covert*). El factor manifiesto más evidente es el idioma, que tiene un efecto significativo sobre cómo se representan internamente los caracteres, qué constituye un elemento textual para efectos de ordenación y qué particularidades del idioma hay que considerar al momento de desplegarlo en la interfaz. La morfosintaxis de cada idioma, además, afecta la concatenación de las cadenas de texto, un recurso muy utilizado por los programadores para ahorrar tiempo. Otros elementos relacionados con el idioma y con la cultura, como por ejemplo los formatos de fecha, moneda, tiempo, dirección, teléfonos, unidades de medida y disposición de los días de la semana en el calendario, también forman parte de estos factores manifiestos. Los factores no manifiestos son más subjetivos y se refieren a las metáforas que usan las interfaces, como lo son el *desktop*, las ventanas, cortar y pegar, entre otros, y los elementos visuales, como los iconos, los gráficos y la selección de colores (*Ibidem*, pp. 5-13).

Young presenta las dos maneras en que se traducen los programas en ese momento. El método *remove/replace*, que requiere que el programa esté terminado en su totalidad (o casi terminado), implica buscar dentro del programa todas las instancias en que aparecen textos en el idioma fuente y reemplazarlos por los correspondientes textos del idioma meta. El primer problema de este método es que no permite que se comercialicen las versiones traducidas en el

mismo momento en que se lanzan las del texto fuente, porque traducir todo el programa tomaba unos seis meses. El segundo y más grande problema era dar mantenimiento al programa, pues terminaba habiendo una serie de programas en distintos idiomas que para fines prácticos eran distintos entre sí. Añadir nuevas funcionalidades o lanzar nuevas versiones implicaba comenzar de nuevo con la traducción, y descubrir problemas de programación se convertía en otro gran dolor de cabeza porque los cambios que se introducían en el código eran tantos que se dificultaba dar con estos errores. El otro método usado era el método de *add-on*, donde no se tocaba el código original, sino que se añadía funcionalidad para cada *locale* que se pretendía crear. Este método no solo multiplica el esfuerzo en función de la cantidad de idiomas meta que se vayan a comercializar, sino que también crea un programa gigantesco que los equipos apenas pueden ejecutar. En los apéndices de su tesis incluye ejemplos de programas que siguen estas estrategias de localización (*Ibidem*, pp. 15-17).

La autora describe, sin profundizar mucho, el concepto de la internacionalización como un proceso de generalización y aislamiento de las variables culturales. Luego habla sobre el estado de la internacionalización en cuanto a avances técnicos, estandarización e investigación (*Ibidem*, pp. 17-27). Cuando desarrolla su *framework*[98] preliminar, en el que inserta la internacionalización dentro del proceso de desarrollo de *software*, específicamente en el modelo de cascada[99], parte de la premisa de que durante sus investigaciones descubrió que existía una carencia de instrucción en los desarrolladores de programas sobre cómo aplicar la internacionalización al desarrollo de *software*: "[…] I found that there was a lack of practical advice on integrating internationalization into the standard procedure for software development […]" (*Ibidem*, p. 28). Como veremos más adelante[100], once años después Murtaza y Shwan se dan cuenta de que "[…] the problem was due to obliviousness of local software

[98] "A framework is a reusable design for all or a part of a software system. Framework is made up of a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions" (IBM, 1999 en Venkatesan 2008, p. 29).

[99] SDLC. Véase el apartado "La era de metodología antigua" en la página 46.

[100] Véase página 124.

developers about the numerous solutions, tools and approaches for multilingual software development […]" (2012, p. 36). Con el pasar de los años, los consejos prácticos se han convertido en más que eso, se han convertido en herramientas, nuevas prestaciones de las plataformas de desarrollo y estrategias de desarrollo de *software* multilingüe, pero seguimos con el mismo problema de no lograr armonizar los conocimientos informáticos con los conocimientos lingüísticos y culturales.

Este *framework* toma forma según la autora va documentando el proceso de desarrollo de una sencilla aplicación bilingüe que desarrolla siguiendo las etapas del SDLC. A lo largo de este proceso describe el impacto que tiene la internacionalización en cada una de las etapas del modelo de cascada. Todas estas explicaciones y las sugerencias que hace surgen de su reflexión durante este proceso, y esta experiencia le permite señalar muchos de los problemas comunes que enfrentan los programadores al desarrollar aplicaciones multilingües. Termina presentando

la versión final de su *framework*, resumido en la Figura 2.1, basado en la experiencia que tuvo al desarrollar dicho *software*.

Summary of the Final Framework

1.0 Requirements Phase

1.1 Determine initial and future target locales.
1.2 Perform requirements analysis for each initial and future target locale.
1.3 Based on the requirements for each locale, determine if it is necessary and logical to internationalize this software product.

2.0 Specification Phase

2.1 Outline the environment in which the application must run including any features or constraints the environment provides to the internationalization process.
2.1.1 What operating system(s) must this program run on? What facilities/constraints does it provide? What locales does it support?
2.1.2 With what other software must this program be compatible? How internationalized is that software? What locales does it support?
2.2 Based on the general information gained about the specifics of the application, decide if it is still feasible to internationalize this application.
2.3 Decide which locales will be developed now and which locales will merely be enabled (if applicable) for later development.
2.4 Decide on a character set or sets.
2.5 Finalize the level of internationalization that will be attained in the development of this software product.
2.6 Identify which processes, modules or objects will be affected by internationalization.
2.7 Determine who will handle localization and at what point in the development cycle localization will begin.

3.0 Design Phase

3.1 Choose a language with adequate support for the level of internationalization to be used for this project.
3.2 Determine what localization information and functions will be handled by the outside environment and which must be handled by the developed application.
3.3 Determine how the locale files will be laid out and accessed.
3.4 Compile the locale files and write instructions on how to localize them. Assemble the localization kit.
3.5 If the product will not be fully internationalized, document areas that are left uninternationalized (in case future internationalization will be done).

4.0 Implementation and Integration Phase

4.1 Thoroughly test the product for all locales including the scripts and character sets used, localized operating systems and local third-party software.

5.0 Maintenance Phase

5.1 Perform steps 1 to 4 in the small.

Figura 2.1 - Resumen del framework *de Young (2001, pp. 81-82)*

2.3.1.1 Un buen punto de partida, en su momento

Este es el primer estudio que se hace a fin de considerar cómo afecta el proceso localizador al proceso de desarrollo de programas informáticos multilingües y por ello fue, durante siete años (que en informática significa un largo tiempo), el referente para muchos de los que se acercaron por primera vez a la localización. Aunque es una buena guía para el momento en que se publica, la autora abre la puerta para que no se internacionalice la aplicación en muchos de los pasos de las etapas del SDLC, como cuando dice "[…b]ased on the general information gained about the specifics of the application, decide if it is still feasible to internationalize this application" (Young 2001, p. 81) y en los puntos 1.3 y 2.2 de la

Figura 2.1. En la actualidad, un proyecto de *software* que se cuestione la viabilidad de internacionalizar limitaría las posibilidades de expansión hacia los mercados mundiales, algo que no era tan evidente ni real en el momento en que la autora escribió su propuesta, sobre todo si lo comparamos con nuestra actualidad. No internacionalizar, como una decisión consciente de un equipo de trabajo de un proyecto, es cerrar las puertas a mercados que podrían expandir los beneficios económicos del programa en desarrollo y ampliar el ciclo de vida del producto. Incluso en países donde se hablan múltiples idiomas, como en la India, se pueden ver excluidos varios segmentos dentro de un mismo mercado nacional[101].

Young tampoco discute nada desde el punto de vista de la traducción de la aplicación, puesto que la desarrolla en una combinación de idiomas que ella conoce. Al ser ella misma programadora y traductora, no experimenta las dificultades que experimenta un localizador, como lo son escudriñar el código para encontrar las cadenas de texto o recibir cadenas de texto fuera de contexto. Igualmente, como trabaja sola, la autora no puede experimentar cuál sería la relación entre el equipo de desarrollo y un traductor, no puede determinar cuáles son sus necesidades ni tiene idea de las complejidades del proceso del otro lado de la "caja negra" que produce la traducción. A pesar de esto, y como dijéramos antes, este fue el texto de referencia durante mucho tiempo y sirvió de faro a quienes se interesaron en tratar el tema.

2.3.2 La integración de la localización en el desarrollo de programas informáticos desde el punto de vista de los idiomas bidi: Abufardeh (2008)

No fue sino hasta siete años después que otro programador se interesó en el tema de la localización de *software* como parte del proceso de desarrollo de programas. La tesis doctoral de Sameer O. Abufardeh, "A Framework for the Integration of Internationalization and Localization Activities into the Software Development Process" (2008), es un trabajo muy abarcador y completo, como veremos a continuación. El autor presenta las definiciones de rigor, y con la finalidad de entender las dificultades inherentes a la localización, dedica los

[101] Véase la cita de Venkatesan que comentamos en la página **Error! Reference source not found.**4.

capítulos iniciales a indagar sobre qué factores inciden sobre ellas. Para ello, divide la ortografía de los idiomas en dos tipos (pictográficos y ortográficos), e informa sobre los esfuerzos e investigaciones hechas sobre idiomas unidireccionales (o principalmente indoeuropeos) y bidireccionales (bidi)[102] (*Ibidem*, pp. 5-10). Hace referencia a las prácticas prevalentes en la internacionalización (*Ibidem*, pp. 39-42) y los estándares internacionales relacionados con la globalización (*Ibidem*, pp. 16-17), y explica qué son los conjuntos de caracteres (ASCII/Unicode)[103] y qué ventajas ofrece Unicode sobre otros sistemas de conjuntos y codificación de caracteres, las normas ISO/IEC[104], los formatos de intercambio (OLIF, XLIFF, TMX, TBX[105]) que se usan en el ámbito de la localización (*Ibidem*, pp. 17-39). Habla del *framework* GNU gettext[106], que es el que más se usa en los programas de uso libre y código

[102] Los idiomas bidireccionales o "bidi", como el árabe y el hebreo, son idiomas que se escriben de derecha a izquierda (RTL) pero que tienen caracteres, como los numerales, que se escriben de izquierda a derecha (LTR). Los unidireccionales se escriben LTR o RTL únicamente.

[103] Sobre Unicode véase el apartado "Codificación de caracteres, Unicode" en la página 170.

[104] Específicamente menciona las normas vigentes en ese momento: ISO/IEC DTR 11017:1998 "Framework for Internationalization" (aún vigente), ISO/IEC 14651:2007 "International string ordering and comparison" e ISO/IEC DTR 14652:2004 "Specification methods for cultural conventions" de la Organización Internacional de Normalización. Esta última ha sido anulada y sustituida en 2014 por la ISO/IEC TR 30112:2014 "Specification methods for cultural conventions". La ISO/IEC 14651:2011 es la versión actual de la ISO/IEC 14651:2007 e incluye dos enmiendas hechas en 2012 y 2015: ISO/IEC 14651:2011/Amd 1:2012 e ISO/IEC 14651:2011/Amd 2:2015 (International Standards Organization 2015).

[105] OLIF, siglas de *Open Lexicon Interchange Format*, es un formato abierto para codificar e intercambiar datos léxicos y terminológicos (OLIF Consortium 2008). XLIFF (*XML Localisation Interchange File Format*) es una especificación basada en lenguaje de marcación ampliable (XML) para intercambiar datos localizables e información relacionada entre herramientas (OASIS 2008). El formato TMX especifica un método para describir los datos contenidos en una memoria de traducción a fin de que esta se pueda intercambiar entre herramientas o proveedores de servicios de traducción sin que se pierda información vital durante este proceso (ETSI 2013, p. 5) y el TBX (Term Base eXchange) es un formato de intercambio de datos terminológicos estandarizado por ISO 30042:2008 "Systems to manage terminology, knowledge and content – TermBase eXchange (TBX)".

[106] *GNU gettext* es parte del GNU Translation Project y brinda una colección de herramientas integradas y documentación para programadores y traductores. "[… T]he GNU 'gettext' utilities are a set of tools that provides a framework to help other GNU packages produce multi-lingual messages. These tools include a set of

abierto (*Free and Open Source* Software, FOSS), Natural Language Support Objects (NLSO), que usa bases de datos con los textos almacenadas en bibliotecas (*Ibidem*, p. 42) y del método de *wrapper*[107] (*Ibidem*, pp. 42-43). Termina los capítulos iniciales con un resumen del estado en que se encuentra la globalización, internacionalización y localización y concluye que existe poca investigación en el ámbito de la integración de las actividades de globalización[108] en el desarrollo de *software* (*Ibidem*, p. 46) y mucho menos en el área de *software* bidi.

La manera en que el autor categoriza los niveles de internacionalización/localización refleja un buen acercamiento para entender el alcance y la profundidad en que se puede integrar la internacionalización/localización dentro de los programas y en el material de apoyo: solo inglés, inglés con funcionalidad para procesar datos "europeos", inglés con funcionalidad para procesar datos del Extremo Oriente, inglés con funcionalidad para procesar idiomas bidi, traducción completa o parcial de la interfaz y la documentación en inglés, y localización completa más prestaciones adaptadas para atender las necesidades específicas del mercado local (*Ibidem*, pp. 49-51). Abufardeh divide la internacionalización en tres componentes: la interfaz del usuario (IU) y la documentación, el almacenamiento de datos y la forma en que se procesa el texto en los programas y los componentes específicos locales y de la cultura correspondiente, y explica cómo cada uno afecta el *software* (*Ibidem*, pp. 52-60). Asimismo, divide la localización en cuatro componentes y también explica su interacción con los programas: el idioma, la disposición de los elementos en pantalla o *layout*, los gráficos, y el *locale* (*Ibidem*, pp. 60-70).

conventions about how programs should be written to support message catalogs, a directory and file naming organization for the message catalogs themselves, a runtime library supporting the retrieval of translated messages, and a few stand-alone programs to massage in various ways the sets of translatable strings, or already translated strings" (Free Software Foundation 2015).

[107] Un *wrapper* es una capa sobre el sistema de I/O que se encarga de interceptar texto de la interfaz y sustituirlo por su adecuada traducción.

[108] En muchas ocasiones a lo largo del texto el autor usa globalización cuando en realidad lo que quiere decir es internacionalización.

El *framework* que Abufardeh presenta se basa en la *Dynamic Systems Development Methodology* (DSDM)[109]. El objetivo del autor es importante: "[…o]ur goal is to weave the globalization[110] requirements and activities into the normal development lifecycle […]" (*Ibidem*, pp. 71-72). Como Young, busca integrar el proceso de internacionalización dentro del desarrollo de programas informáticos.

La DSDM se desarrolla en 1994 y es la primera metodología RAD[111] que se publica. Es similar al método de cascada en cuanto a la estructura por fases, con la diferencia de que no es necesario dar por terminada una fase para poder comenzar con la siguiente. Basta con que una fase se haya completado *lo suficiente* como para que se pueda pasar a la siguiente. Más adelante se puede revisitar la fase anterior y hacer cambios o expandir el alcance (*Ibidem*, pp. 71-73). DSDM tiene como base nueve principios clave:

1. Es absolutamente necesario que los usuarios se involucren de manera activa.
2. Los equipos deben poder tomar decisiones vinculantes.

[109] El autor presenta tres razones principales para escoger la DSDM: en primer lugar, porque piensa que más que una metodología es un *framework*, un esqueleto de procesos y descripciones de productos que se pueden adaptar y servir como fundamento para cualquier tipo de proyecto. La segunda razón es porque su ciclo de vida es iterativo, funciona por incrementos y permite entregas rápidas, pues es una metodología ágil (véase próximo párrafo). La tercera razón es porque enfatiza en desarrollar programas que atiendan las necesidades del usuario, es decir, que está centrada en el usuario. Al estar centrada en el usuario, se promueven las pruebas y el control de calidad, elementos importantes al desarrollar programas multilingües.

Las metodologías ágiles para el desarrollo de *software* abarcan "[…] a group of software development methods based on iterative and incremental development, where requirements and solutions evolve extremely rapidly. […]" (Roturier 2015, p. 17) Se basan en el "Manifiesto por el desarrollo ágil de *software*" de donde derivan los "Doce principios del software ágil". Para más información, véase (Beck et al. 2001a, 2001b).

[110] Internacionalización. Véase nota 108.

[111] El DSDM Consortium define RAD o *Rapid Application Development* como "[…] a project delivery framework which actually works. It aids the development and delivery of business solutions to tight timescales and fixed budgets […]", aunque en la actualidad se alinean más con el movimiento ágil que con el RAD (Avison y Fitzgerald 2006a, p. 473). "The mission of the DSDM Consortium is to develop and promote best practice and learning for successful Agile projects" (DSDM Consortium 2015).

3. El enfoque del *framework* es en que se entreguen resultados (*deliverables*) con frecuencia.
4. El criterio esencial para aceptar estos resultados es que estén alineados con los objetivos o fines del negocio.
5. El desarrollo iterativo y progresivo es necesario para converger en una solución para el negocio que se adecúe con precisión (*accurate business solution*).
6. Todos los cambios hechos durante el desarrollo se pueden revertir.
7. Los requisitos se describen a un nivel alto (es decir, más generalizado).
8. Las pruebas se integran a lo largo de todo el ciclo de vida de la DSDM.
9. Es esencial que entre las partes implicadas exista un acercamiento colaborativo y co-operativo (*Ibidem*, p. 74).

Para crear su nuevo "Global Software Development Life Cycle—Dynamic Model", Abufardeh integra las actividades relacionadas con el proceso de internacionalización y localización a este *framework*, explica las fases sobre las cuales se basa esta metodología y por qué le parece adecuada para la internacionalización. El autor aborda cada fase de la DSDM en detalle a fin de explicar los aspectos de la internacionalización y la localización que inciden sobre las actividades que se realizan en cada una de ellas, y el impacto que estos aspectos tienen sobre la DSDM y el proyecto.

2.3.2.1 Fase de estudio de viabilidad y estrategia de desarrollo

Un análisis del mercado indicará a los desarrolladores del proyecto cuáles son los posibles *locales* que se deben implementar. El autor reduce los niveles de localización e internacionalización mencionados en la página 85 a tres posibles casos: producto existente creado sin considerar internacionalización ni localización que hay que internacionalizar y localizar, producto nuevo diseñado y creado considerando la internacionalización y localización, y producto previamente internacionalizado que hace falta localizar. El estudio de viabilidad facilitará la toma decisiones en cuanto a tiempo, costos y riesgos, según los *locales* que se pretenda implementar. También en esta etapa se debe considerar cómo llevar a cabo el proceso de localización, es decir, si se pretende usar recursos internos o externos, y qué

herramientas (por ejemplo, de TAO) hay disponibles para facilitar la traducción. La última consideración importante en esta etapa está relacionada con los conjuntos de caracteres, que tienen un impacto importante sobre los posibles *locales* que se pueden implementar (*Ibidem*, pp. 80-86).

2.3.2.2 Fase de identificación de requisitos

Las aplicaciones cuentan con unos requisitos intrínsecos con relación al dominio de la aplicación[112] en sí a los cuales se les añade requisitos lingüísticos, de formato, legales y culturales que derivan de la localización e internacionalización. Categorizar y priorizar estos requisitos ayuda a determinar el impacto que tendrán sobre el desarrollo de la aplicación. Los requisitos de desarrollo de *software* tradicional se categorizan en tres tipos: requisitos que introducen una funcionalidad o prestación totalmente nueva, requisitos que introducen modificaciones significativas a funcionalidades o prestaciones existentes, y requisitos que introducen modificaciones leves a funcionalidades o prestaciones existentes. Estas categorías se pueden usar para clasificar el impacto de la internacionalización sobre estas funcionalidades y prestaciones nuevas o existentes a fin de tener idea de la complejidad que se le está añadiendo al proyecto. Luego se pueden priorizar los requisitos usando niveles de prioridad o el método MoSCoW[113]. Otros requisitos importantes que afectan el desarrollo de *software* internacionalizado son los requisitos de las BD[114], el lenguaje de programación que se va a usar y sobre qué entorno (SO[115], requisitos de redes, compatibilidad con programas de terceros) el producto localizado funcionará (*Ibidem*, pp. 86-97).

[112] Véase nota 128.

[113] ***M[o]****ust have,* ***S****hould have,* ***Co****uld have,* ***W****ould not have*. MoSCoW es una técnica de priorización que ayuda a entender y gestionar prioridades (DSDM Consortium 2014).

[114] Bases de datos.

[115] Sistema operativo.

2.3.2.3 Fase de análisis y diseño

Abufardeh plantea que durante el **análisis** es importante separar los aspectos lingüísticos y culturales del núcleo (*core*) de la aplicación. Si consideramos el modelo a un nivel de abstracción alto, se puede dividir el sistema en tres estratos. En el *estrato de presentación* es donde se hace la localización de la interfaz, y es la parte con la cual el usuario interactúa directamente. El *estrato de aplicación o lógica del negocio* incluye la parte de la localización que está relacionada con procesos[116]. El *estrato de repositorio de datos* dicta, por lo general, cómo funcionan los estratos superiores (porque almacena los datos que se procesan en el estrato de aplicación y que se presentan en el estrato de presentación) y es el que toma en consideración el almacenamiento de los datos persistentes[117]. La Figura 2.2 muestra cada uno de estos estratos,

[116] La parte de la localización que está relacionada con los negocios incluye procesos lógicos específicos para un *locale* que no necesariamente aplican a los demás. Por ejemplo, la legislación que regula la paga a proveedores en Puerto Rico incluye ciertas particularidades para este *locale*. Un programa que gestione los procesos de paga a proveedores incluirá módulos estándar que funcionarán para todos los *locales* y unos módulos especialmente diseñados para gestionar los pagos a proveedores en Puerto Rico. Estos procesos lógicos se gestionan en el estrato de la aplicación.

[117] *Persistent data*. "Se llama persistencia a la capacidad de guardar la información de un programa [sic] para poder volver a utilizarla en otro momento" (Wachenchauzer et al. 2014). También se puede definir como las características de un objeto y de un proceso que continúan existiendo aun cuando el proceso que las ha creado termina o la máquina donde se ha creado se apaga. Cuando es necesaria la persistencia, el objeto o las características del proceso deben ser almacenados en un equipo de almacenamiento no volátil, como por ejemplo, un disco duro. En cuanto a datos, la persistencia se refiere a que un objeto no se debe borrar a menos que verdaderamente se desee eliminarlo (Janssen 2016, s. v. persistence).

sus relaciones y la manera en que se implementa el modelo multilingüe basado en bibliotecas que propone Abufardeh.

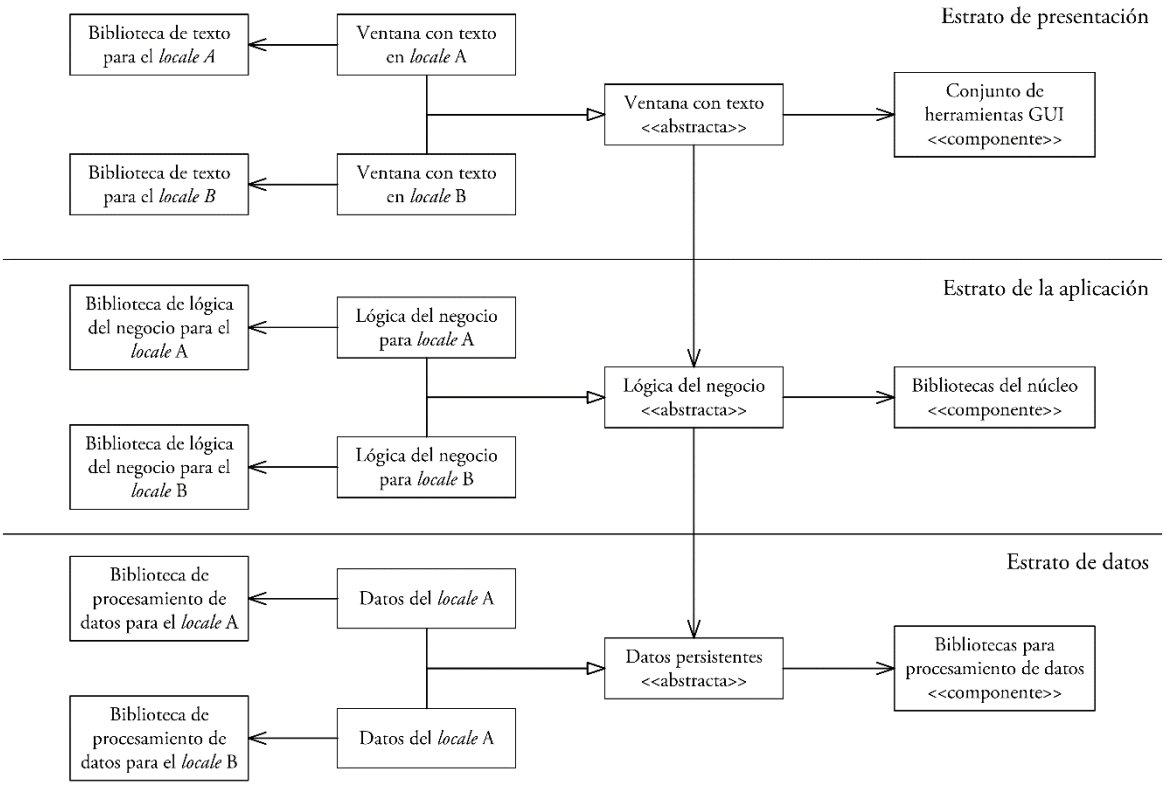

Figura 2.2 - Internacionalización de los estratos de presentación, de la aplicación y de datos (Abufardeh 2008, pp. 98, 100, 102) Los estratos superiores dependen de los estratos inferiores.

Cuando pasamos a la etapa de **diseño**, deben primar los principios de flexibilidad, a fin de que el *software* sea fácil de adaptar a las variaciones locales; de extensibilidad, para posibilitar añadir funcionalidad futura; y de "traducibilidad" medido en términos del tiempo y esfuerzo necesarios para traducir. Durante esta etapa los módulos se separan en cinco tipos: componentes localizados de la IU, componentes o *software* de terceros, componentes del núcleo del producto, componentes de lógica específicos para el *locale* y componentes de procesamiento de datos. Los componentes localizados de la IU deben presentar características visuales fáciles de cambiar para adaptarlos con facilidad a las distintas culturas hacia las cuales se localizará el producto. Asimismo, las consideraciones sobre compresión o expansión de los textos toman precedencia al momento de diseñar estos componentes. Los componentes o programas provistos por terceros que se acoplen interna o externamente al producto deben ofrecer

prestaciones compatibles con los distintos *locales* en los que se espera que la aplicación funcione. Si se ha aislado correctamente del núcleo de la aplicación todo componente que esté relacionado con los distintos *locales*, esta podrá ejecutar sus funciones sin verse afectada por cambios en el *locale*, puesto que los componentes informáticos relacionados con un *locale* específico estarán encapsulados dentro de los componentes de lógica específicos para el *locale*. Abufardeh recomienda que los componentes de procesamiento de datos estén configurados de tal manera que las bases de datos correspondientes se creen de forma dinámica[118], que tomen en cuenta el ancho de las columnas[119] según el tipo de codificación de caracteres[120] y que consideren el formato adecuado para los campos de números, texto, fecha, hora, moneda y medida. El diseño de la base de datos debe ser lo primero que se hace en esta etapa (*Ibidem*, pp. 97-108). La Figura 2.3 nos presenta las relaciones que tienen estos componentes. Al centro, el núcleo de la aplicación, que depende de los componentes de terceros y de las bibliotecas de lógica para cada *locale* para funcionar. Estos tres elementos sirven de apoyo a los componentes localizados de la IU, pero a la vez se sostienen sobre los componentes de procesamiento de datos, que se encargan de manejar la persistencia de estos. Si nos fijamos bien, los tres niveles corresponden a los tres estratos de la Figura 2.2: arriba, el estrato de presentación, abajo, el estrato de datos, y en el medio, el estrato de la aplicación.

[118] La creación dinámica de objetos o, como en este caso, de bases de datos implica que los elementos que los conforman se crean en el momento en que se necesitan, es decir, que no vienen definidos de antemano. Este dinamismo permite que los objetos estén configurados de forma tal que su constitución se adecue a las necesidades del *locale* particular para el cual fueron creados.

[119] Las bases de datos están compuestas por tablas y las tablas se conforman en columnas, que corresponden a cada tipo de información que se almacena, y filas, que corresponden a cada uno de los registros almacenados en la tabla.

[120] Véase el apartado “Codificación de caracteres, Unicode“ en la página 170.

Una vez se estabilizan las especificaciones detalladas del diseño del componente, se empieza a programar y el proyecto entra en la etapa de **codificación**. Es en esta etapa en que el código se internacionaliza, se traducen las cadenas de texto y realizan las pruebas de control de calidad. Es necesario externalizar del código los elementos que dependen de o involucran la cultura, y para ello el autor ofrece una lista de los elementos que dependen directamente de ella y los divide en dos tipos. Por un lado están los elementos de la IU, que incluyen los mensajes y avisos, las etiquetas en la interfaz, el texto de ayuda, los colores, la iconografía y simbología usada, los sonidos, las fechas y horas, las cifras, la moneda, las unidades de medida, los números telefónicos, las direcciones postales y, finalmente, la disposición de los elementos de la interfaz. Por otro lado, las funcionalidades que están ligadas al *locale*, como lo son las funciones de búsqueda, la ordenación, la indexación, y la gramática y ortografía, también tienen que ser tomadas en consideración al momento de codificar. Si existen componentes desarrollados por terceros, se deben integrar con cuidado con los componentes de la aplicación según las exigencias del *locale* y del componente en sí. Abufardeh menciona que hay tres acercamientos en cuanto a cómo se aborda la internacionalización el código, y los codificadores pueden seleccionar uno o una combinación de estos. Los codificadores pueden dejar que los elementos que dependen de la cultura se queden dentro del código, los pueden extraer y colocar en una biblioteca externa, o pueden usar una arquitectura en la cual el núcleo esté completamente desligado de factores culturales y mediante la cual la aplicación accede a los elementos y factores culturales de forma dinámica por medio de paquetes de localización. La selección del acercamiento dependerá de varios

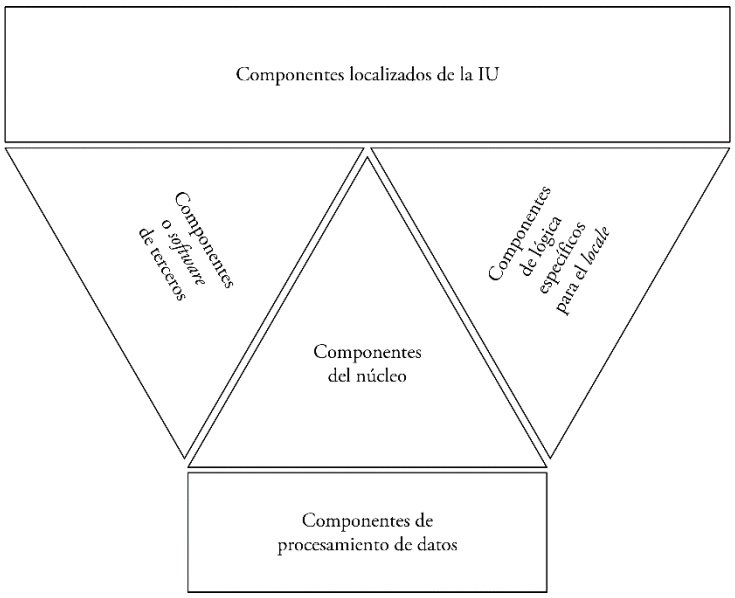

Figura 2.3 - Separación de módulos en la etapa de diseño (Abufardeh 2008, p. 104)

factores, como la naturaleza del proyecto, la experiencia de los desarrolladores y las herramientas que tengan disponibles (*Ibidem*, pp. 108-115).

La fase de **pruebas** incluye dos categorías principales, las pruebas de internacionalización y las pruebas de localización. Cualquier panel de pruebas que se aplique al proyecto se puede ejecutar siguiendo cinco pasos: crear los planes de prueba, preparar el entorno de pruebas, crear los conjuntos de pruebas, ejecutar los conjuntos de pruebas y analizar los resultados de las pruebas. Las pruebas de internacionalización incluyen pruebas de preparación o suficiencia en la localización mediante las cuales se verifica que el código fuente pueda manejar los requisitos de internacionalización sin que se vea afectada la funcionalidad. Las pruebas de preparación o suficiencia se dividen en pruebas del código fuente, que son pruebas mediante las cuales se verifica que el producto funcione adecuadamente en el idioma fuente y pruebas de pseudolocalización, conformadas por pruebas de pseudotraducción, un tipo de prueba que simula el efecto de la traducción en las cadenas de texto y las pruebas de *pseudo-mirroring* (para productos bidi), que simulan un efecto de espejo en la interfaz, conforme los requisitos específicos que tienen los idiomas bidi (*Ibidem*, pp. 115-119).

Para las pruebas de localización existen pruebas de traducción, lingüísticas, de usabilidad, visuales, de integración y prueba sobre la(s) plataforma(s) en que operará el *software* (véase Tabla 2.1). El enfoque se divide en dos áreas: la primera, relacionada con problemas que puedan surgir como resultado de la localización en sí, como lo son problemas en la interfaz (palabras truncadas, por ejemplo) o problemas en los archivos de contenido (por ejemplo, codificación de caracteres[121] incorrecta), y la segunda, relacionada con problemas específicos de la cultura, del lenguaje o del país (*Ibidem*, pp. 120-123).

[121] Véase el apartado "Codificación de caracteres, Unicode" en la página 170.

Translation testing	Verifies translation accuracy and contextual accuracy for the modules and checks for typographical errors.
Linguistics testing	Checks whether the linguistics services provided by the module conform to conventions in the target country or region.
Usability testing	Establishes whether the modules really fit the end-user's cultural expectations. Usability testing is user-driven, not technology-driven and quality is defined by user satisfaction, rather than product/solution defects.
Integration testing	Verifies the correct functionality of the system after integrating the completed module/component. Another important aspect for testing is to verify accurate translation in the context of other modules.
Cosmetic testing	Verifies that the localization process did not introduce any visual errors or any inconstancies in the UI controls and layout. This includes dialog boxes, menus, messages, and report appearance and formatting.
Platform	Setup, upgrade, and uninstall tests run in the localized environment.

Tabla 2.1 - Pruebas de localización según Abufardeh (2008, pp. 121-122)

Finalmente, para la fase de lanzamiento se incluyen las actividades de asistencia y mantenimiento, y en algunos casos, una revisión posimplementación a fin de evaluar el sistema según está siendo usado (*Ibidem*, p. 123).

El autor es muy escueto cuando enumera y describe las cuestiones de colaboración entre los implicados, específicamente los diseñadores, programadores, traductores, *testers* de control de calidad, usuarios/clientes que deben darse durante todo este proceso que hemos descrito antes (*Ibidem*, pp. 124-126). Termina el capítulo explicando en detalle el impacto sobre cada una de las fases de la DSDM que tienen estas cuestiones: el hecho de que los controles de la IU

no deben contener elementos ligados a la cultura (que deben ser *culture neutral*), la direccionalidad del texto, y los editores de métodos de *input* (*input method editors, IME*) (*Ibidem*, pp. 126-129).

Abufardeh dedica un capítulo al problema de localización del árabe y lo explica en detalle (*Ibidem*, pp. 130-152), y otro a presentar una hoja de ruta para hacer pruebas de QA[122] a aplicaciones localizadas al árabe pero que también se aplica a otros idiomas (bidi y no bidi) (*Ibidem*, pp. 153-190). En el siguiente capítulo discute el estado de la "arabización" del *software* (por medio de una encuesta) (*Ibidem*, pp. 191-214) y luego comenta sobre las herramientas para localización disponibles: Alchemy Catalyst, Trados/Passolo, ENLASSO, y termina mostrando las herramientas de localización que provee Visual Studio .NET por medio de ejemplos prácticos e ilustrados, con comentarios y recomendaciones finales (*Ibidem*, pp. 215-258). Termina con un resumen de su investigación y posibles investigaciones futuras (*Ibidem*, pp. 259-269).

2.3.2.4 Se expande y se actualiza el impacto sobre el desarrollo de *software*

Como se aprecia, esta investigación está muy cuidada y aborda todas las consideraciones necesarias para lograr una buena internacionalización integrada a una metodología ágil de desarrollo de *software*. En el capítulo cuatro, que es donde Abufardeh explica cómo se integra el proceso de internacionalización a la metodología DSDM, se explican todos los aspectos significativos que afectan a cada una de las etapas. Pero si leemos la entrelínea, queda claro que la labor del localizador está totalmente subordinada al proceso de desarrollo de *software* y solo se le considera participante *ad hoc* en su producto: la traducción. "[…M]ultilingual […] software requirements are frequently pushed aside by developers during the development process and treated as a text translation job that can be accomplished after the application is completed […]" (*Ibidem*, p. 11), critica el autor al principio de su investigación; sin embargo, su propuesta no incluye ninguna actividad que acerque al localizador y lo haga partícipe en el

[122] Véase nota 93.

proceso de desarrollo de *software*. De hecho, en la página 79 de su tesis presenta la figura 4.3, "Dynamic development & testing processes for I18N & L10N", que consta de dos rectángulos. El superior, titulado "Internationalization (I18N) & Localization Readiness", describe en detalle, mediante cajas y flechas, qué sucede al desarrollar *software* internacionalizado durante las etapas de recopilación de requisitos, diseño, implementación y pruebas de internacionalización; justo debajo hay otro rectángulo, titulado "Localization (L10N) & Translation", cuya única conexión directa es una flecha bidireccional subordinada a las pruebas de internacionalización y una flecha entrecortada etiquetada "feedback" que conecta con la implementación de arriba. La localización y la traducción continúan siendo una caja negra, y el localizador, poco más que un traductor.

En las primeras páginas de su tesis Abufardeh explica que lo que lo motiva a llevar a cabo esta investigación es la carencia de una estrategia de internacionalización y la no-integración de las actividades relacionadas con la internacionalización y localización durante el proceso de desarrollo de *software*.

> […A] key challenge of the software industry [… is] the lack of an effective internationalization strategy as well as the lack of a comprehensive framework for the integration of internationalization and localization activities into the normal software development lifecycle. […][123]
>
> (*Ibidem*, pp. 2, 12)

Los autores que reseñamos adelante también inciden con Abufardeh en cuanto a la integración de las estrategias de internacionalización.

> […T]echnologies and advancements in the field of software development allow us to successfully develop and even modify multilingual software (by adding

[123] Más adelante en el texto lo repite sin ofrecer alternativas integradoras: "[…] We still find that the internationalization and localization activities are treated as two separate processes by the majority of software developers. […] Furthermore, the localization process continues to be narrowly focused on language translation issues and activities. […]" (Abufardeh 2008, p. 44)

> new language versions). However, most developers are oblivious to these technologies and approaches, and project owners fail to correctly prioritize multilingual requirements. […]
>
> (Murtaza y Shwan 2012, p. 18)

Venkatesan no es tan claro en relación con este tema, pero hace referencia a lo mismo: "[i]n the existing multilingual software development approaches, domain[124] functionalities are the major concern and multilingual functionalities are the secondary concern […]" (Venkatesan 2008, p. 32), es decir, que los desarrolladores ponen todo el esfuerzo en desarrollar las funcionalidades inherentes a la aplicación y relegan a un segundo plano las funcionalidades relacionadas con la eventual localización. A pesar de que estos tres autores están de acuerdo en que hace falta establecer propuestas integradoras, ninguno propone soluciones que acerquen o integren al localizador al proceso de desarrollo de *software* a fin de permitirle aportar sus conocimientos como mediador intercultural durante los procesos que solo manejan los desarrolladores. Como veremos en el capítulo 3 y en consonancia con la hipótesis que proponemos, nuestra investigación busca llenar ese vacío.

Sorprende, pues, cuando encontramos que Abufardeh dice, en la sección del capítulo cuatro que dedica a las tareas de traducción de la interfaz durante la fase de codificación de la DSDM, lo siguiente:

> […] Having a translation expert available for the development team will surely have great advantages. There is a great advantage to having a content translation expert in the development team. A translation expert can help in content preparation, organizing the source file structures, and in coordinating testing aspects of the globalized product. Furthermore, the translation expert can help the development team understand and embed globalization requirements such as format and style needs of local languages. This can reduce

[124] Véase nota 128.

> the need for code modifications at later stages which are usually expensive and a cause for delays.
>
> (Abufardeh 2008, pp. 113-114)

Y luego, casi al terminar, en las recomendaciones que da en el capítulo ocho:

> Having a professional native speaker of the targeted languages on site or as a consultant is always a plus. This helps by saving time and money, and it also provides the development team members with rapid feedback throughout the development process.
>
> (*Ibidem*, p. 257)

El autor da justo en el centro de la diana, pero falla en identificar y destacar la importancia de los aportes del experto en traducción a lo largo de todas las fases de desarrollo del proyecto. El experto en traducción no solo es capaz de ayudar a preparar y extraer los archivos para la traducción, y de preparar las pruebas necesarias para asegurar la calidad de la localización, sino que también puede ayudar al equipo a entender el impacto de los requisitos de internacionalización[125] sobre el proyecto. Pero más importante aún, sus conocimientos pueden reducir los costos y retrasos que surgen en el proyecto por no tomar en consideración estos requisitos desde el principio. A pesar de que en la segunda mención que hace en su texto no se refiere a este participante como un experto en traducción sino como un "professional native speaker", la idea de incluir un colaborador que sea capaz de brindar retroalimentación relacionada con el idioma es positiva y podría, en efecto y como él muy bien señala, ahorrar tiempo y dinero. Aun así, podemos intuir que un localizador experto sería una mejor opción, pues se aseguraría de que los resultados, en un proyecto de desarrollo de *software* multilingüe, sean los esperados por las partes implicadas.

[125] Que él llama globalización, véase nota 108.

2.3.3 La arquitectura de *software* como fundamento al desarrollar programas informáticos: Venkatesan (2008)

Venkatesan busca acercarse al *software* multilingüe desde la perspectiva de la arquitectura de *software*, un aspecto que definiremos a continuación. Para ello, en su tesis doctoral, titulada "ARMMS: An Architectural Reference Model for Multilingual Software" (2008), investiga qué es *software* multilingüe. Como parte desde la perspectiva de la arquitectura de *software*, intuye que el desarrollo de *software* multilingüe debe estar basado en ciertos criterios o atributos de calidad[126]. Entonces, a partir de la literatura existente, extrae dichos atributos. Después de llevar a cabo su revisión de literatura, encuentra que no existe ningún modelo explícito de *software* multilingüe, pero entiende que sí puede derivar de ella los modelos de desarrollo de *software* multilingüe actuales y describirlos. Una vez los describe, pasa a analizarlos y a constatar los atributos de calidad que los caracterizan; sin embargo, encuentra un sinnúmero de carencias en ellos, desde la perspectiva de calidad, por lo que decide diseñar y proponer un modelo nuevo para desarrollar *software* multilingüe.

En el capítulo dos, y como resultado de la revisión de literatura, el autor encuentra que el *software* multilingüe existente se puede dividir en tres grupos principales: *software* para la creación de contenido multilingüe (usando editores multilingües), *software* de entornos de procesamiento y de trabajo multilingües (por ejemplo, Windows), y herramientas para el desarrollo de *software* multilingüe (entornos de desarrollo, lenguajes y compiladores, por ejemplo).

[126] *Atributo de calidad* es un concepto fundamental de la arquitectura de *software* y se refiere a las propiedades mensurables y validables que exhibe un sistema; no se refiere a *qué* hace el sistema (una cualidad *funcional*) sino a *cómo* lo hace. "[…] A quality attribute […] is a measurable or testable *property* of a system that is used to indicate how well the system satisfies the needs of its stakeholders. […]" (Bass, Clements y Kazman 2013, p. 63, énfasis añadido.) El autor también se refiere a estos atributos como *cualidades no funcionales relacionadas con el idioma* o *cualidades no funcionales del* software *multilingüe*; las describimos en la página 101. Véase también nota 139.

Venkatesan define *software multilingüe* como aquel que exhibe *concerns*[127] del dominio[128] junto con *concerns* del idioma a fin de que un usuario pueda trabajar en uno o más idiomas, según la selección o configuración del idioma (o de los idiomas). Este tipo de *software*, además, exhibe *cualidades no funcionales relacionadas con el idioma*[129] y se caracteriza por la cantidad de idiomas que se pueden usar, la manera en que cambia de idioma (dinámica o estática[130]), la manera en que incluye o excluye los idiomas (dinámica, al compilar, al ejecutar), los módulos que permiten multilingüismo (módulo de I/O[131], de procesos, de ayuda, de manejo de excepciones, de almacenamiento, etc.), el nivel de multilingüismo que exhibe (a nivel de interfaz, de datos o en ambos niveles) (*Ibidem*, pp. 15-16).

Venkatesan explica que el acercamiento hacia el desarrollo de *software* multilingüe es variado. Está el de desarrolladores privados, donde el mismo desarrollador se encarga de lanzar las versiones en otros idiomas (el desarrollo de este tipo de *software* no se puede investigar

[127] Elrad et al. definen *concern* como "properties or areas of interest" (2001, p. 30) "[…] Un *concern* puede ser definido de una manera genérica como algo de interés para un proceso de ingeniería. […]" (Conejero, Berg y Hernández 2007, p. 224). En castellano se le puede llamar *asunto* a un *concern*. Un ejemplo del uso de *concern* es la separación de *concerns* que estos mismos autores traducen como separación de asuntos. "Concerns can range from high-level notions like security and quality of service to low-level notions such as caching and buffering. They can be functional, like features or business rules, or nonfunctional (systemic), such as synchronization and transaction management" (Elrad, Filman y Bader 2001, p. 30).

[128] El dominio es el área específica en que funciona la aplicación. Serebrenik pone como ejemplo compiladores, equipamiento electrónico para consumidores, sistemas de *e-commerce* o tiendas web, videojuegos, aplicaciones empresariales, entre otros (2014, p. 10).

[129] Son "*[m]aintainability, reusability, understandability, adaptability and language neutrality*", que abordaremos más adelante en la página 101. También las llama *cualidades no funcionales del* software *multilingüe*. Véase nota 139.

[130] Los cambios de idioma dinámicos ocurren al momento en que el usuario decide cambiar el idioma de la interfaz. Si para que tome efecto el cambio de idioma el usuario tiene que salir del programa y volver a entrar (o apagar y encender el equipo), se dice que el cambio de idioma es estático.

[131] *Input/output*.

porque es una suerte de caja negra). También están el acercamiento de código abierto, el de programación (compuesto por programación estructurada, programación orientada a objetos, programación de componentes) y el arquitectural[132] (*Ibidem*, pp. 17-32).

2.3.3.1 Arquitectura de *software*

La arquitectura de *software* se puede definir como un diseño a alto nivel del sistema. La MSDN la define:

> Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.
>
> (Microsoft Developer Network 2009, cap. 1)

Bass, Clements y Kazman definen la arquitectura de *software* de un sistema como el "[…] set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both" (2013, p. 5). La arquitectura de *software* busca crear una serie de metáforas paralelas a la arquitectura clásica[133] a fin de crear representaciones o proyecciones gráficas que permitan entender los sistemas a distintos niveles de abstracción, definir sus interacciones, y especificar las propiedades de estos sistemas y de estas interacciones. De la misma forma en que el arquitecto dibuja planos de planta, secciones, elevaciones y proyecciones, y especifica materiales, cargas, maneras de construir, etc., el arquitecto de *software*

[132] Aquí y en adelante calcamos adrede el término *architectural* del inglés para que se distinga claramente y sin duda alguna que estamos hablando de arquitectura de *software*.

[133] No en el sentido de arquitectura clásica romana, por poner un ejemplo, sino a la Arquitectura como área de estudio. La llamamos "clásica" para distinguirla de la *arquitectura de software*.

representa por medio de dibujos y especificaciones[134] los conceptos que comprenden las partes de los sistemas, sus interrelaciones y qué propiedades tienen y median entre ellos. El propósito es que todas las partes implicadas puedan entender las bases sobre las que se cimienta el sistema y el comportamiento esperado del sistema.

La *abstracción* es fundamental en la arquitectura de *software* y nos valemos de ella principalmente para poder lidiar con la complejidad inherente a los sistemas.

> […A]n architecture is foremost an *abstraction* of a system that selects certain details and suppresses others. […] This abstraction is essential to taming the complexity of a system—we simply cannot, and do not want to, deal with all of the complexity all of the time.
>
> (*Ibidem*, p. 6, énfasis en el original)

Es un recurso que se puede usar en varios niveles en el diseño. A alto nivel la abstracción es más generalizada y a bajo nivel está más orientada hacia procedimientos específicos. El nivel más bajo de abstracción posibilita una implementación directa de sus especificaciones, es decir, es explícita en cuanto a cómo se implementa una especificación. Cuando recurrimos a la abstracción creamos una suerte de máquina virtual que conforma un componente y su función es esconder los procesos internos que ella ejecuta. Las *estructuras arquitecturales* (o de *software*) son abstracciones que proveen un *framework* natural para entender de manera más amplia los *concerns* a nivel de sistema (por ejemplo, tasas de flujo globales, patrones de comunicación, estructura de control de ejecución, evolución del sistema, etc.). Estas estructuras son: de módulo, conceptual o lógica, de proceso o coordinación, física, de usos, de llamadas (*calls*), de flujo de datos, de flujo de control, y de clase. Cada una representa una abstracción con relación a criterios específicos (Venkatesan 2008, pp. 24-25).

[134] Entiéndase aquí y en adelante como especificaciones técnicas, es decir "2. f. Información proporcionada por el fabricante de un producto, la cual describe sus componentes, características y funcionamiento." (Real Academia Española 2014, s. v. especificación).

Las *proyecciones* (*views*) describen el sistema desde distintas perspectivas. Cada proyección del sistema describe la constitución de los componentes, los conectores y datos, y su configuración, junto con las limitaciones o *constraints*. La *proyección conceptual* describe al sistema en término de sus elementos principales y sus interrelaciones. La *proyección de módulos* descompone el sistema y separa los módulos en capas. La *proyección de código* estructura la organización de los objetos, bibliotecas, binarios[135], ficheros y directorios. La *proyección de ejecución* describe cómo se asignan los componentes funcionales a los componentes físicos del sistema y cómo se gestiona la comunicación, coordinación y sincronización entre ellos. Termina explicando qué es un *modelo de referencia*[136], qué es *arquitectura de referencia*[137], qué es un *framework*, y qué y cuáles son las *cualidades arquitecturales*[138]. Luego compara estos cuatro acercamientos en una tabla a fin de contrastar las características y las cualidades que cada uno de ellos aporta (*Ibidem*, pp. 25-33).

[135] Binarios se refiere a programas ejecutables.

[136] "A reference model is an abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment" (Venkatesan 2008, p. 28).

[137] "Reference architecture is a reference model mapped onto software components and defines the data flows between the components. A reference model divides the functionality, whereas reference architecture is the mapping of that functionality onto system decomposition" (*Ibidem*, p. 29).

[138] "Architectural qualities are the key concerns of stakeholders of software. […] Architectures are evaluated by comparing the qualities that they offer. […]" (*Idem*) Las cualidades arquitecturales son: reutilización, modificabilidad, portabilidad, testeabilidad, integrabilidad, ejecución, disponibilidad, funcionalidad, usabilidad y seguridad.

En el tercer capítulo el autor describe las *cualidades no funcionales*[139] *del* software *multilingüe* (véase Figura 2.4). La *modificabilidad* (*modifiability*)[140] indica hasta qué punto un componente multilingüe se puede modificar una vez este componente ha sido lanzado. Es una medida directa del nivel de personalización (*customization*) de las necesidades o *concerns* multilingües (este es el atributo de medida o *attribute for measurement*). La posibilidad de que un segmento de código fuente multilingüe que ya ha sido creado o esté disponible pueda reusarse sin ninguna o poca modificación es lo que se conoce como *reutilización* (*reusability*). Cuando hablamos de *comprensibilidad*

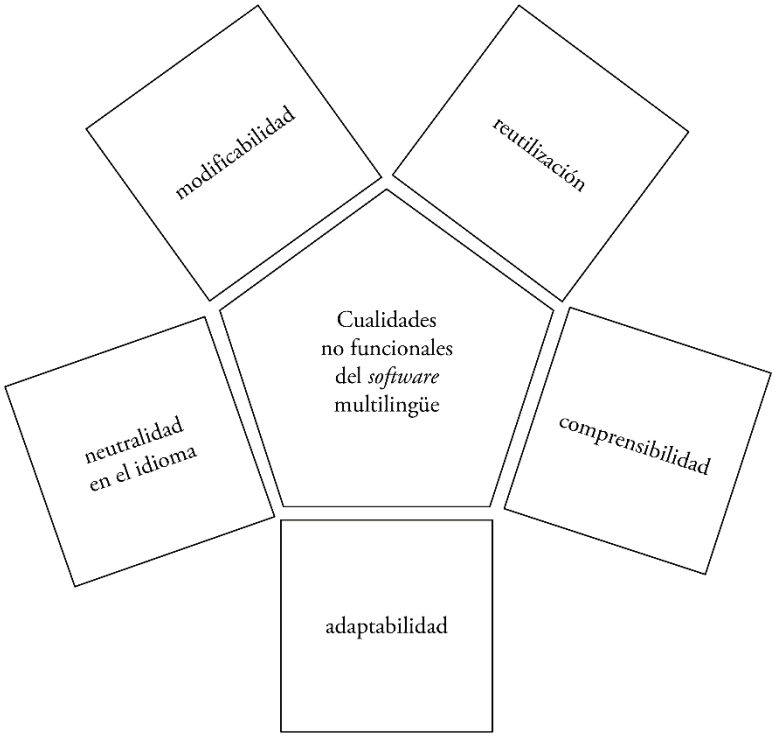

Figura 2.4 - Cualidades no funcionales del software *multilingüe*

[139] Las cualidades no funcionales *describen* la manera en que un sistema se comporta en relación con un atributo observable o mensurable. También puede ser un atributo necesario o requerido. No tienen nada que ver con lo que el programa hace. "Software systems are characterised both by their *functionality* (what the system does) and by their *non-functionality* or quality (how the system behaves with respect to some observable attributes like performance, reusability, reliability, etc.). [...A *n*]*on-functional attribute* (short, *NF-attribute*) [... is] any attribute of software which serves as a mean to describe it and possibly to evaluate it. Among the most widely accepted […] we can mention: time and space efficiency, reusability, maintainability, reliability and usability. [...A *n*]*on-functional behaviour* [… is] any assignment of values to the NF-attributes that are in use in a particular software unit (component, connector, port, etc.). [...A *n*]*on-functional requirement* [… is] any constraint referred to a subset of the NF-attributes that are in use in a particular software unit […]" (Franch y Botella 1998, p. 60, énfasis en el original).

[140] El autor usa *maintainability* en lugar de *modifiability*, que es lo que usa anteriormente, para describir las cualidades arquitecturales (véase nota 138). Lo cambiamos por *modificabilidad* para fines de uniformidad.

(*understandability*) nos referimos a aquello que facilita entender y comprender el diseño y la documentación del *software*. *Adaptabilidad* (*adaptability*) es la cualidad por medio de la cual el *software* multilingüe debe funcionar con múltiples idiomas de forma concurrente según lo seleccione el usuario. La selección del idioma puede ser estática (cuando se compila) o dinámica (cuando se ejecuta). Bajo condiciones normales, los componentes lingüísticos se encuentran íntimamente ligados a los *componentes del dominio*[141] (*domain components*). La *neutralidad en el idioma* (*language neutrality*) es una cualidad que ayudará al desarrollador a crear aplicaciones no dependientes del idioma en el nivel del dominio de la aplicación. Cuando se pretende un acercamiento arquitectural para desarrollar *software* multilingüe es de suma importancia tener en cuenta todas estas cualidades (*Ibidem*, pp. 36-38).

Luego el autor presenta dos propuestas (a las cuales se refiere como hipótesis) sobre cómo pretende abordar su investigación. Primero, se propone analizar los modelos actuales de *software* multilingüe basándose en las cualidades antes mencionadas:

> The design rationale behind the existing multilingual software has to be mined and formally modeled [… using] multilingual concerns. An analysis of the mined models has to be carried out based on the multilingual software qualities.
>
> (*Ibidem*, p. 39)

Seguido, se plantea buscar cuáles son los *aspects*[142] que tienen relación directa con el idioma a fin de crear un modelo de referencia que refleje las cualidades esperadas para *software* multilingüe:

---

[141] Véase nota 128.

[142] No es aspecto en el sentido que se puede encontrar en el diccionario (rasgo, característica, cualidad), sino como se describe a continuación: "An aspect is a concern that cross-cuts the primary modularization of a software system. An aspect-oriented programming language extends traditional programming languages with constructs for programming aspects. Such constructs can localize the implementation of crosscutting concerns in a small number of special program modules, rather than spreading the implementation of such concerns throughout the primary program modules" (Kiczales, Gregor J. et al. 2002). "[…] El término *crosscutting* normalmente se describe en términos de los conceptos *scattering* y *tangling*. Por ejemplo, un *crosscutting* concern

> Based on the analysis of mined models, separation of the language aspects [… will] be carried out and […] used to [… propose] a new model for multilingual software development. This proposed model [… will] be refined into a reference model which exhibits the expected quality requirements.
>
> (*Ibidem*, p. 40)

Con este modelo de referencia, el autor aspira a proporcionar a los arquitectos de *software* una base arquitectural sobre la cual construir sistemas de *software* multilingües que brinden las cualidades esperadas para este tipo de *software*.

Antes de proseguir con la reseña de la investigación de Venkatesan creemos pertinente detenernos un poco para señalar cuán complicadas resultan las explicaciones que da el autor sobre los modelos existentes y el modelo que propone más adelante. Hemos observado que su acercamiento investigativo se rige por el concepto de abstracción que promulga la arquitectura de *software* y que busca hacer generalizaciones que le permitan analizar y contrastar las características que muestran los modelos estudiados en abstracto. Como indica en su primera "propuesta", Venkatesan investiga los modelos que se usan para desarrollar *software* multilingüe y abstrae de esta investigación las cualidades no funcionales de *software* multilingüe que dichos modelos exhiben. Al extraer esta información generaliza los conceptos usando lenguaje matemático para explicar cómo cada parte se relaciona con las demás. El autor no da ejemplos concretos de nada de lo que habla. Tampoco es explícito cuando habla de los componentes ni de las funciones, qué los conforman, qué función tienen, qué aspectos del idioma albergan. Los recursos gráficos se limitan a conjuntos matemáticos y a cajones unidos por rayas con poco menos que la información más necesaria. En nuestras ilustraciones hemos hecho un intento de ser más explícitos, pero nos limita grandemente lo abstracto y árido del trabajo investigativo de Venkatesan. Así, pues, intentamos abordar de la mejor manera su propuesta y de presentar al lector los elementos que pudieran permitir al localizador entenderse de la mejor manera con un arquitecto de *software* y entender el modelo multilingüe de arquitectura que propone.

---

puede ser definido como un *concern* que se encuentra disperso (*scattering* [sic]) y enmara[ña]do (*tangling* [sic]) con otros *concerns* debido a una mala modularización […]" (Conejero, Berg y Hernández 2007, p. 224, bastardillas nuestras). Toda vez que se refiera a aspecto en este sentido lo dejaremos en inglés y usaremos bastardillas.

El capítulo cuatro Venkatesan lo dedica a clasificar, describir y analizar los modelos de *software* multilingüe que se usan en la actualidad. Tomando como base el *software* multilingüe existente, los describe, los modela de forma abstracta en cuanto a estructura, y analiza los puntos a favor y en contra que refleja cada uno.

El primer modelo que presenta es el "modelo de *wrapper*". El *wrapper* es en esencia una capa que encapsula la aplicación monolingüe e intercepta su sistema de I/O. Funciona inyectando cadenas texto traducidas a la interfaz que percibe el usuario. La Figura 2.5 nos muestra este modelo. En ella podemos apreciar la función del interceptor, que en efecto funge como mediador entre la interfaz interna (la no localizada) y la interfaz externa (que da la impresión al usuario de estar localizada). El modelo de *wrapper* es específico a cada aplicación

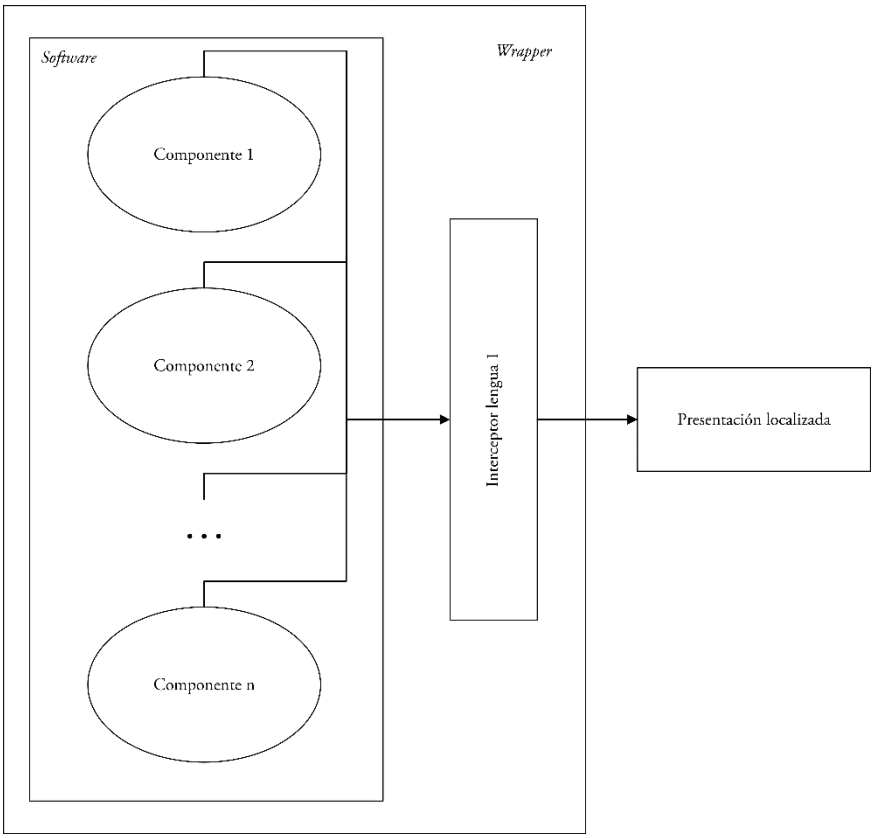

Figura 2.5 - Modelo de wrapper

"multilingüe"; es decir, se crea un *wrapper* que solo funciona con la aplicación que se quiere localizar. El punto a favor que menciona el autor es que es un modelo simple y rápido de aplicar, aunque primitivo. Como punto en contra, debido a que este modelo se limita al I/O de la aplicación, solo la interfaz aparece traducida; es decir, no es posible modificar la lógica

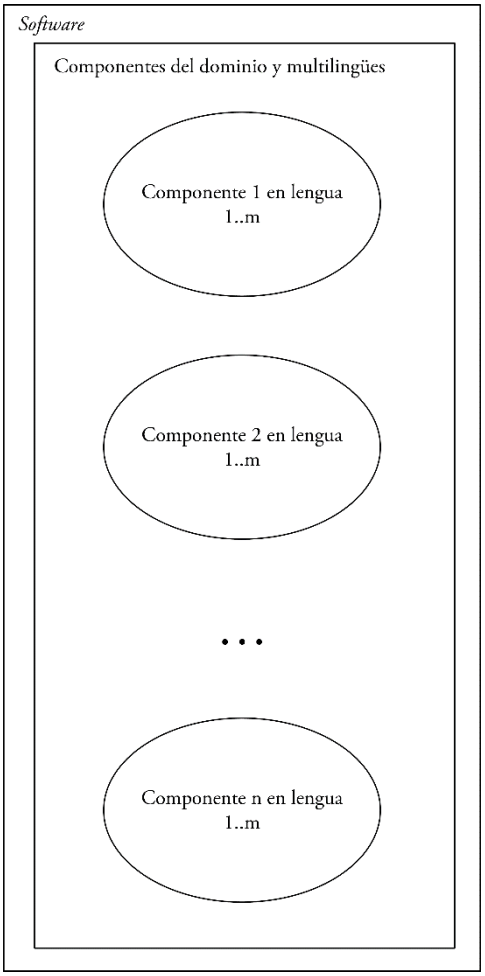

Figura 2.6 - Modelo monolítico (Venkatesan 2008, p. 47)

interna del programa adaptado al *locale*, pues este permanece encapsulado dentro del *wrapper*. Además, cada vez que se actualiza la aplicación hay que actualizar el *wrapper*, pues el programa está encapsulado adentro: el *wrapper* está programado específicamente para cada pantalla que presenta la aplicación, por lo tanto, si se cambia la presentación en la interfaz, hay que cambiar el *wrapper*. Por su naturaleza, este modelo carece de las cualidades no funcionales que muestra el *software* multilingüe[143] (*Ibidem*, pp. 45-47).

El segundo modelo es el llamado "modelo monolítico", en el cual se colocan las funcionalidades lingüísticas junto con las del dominio en un solo componente. Este es el modelo que se usaba para localizar en los inicios de la localización, donde el localizador entraba en el programa a hacer cambios en las cadenas de texto y se creaba un programa para cada versión localizada. La Figura 2.6 muestra cómo se le añade funcionalidad lingüística a cada componente, que tiene un total de "n" componentes y funciona en "m" idiomas. Aquí es difícil modificar cualesquiera de las funcionalidades porque todo está enmarañado[144], además, no sigue las convenciones de diseño modular ni las orientadas a objetos, con lo cual se hace difícil obtener alguna de las cualidades no funcionales que requiere el *software* multilingüe. Como punto a favor indica que cumple con los requisitos

[143] Las cualidades no funcionales del *software* multilingüe se enumeran a partir de la página 101.

[144] *Tangled*, véase nota 142.

funcionales y lingüísticos de las partes implicadas, pero Venkatesan hace una importante salvedad: "si funciona". La dificultad para entender la implementación hace que se haga difícil modificarla o reusar el contenido (*Ibidem*, pp. 47-48).

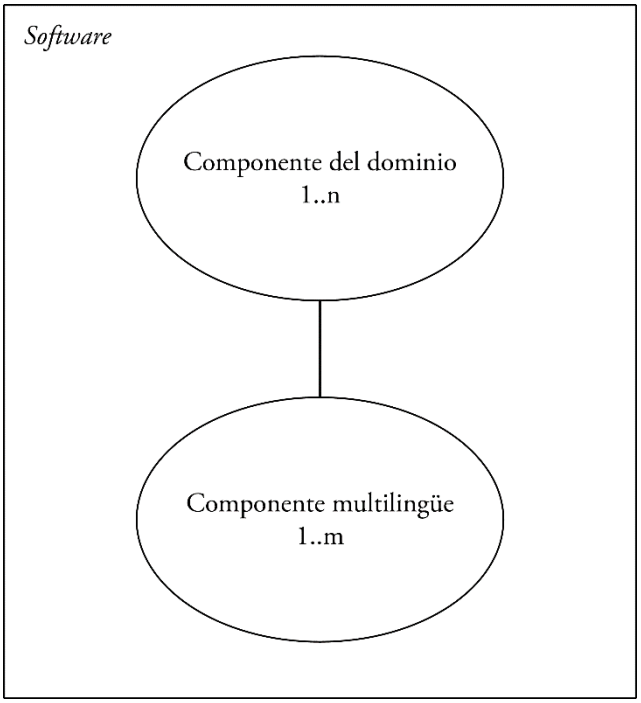

Figura 2.7 - Modelo de biblioteca multilingüe (Venkatesan 2008, p. 48)

El tercer modelo que el autor describe se llama "modelo de biblioteca multilingüe", en donde hay una clara separación entre las funcionalidades del dominio y las funcionalidades multilingües, con lo cual los programadores pueden enfocar adecuadamente el diseño y programación de los componentes multilingües aparte de los componentes del dominio. Sin embargo, Venkatesan indica que la biblioteca multilingüe es difícil de diseñar pues cada idioma tiene sus particularidades[145], lo cual aumenta la complejidad en el desarrollo y manejo de varios idiomas. A favor en este modelo, el autor destaca la separación entre los componentes del dominio y los componentes multilingües, lo que incrementa hasta cierto punto las cualidades de reutilización y modificabilidad en general. Aun así, los componentes multilingües siguen siendo monolíticos (porque están contenidos todos en una sola biblioteca, véase Figura 2.7) y por ello reducen la modificabilidad en este sentido. El autor clasifica esta característica del modelo como negativa (*Ibidem*, pp. 48-49).

[145] La dificultad radica en que la interacción "componente del dominio X" con "componente del multilingüe A" cambia en función del idioma que está mediando. Esta relación no va a ser igual si el idioma que media es, por poner un ejemplo, un idioma que se escribe LTR o si media un idioma BIDI (véase nota 102). El idioma LTR no tiene las mismas particularidades programáticas que un idioma BIDI, de ahí la dificultad de programar estas bibliotecas multilingües.

El último modelo que el autor describe se conoce como el "modelo de biblioteca de idioma", en el cual existe una biblioteca para cada uno de los idiomas disponibles en el *software*, y cada una de estas bibliotecas aporta las funcionalidades multilingües requeridas por el componente del dominio que le corresponde (véase Figura 2.8). Esta separación ayuda a que durante el análisis y diseño del *software* multilingüe se pueda prestar atención a cuestiones relacionadas con el idioma y a que se puedan planificar y ejecutar adecuadamente el diseño y desarrollo del *software*. La fortaleza principal de este modelo, según el autor, es que hace hincapié durante el análisis y desarrollo en las particularidades de cada idioma. La debilidad de

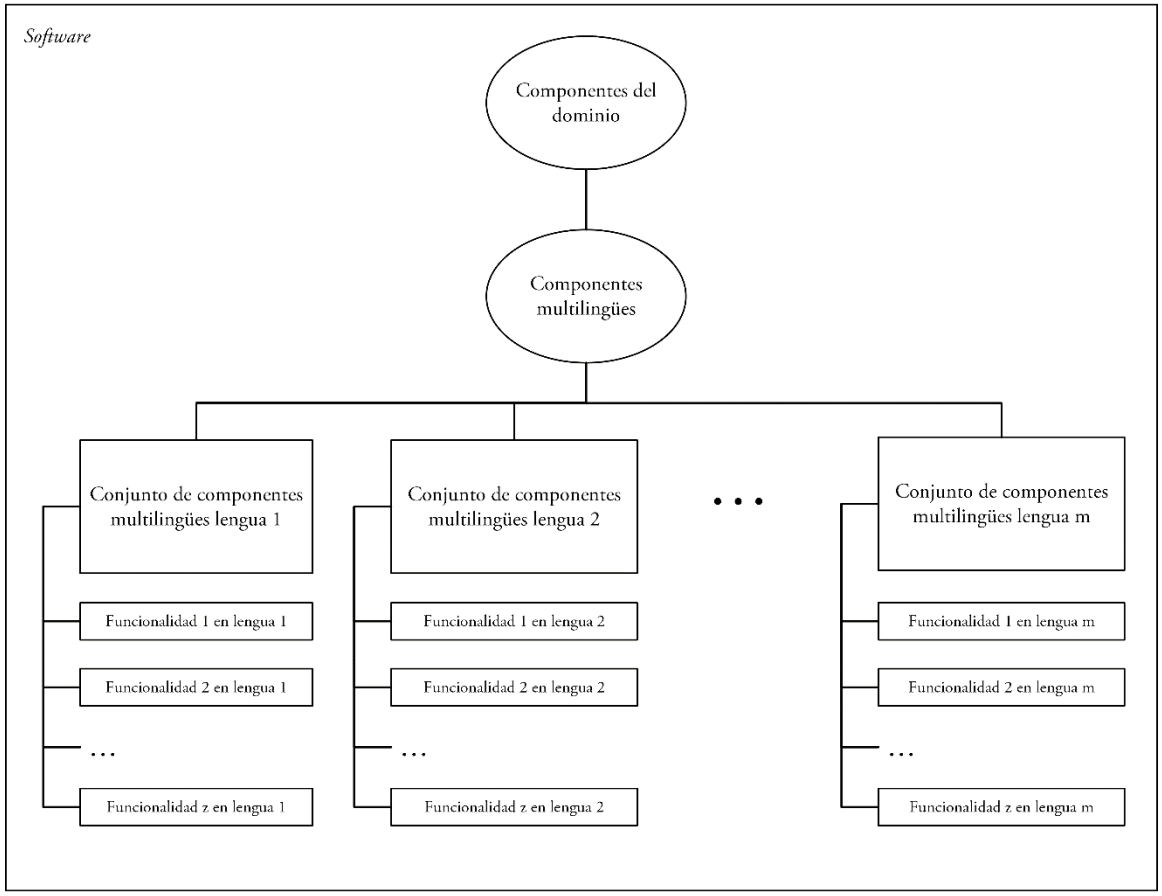

Figura 2.8 - Modelo de biblioteca de idioma
(Venkatesan 2008, p. 50)

este modelo es que no toma en consideración los *aspects* (véase nota 142) del idioma, como lo son las cuestiones lingüísticas, de la implementación y del dominio, lo cual ocasionará frecuentes modificaciones de los componentes (*Ibidem*, pp. 49-51).

El autor resume las cualidades no funcionales del *software* multilingüe en una tabla comparativa (véase Tabla 2.2) que ilustra claramente cómo cada modelo cumple o no con estas cualidades, donde + + significa que cumple a cabalidad con la cualidad, + – significa que cumple parcialmente con la cualidad y – – significa que no cumple con la cualidad (*Ibidem*, p. 51). Así llega a la conclusión de que, para propósitos prácticos, ninguno de los modelos existentes cumple a cabalidad con todas las cualidades no funcionales que necesita tener un *software* multilingüe. El *wrapper*, como se crea para una aplicación específica, demuestra la cualidad de *comprensibilidad*, pero carece de las demás cualidades porque funciona solo sobre un diseño específico. En el modelo monolítico las funcionalidades del dominio y las multilingües no están separadas, con lo cual no exhibe ninguna de las cualidades. El de biblioteca bilingüe separa las funcionalidades multilingües de las del dominio y así facilita la *comprensibilidad* y la *modificabilidad* del diseño, pero este monolito lingüístico dificulta la presencia de las demás cualidades.

Cualidades/ modelo analizado	Modificabilidad	Comprensibilidad	Reutilización	Adaptabilidad	Neutralidad en el idioma
Wrapper	– –	+ +	– –	– –	– –
Monolítico	– –	– –	– –	– –	– –
Biblioteca multilingüe	+ –	+ –	– –	– –	– –
Biblioteca de idioma	+ –	+ –	+ –	+ –	– –

Tabla 2.2 - Comparación de los modelos analizados en cuanto a las cualidades no funcionales de software multilingüe (Venkatesan 2008, p. 51)

El que cumple con la mayor cantidad de cualidades no funcionales, aunque todas de manera *parcial*, es el de biblioteca de idioma porque simplifica el añadir y modificar idiomas (cualidad de *modificabilidad*) al reunir las funcionalidades individuales del idioma dentro de componentes multilingües. Esta agrupación aporta la cualidad de *comprensibilidad*. La

reutilización es posible a nivel de idioma, pero está restringida a nivel de *aspect*, donde también carece de *adaptabilidad*. El hecho de que los *aspects* del idioma están codificados dentro de las funcionalidades le resta *neutralidad en el idioma* (*Ibidem*, pp. 51-53).

2.3.3.2 Enfoque de espacios de diseño

Puesto que los modelos que el autor ha estudiado no satisfacen del todo las cualidades de *software* multilingüe que se definieron anteriormente, Venkatesan propone en el capítulo cinco un nuevo modelo que satisfaga todas las cualidades. Ya en el capítulo tres el autor nos adelantaba que usaría un enfoque de espacios de diseño (*design space approach*)[146] pues este lo ayudaría a lidiar con la complejidad durante el proceso de diseño (*Ibidem*, p. 55). Un espacio de diseño identifica las dimensiones funcionales y las dimensiones estructurales clave que se usan para crear un diseño de sistema dentro de un dominio de aplicación específico. Las dimensiones funcionales[147] describen cuestiones que se refieren al espacio-problema (*problem space*) en términos de alternativas de diseño relacionadas con la funcionalidad del programa o con su desempeño; las dimensiones estructurales[148] describen las cuestiones que se refieren al espacio-solución (*solution space*) en términos de un conjunto de características de implementación (*Ibidem*, p. 56).

[146] Para una explicación más clara sobre los espacios de diseño en la arquitectura de *software* véase Lane (1990). El autor prácticamente copia *verbatim* partes de este artículo en esta sección de su tesis.

[147] Las dimensiones funcionales identifican los *requisitos* para un sistema de IU que más afectan su estructura. Se agrupan en requisitos externos (requisitos para programas en particular, para usuarios y de aparatos de I/O a los que se les dará apoyo; y las limitaciones que imponen los sistemas computadorizados del entorno), comportamiento de la interacción básico (decisiones clave sobre el comportamiento de la IU que influyen la estructura interna) y consideraciones prácticas (costos de desarrollo y el grado de adaptabilidad del sistema) (*Ibidem*, p. 6).

[148] Las dimensiones estructurales tienen que ver con decisiones que determinan la *estructura general* de un sistema de IU. Se agrupan en división de funciones y conocimiento intermodular (cómo las funciones se dividen en módulos, las interfaces entre módulos y la información que contiene cada módulo), cuestiones relacionadas con la representación de datos (los datos en sí, cómo se comunican por medio de interfaces y los metadatos que afectan el despliegue y el comportamiento de la IU) y cuestiones relacionadas con el flujo de control, comunicación y sincronización (el comportamiento dinámico de la IU) (*Ibidem*, pp. 9-10).

El espacio de diseño es multidimensional: cada dimensión describe una variación de una característica o requisito del sistema. Los valores que acepta cada dimensión corresponden a las distintas alternativas de diseño. Cuando existe una relación entre dimensiones se crean *reglas* que se usan para encapsular el conocimiento adquirido sobre el diseño. Hay reglas que enlazan dimensiones funcionales con dimensiones estructurales y que por lo tanto permiten que los requisitos del sistema dirijan el diseño estructural. La otra categoría, las reglas que entrelazan dimensiones estructurales, permite que haya uniformidad interna en el diseño (*Idem*).

2.3.3.2.1 Espacio de diseño del software

Usando un espacio de diseño, el autor analiza los modelos multilingües e intenta describirlos a fin de establecer la base sobre la cual crear su *framework* y su modelo de referencia arquitectural. El espacio de diseño del *software* D lo describe usando lenguaje matemático, matrices y funciones. Para empezar, un *software* tendrá una serie de requisitos

$$R = \{r_{1,}, r_2, r_3, \cdots, r_a\}$$

donde *a* corresponde a la cantidad de requisitos de dicho *software*. El *software* multilingüe tendrá que poder manejar una serie de idiomas

$$L = \{l_{1,}, l_2, l_3, \cdots, l_d\}$$

donde *d* corresponde a la cantidad de idiomas que maneja dicho *software* multilingüe. Para cada requisito *R* de cada idioma *L* habrá un componente C_{RL} que atienda dicho requisito. Entonces el conjunto de componentes *C* se puede expresar con la siguiente matriz:

$$C = \begin{bmatrix} C_{r1l1} & C_{r2l1} & C_{r3l1} & \cdots & C_{ral1} \\ C_{r1l2} & C_{r2l2} & C_{r3l2} & \cdots & C_{ral2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{r1ld} & C_{r2ld} & C_{r3ld} & \cdots & C_{rald} \end{bmatrix}$$

Como consecuencia de esta relación requisito-idioma, según se añadan requisitos o idiomas la matriz C crecerá pues el conjunto de componentes aumentará. La cualidad no funcional de *modificabilidad* se verá afectada cuando haga falta modificar un requisito. Si, por ejemplo, el requisito r_x cambia pues habrá que modificar todo componente C_{rxL} para cada idioma $L = \{l_{1,}, l_2, l_3, \cdots, l_d\}$ que se vea afectado. Lo mismo pasa si cambia algún *concern* del idioma l_x: hay que modificar cada componente C_{Rlx} para cada requisito. (*Ibidem*, pp. 58-60). Así pues,

dice el autor, el diseño de un componente de idioma lo rigen los *concerns* del idioma y, aunque esta conclusión tiene sentido, Venkatesan no explica cómo llega a ella. Estos *concerns*, a los que también se les llama *aspects* del idioma, son un tipo de *crosscutting concern*[149] y se dividen en tres tipos (*Ibidem*, p. 60):

- Cuestiones lingüísticas (*linguistic issues*) - son los *aspects del idioma a nivel lingüístico*, es decir, el conjunto de caracteres escritos y fonéticos, las reglas sintácticas y semánticas, etc. Los cambios en las gramáticas u ordenaciones por parte del Gobierno pueden suscitar alteraciones que requieran modificación (*Ibidem*, p. 61).
- Cuestiones relacionadas con la implementación del idioma (*language implementation issues*) - son los *aspects del idioma a nivel de la implementación*, por ejemplo, los esquemas de codificación[150], la disposición del teclado, los estándares de despliegue en pantallas, los estándares de *output*, entre otros. Se han comenzado a estandarizar estas cuestiones (por ejemplo, la creación de Unicode y sus prestaciones, el repositorio de datos de *locale* comunes o CLDR[151]), pero hasta que existan estándares universales se implementarán de distinta manera añadiéndole complejidad al proceso de diseño (*Idem*).
- Cuestiones relacionadas con el idioma a nivel de dominio (*domain level language issues*) - corresponden a los *aspects del idioma a nivel del dominio del software* (no del dominio del idioma), como lo son el reconocimiento de voz basado en el dominio, *parsing* o análisis sintáctico basado en el dominio, e indexación basada en el dominio (*Ibidem*, pp. 61-62).

[149] Véase nota 142.

[150] Los esquemas de codificación se explican más adelante en el apartado "Codificación de caracteres, Unicode". Véase nota 197.

[151] Véase el apartado "Una fuente común de datos relacionados con la localización" en la página 173.

2.3.3.3 Modelo de biblioteca de idiomas basada en *aspects*

Venkatesan propone el *modelo de biblioteca de idiomas basada en aspects* que construye a partir de operaciones de adición, eliminación y modificación sobre los componentes, elementos y conjuntos derivados anteriormente descritos usando, como mencionáramos antes, estructuras algebraicas complejas. De esta manera crea los espacios de *aspects* del idioma a nivel lingüístico, de implementación y del dominio, y a partir de ellos deriva el diseño que se muestra en la Figura 2.9 y el cual usa como punto de partida para crear un modelo de referencia arquitectural para *software* multilingüe (*Ibidem*, pp. 62-71).

2.3.3.4 Modelo de referencia arquitectural para *software* multilingüe

Los modelos abstractos o modelos de referencia representan los componentes centrales

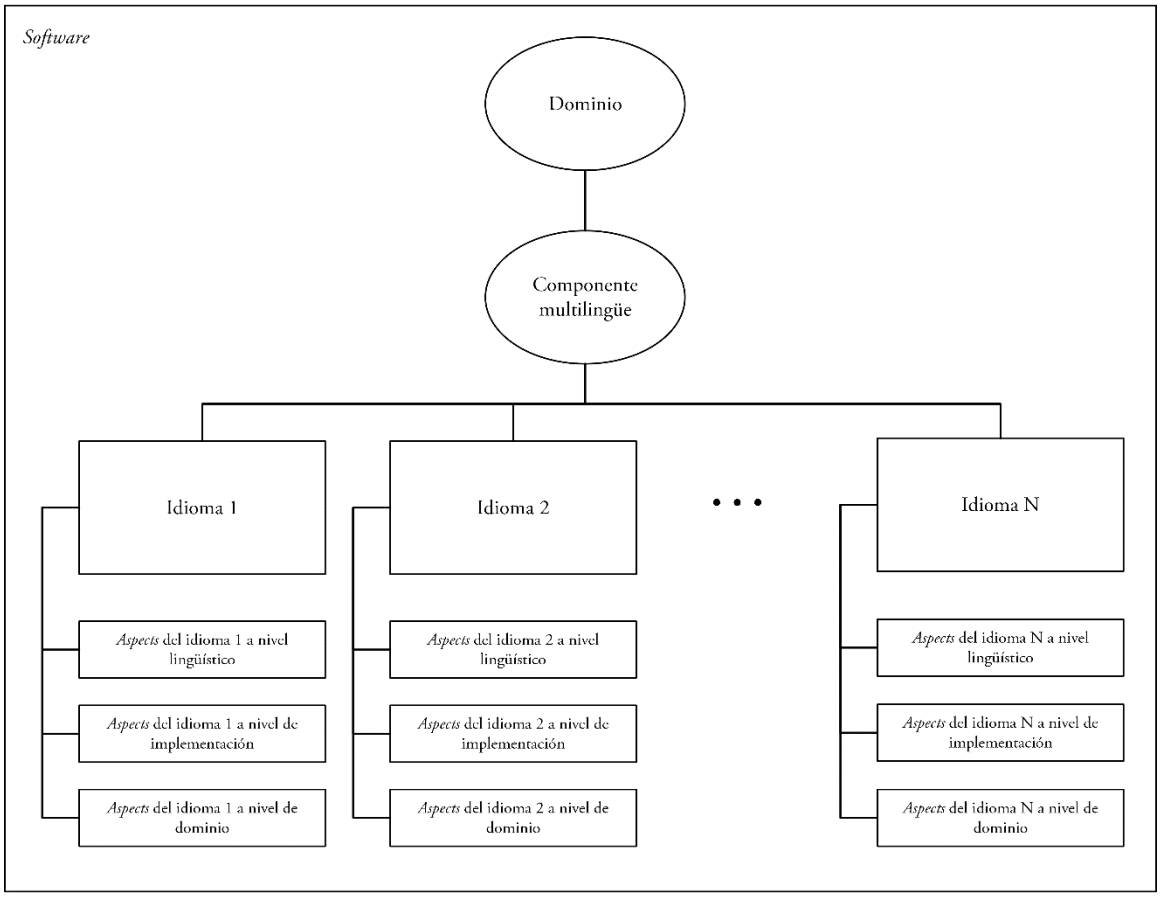

Figura 2.9 - Modelo de biblioteca de idiomas basada en aspects
(Venkatesan 2008, p. 71)

(nucleares, *core*) que se supone que tenga el *software* sin entrar en los detalles de cómo

exactamente se van a implementar. Es una representación abstracta del *software* que muestra las entidades involucradas y cómo se interrelacionan; es una plantilla sobre la cual se construye la arquitectura de *software*. Este modelo de referencia también ayuda a explicarles a las partes implicadas el funcionamiento del *software* antes de que este sea desarrollado. Ya que no existe un modelo de referencia para *software* multilingüe, Venkatesan propone crear uno basándose en el *modelo de biblioteca de idiomas basada en aspects* desarrollado en el capítulo anterior de su tesis doctoral. Para crearlo, se le aplican operaciones unitarias que veremos a continuación (*Ibidem*, p. 75).

En la arquitectura de *software* se aplican comúnmente operaciones unitarias con el fin de crear abstracciones más lejanas a la implementación, es decir, de más alto nivel o generalizado. Estas operaciones unitarias son[152]:

2.3.3.4.1 Separación

Es una de las funciones más primitivas del arquitecto de *software* y se usa para segregar funcionalidades y dividirlas en componentes claramente distinguibles que tienen una interfaz bien definida. La separación también se puede usar para asegurarse de que los cambios en el entorno no afecten al componente y viceversa, siempre y cuando la interfaz se mantenga intacta. La separación aporta las calidades de *modificabilidad* y *portabilidad*. Se manifiesta en dos subtipos: la *descomposición uniforme* y la *replicación*. La descomposición uniforme separa un componente grande de un sistema en dos o más componentes más pequeños. Esto facilita

[152] Si miramos la referencia que indica el autor, parece ser que esta sección completa la toma de Kazman y Bass (1994). Las explicaciones sobre las operaciones unitarias que ofrecen Kazman y Reddy (1996, pp. 2-4) son muchísimo más claras que la información que ofrece Venkatesan. Estos autores son quienes, a partir de sus observaciones y entrevistas con expertos en diseño, comienzan a hablar de operaciones unitarias en el ámbito de la arquitectura de *software*, término que toman prestado de la ingeniería química (*Ibidem*, p. 1). Luego de definir la operación de separación, por ejemplo, explican que esta "[…] is the super-type of all other unit operations. […]", algo que no queda claro en la explicación de Venkatesan, porque él deja la descomposición uniforme y la replicación como únicos subtipos de la separación. En realidad, todas las demás operaciones son acciones opuestas a, o subtipos de la separación.

la integración de los componentes y la escalabilidad del sistema (*system scaling*). Los mecanismos de la descomposición uniforme son *part-whole*[153] e *is-a*[154] (*Ibidem*, p. 76). Cuando se duplica un componente dentro de la arquitectura se está usando la replicación, que permite mejorar la fiabilidad (por medio de la redundancia) y el rendimiento del sistema (porque permite dividir la carga de proceso en varios componentes del sistema) (*Ibidem*, p. 77).

2.3.3.4.2 Abstracción

Es una operación donde se crea una máquina virtual que funciona como una caja negra que esconde la implementación interna del componente. Son complicadas de crear, pero una vez creadas, otros componentes la pueden usar. Esto aumenta la *reutilización*, y simplifica la creación y mantenimiento de los componentes del *software* (*Idem*).

2.3.3.4.3 Compresión

Función opuesta a la separación, donde se juntan capas o interfaces que separan funciones del sistema. Se usa para mejorar la ejecutoria del sistema, para eliminar capas cuando no se necesitan sus servicios y para acelerar el desarrollo del sistema (*Idem*).

[153] La "parte" es cada parte que conforma el conjunto de los subcomponentes que representa una porción de la funcionalidad total del sistema que ha sido descompuesto. El "todo" lo constituye cada una de estas partes y solo puede ser creado a partir de estos subcomponentes o partes. "[… E]ach of a restricted set of subcomponents represents a portion of the functionality, and every component in the system can only be built from these subcomponents […]" (Kazman y Reddy 1996, p. 4; Venkatesan 2008, p. 76)

[154] La funcionalidad de cada subcomponente que conforma el componente que ha sido descompuesto usando la operación de "is-a" es una especialización con respecto a la funcionalidad del componente "padre". "Each of the subcomponents represents a specialization of its parent's functionality" (Kazman y Reddy 1996, p. 4; Venkatesan 2008, p. 76)

2.3.3.4.4 Compartir recursos

Se usa para encapsular datos o servicios a fin de compartirlos con varios consumidores[155] independientes. Son difíciles y caros de desarrollar[156], pero aportan *portabilidad*, *integrabilidad* y *modificabilidad*, principalmente porque reducen el acoplamiento[157] (*Idem*).

2.3.3.5 Aplicación de las operaciones unitarias al *modelo de biblioteca de idiomas basada en aspects*

Veamos las operaciones unitarias que se le aplican al *modelo de biblioteca de idiomas basada en aspects* que se muestra en la Figura 2.9 de la página 115 para crear el modelo de referencia. La separación existente entre el dominio de la aplicación y el componente multilingüe se justifica por *descomposición uniforme*. A fin de proporcionar neutralidad en el idioma, se aplica la operación de *abstracción* al componente multilingüe para dividirlo en dos

[155] Un consumidor es cualquier componente, usuario, programa, etc. que necesite usar un recurso.

[156] Es típico que el recurso o los recursos compartidos sean gestionados por un gestor de recursos, que es el único que tiene acceso a los recursos en cuestión (Kazman y Reddy 1996, p. 4) y brinda el acceso de acuerdo con las necesidades que se programen. Por ser una puerta única de acceso a los recursos que controla, está encargado de verificar la disponibilidad de los recursos, asignar y desasignar la utilización, coordinar acciones, etc., y esto es lo que lo hace difícil y costoso de construir.

[157] El acoplamiento es la "fortaleza entre las relaciones de los módulos" (Ruiz y González Harbour 2009, p. 7) Como veremos, este concepto está íntimamente relacionado con la operación de separación que se aborda en la página 113. "Separation […] involves breaking a complicated problem into two or more distinct portions, solving each portion and then unifying the solution. The success of this technique depends on the difficulty of the unification. If the concerns addressed by each element of the decomposed problem are truly different, then the unification is simple. If not, a large amount of communication between the distinct portions is required to solve the total problem, and the separation technique may have been incorrect. The term *coupling* is used to describe the amount of communication and cooperation necessary to unify the two separated portions. Low coupling indicates that the separation is achieving the goal of making the total problem easier to solve. High coupling indicates that the separation is not achieving that goal" (Bass, John y Kates 2001, p. 42).

partes: los componentes multilingües abstractos y los componentes multilingües concretos. Así se desacoplan las interfaces del idioma (que aparecerían en el componente abstracto) respecto a su implementación. La interacción entre el dominio de la aplicación con los componentes

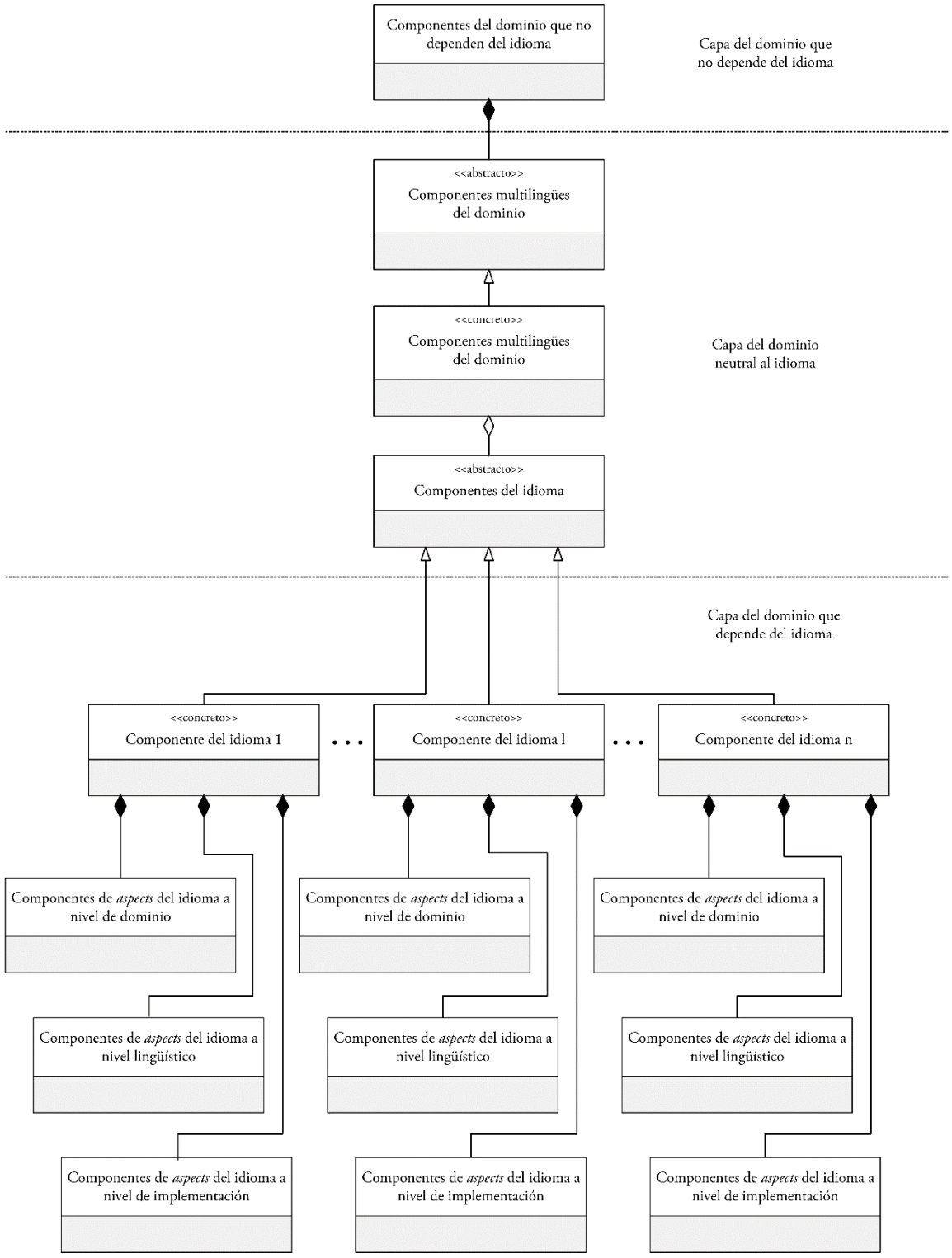

Figura 2.10 - Modelo de referencia arquitectural para software multilingüe *(ARMMS)*
(Venkatesan 2008, p. 85) El rombo negro indica composición, el rombo blanco indica agregación y la flecha blanca indica herencia.

multilingües y su desarrollo están restringidos entonces a los componentes multilingües abstractos. Esto le proporciona *neutralidad en el idioma*.

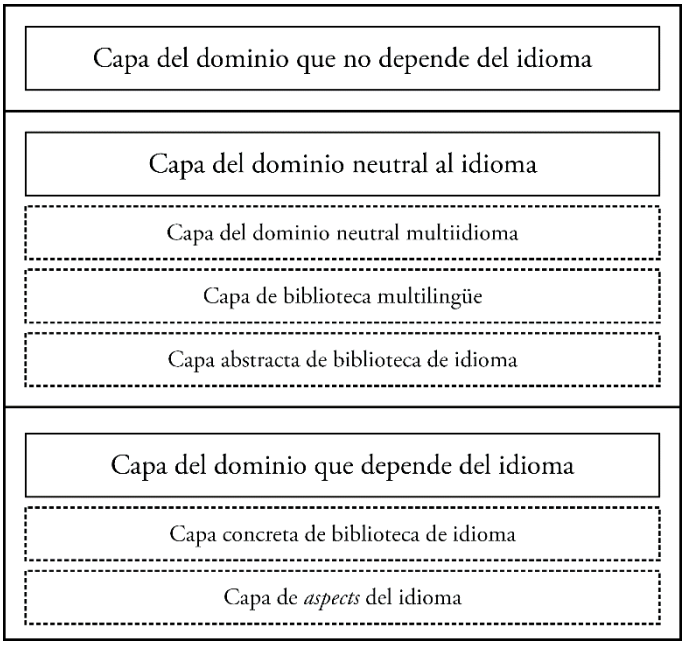

Figura 2.11 - Diagrama de capas de ARMMS (Venkatesan 2008, p. 81)

Al igual que la separación existente entre el dominio de la aplicación y el componente multilingüe se justifica y conserva por descomposición uniforme, conservamos y justificamos la separación entre el componente multilingüe y los componentes del idioma. Sin embargo, es necesario desacoplar esta relación para crear una capa neutra entre el nivel superior y los componentes de cada idioma, con lo cual volvemos a aplicar abstracción y dividimos los componentes concretos de cada idioma, que se abstraen en los componentes abstractos del idioma. La descomposición uniforme se aplica a la capa de los *aspects* por las mismas razones que se aplica y se conserva en los componentes anteriores.

Luego de aplicar estas operaciones obtenemos el *modelo de referencia arquitectural para software multilingüe* o ARMMS, siglas de *Architectural Reference Model for Multilingual Software*, que se ilustra en la Figura 2.10. Si lo comparamos con el *modelo de biblioteca de idiomas basada en aspects* (Figura 2.9) podemos apreciar los cambios que sufre el modelo al convertirlo en un modelo de referencia. Derivamos la representación por capas del modelo (Figura 2.11) a partir de ARMMS (*Ibidem*, pp. 78-80).

2.3.3.5.1 Capa del dominio que no depende del idioma

Esta capa separa los componentes (o los *concerns*) del dominio de la aplicación frente a todo lo que tenga que ver con cosas del idioma a fin de que los diseñadores puedan concentrarse en las funcionalidades del dominio y despreocuparse por las limitaciones (*constraints*) que

imponen los idiomas. Incluye todos los componentes del dominio de la aplicación que no dependen del idioma (*Ibidem*, p. 80).

2.3.3.5.2 Capa del dominio neutral al idioma

Con esta capa se obtiene un acoplamiento ligero entre la capa superior y la inferior (la que no depende y la que sí depende del idioma); la capa funciona como una separación entre estas. Está compuesta por los componentes multilingües del dominio y los componentes abstractos del idioma (*Ibidem*, p. 81).

2.3.3.5.2.1 Capa del dominio neutral multiidioma

Esta capa contiene componentes genéricos de características multilingües a fin de evitar dependencias específicas al idioma de los componentes multilingües del dominio (*Idem*).

2.3.3.5.2.2 Capa de biblioteca multilingüe

Esta capa brinda mecanismos que permiten la coexistencia de múltiples idiomas y que se pueda cambiar de idioma de forma dinámica. Contiene componentes multilingües que permiten operaciones de escritura e interfaces parametrizadas. Aporta a la calidad de neutralidad en el idioma (*Ibidem*, p. 82).

2.3.3.5.2.3 Capa abstracta de biblioteca de idioma

Esta capa se crea por medio de la agregación de las bibliotecas de idiomas para conformar una biblioteca multilingüe abstracta. Aporta a la calidad de modificabilidad (*Idem*).

2.3.3.5.3 Capa del dominio que depende del idioma

En esta capa se ejecuta la implementación de los idiomas y es donde se crean los componentes del idioma por medio de la composición dinámica de los *aspects* del idioma. Aporta a las cinco calidades del *software* multilingüe: modificabilidad, reutilización, comprensibilidad, adaptabilidad y neutralidad en el idioma. Está dividida en dos capas:

2.3.3.5.3.1 Capa concreta de biblioteca de idioma

Aquí se crean componentes del idioma por medio de la composición de *aspects* del idioma que presentan características inherentes al idioma (*Idem*).

2.3.3.5.3.2 Capa de aspects del idioma

Los *aspects* del idioma, divididos en *aspects* lingüísticos, de implementación y de dominio, se tratan aparte en esta capa a fin de aumentar la reutilización y la modificabilidad (*Ibidem*, p. 83).

Este capítulo nos ha presentado la propuesta principal de Venkatesan. En su discusión final el autor explica que debido a la aplicación de las operaciones unitarias antes mencionadas el ARMMS debe mostrar y está diseñado para brindar todas las características no funcionales (modificabilidad, comprensibilidad, reutilización, adaptabilidad y neutralidad en el idioma) que son necesarias al momento de desarrollar programas multilingües. El arquitecto de *software* desempeña un papel esencial como comunicador entre todas las partes implicadas y este modelo le debe servir como herramienta para facilitar esta comunicación. Además, se espera que le ayude a diseñar con facilidad *software* multilingüe. Para los desarrolladores este modelo los debe ayudar durante las etapas de concepción, diseño, desarrollo, pruebas e implementación del *software* en cuestión (*Ibidem*, pp. 86-91).

2.3.3.6 Final y conclusiones

En el último capítulo de su tesis el autor presenta su experiencia al desarrollar *software* usando ARMMS. Explica, mediante el análisis de casos de uso[158] y del dominio, el diseño de cuatro aplicaciones multilingües:

[158] Un diagrama de casos de uso "muestra las relaciones entre los actores y el sujeto (sistema), y los casos de uso" (Anacleto 2005). "Use Case models show both the events the system or business perform, as well as the people, roles, systems, and devices (all three which are referred to as actors) that participate in those events" (Daoust 2012, p. 23). Más adelante explicamos este y otros diagramas. Véase el apartado "Herramientas visuales para la modelación de sistemas" en la página 155.

- PONN Desktop - es un entorno multilingüe que puede funcionar en cuatro idiomas y se ejecuta sobre el sistema operativo del equipo. Provee una GUI, edición de documentos y no documentos multilingües, ejecuta operaciones de ficheros y directorios en varios idiomas, tiene un explorador de ficheros y directorios multilingüe y provee ayuda sobre sus funcionalidades.
- KURAL - es un idioma de programación multilingüe simple que funciona sobre PONN.
- MAYAN - es un *framework* para crear y construir de manera dinámica páginas web multilingües.
- PONN SMS - es un entorno multilingüe para enviar mensajes SMS en teléfonos móviles.

Esta experiencia lo ayuda no solo a probar su propuesta, sino también a refinar el modelo de forma iterativa (*Ibidem*, pp. 92-147).

Venkatesan resume los hallazgos principales de su investigación: descripción y crítica de los modelos multilingües existentes, diseño de un modelo basado en *aspects* para compensar las inadecuaciones de estos modelos existentes, desarrollo de ARMMS al aplicar operaciones unitarias al modelo basado en *aspects* y la aplicación de este modelo de referencia en la creación de *software* multilingüe (*Ibidem*, pp. 148-150).

2.3.3.7 Propuesta de vanguardia, pero sumamente compleja

El modelo de referencia que presenta Venkatesan resulta demasiado complejo y abstracto para que una persona que no sea arquitecto de *software* lo pueda entender. De hecho, pensamos que es muy posible que el modelo por sí solo sea incomprensible incluso para un arquitecto de *software* experimentado si este se acerca a él sin conocer con antelación los retos que representa la creación de programas multilingües. Con esto queremos decir que para entender ARMMS es necesario leer con detenimiento su tesis doctoral de principio a fin y profundizar en los conceptos que propone, tarea no muy fácil para un arquitecto que no tiene idea de las implicaciones lingüísticas que Venkatesan discute y, para todos los efectos, imposible para el resto de los mortales. El modelo no se puede entender sin antes haber comprendido la

razón y manera en que crea cada una de las capas que conforman ARMMS, qué características inherentes a los idiomas las afecta, qué aspectos (no me refiero a los *aspects*) relacionados con la internacionalización contienen y cómo se aplica este modelo al diseño de sistemas multilingües.

Sin embargo, a juzgar por la experiencia arquitectural que Venkatesan presenta en el capítulo ocho, es un modelo que podría aplicarse con éxito para crear entornos multilingües que permitan una localización adecuada, como es el caso de los programas que creó para que funcionaran sobre el entorno PONN. También hay que entender las aportaciones de los modelos de referencia: "[…] The main contributions of reference models are that they provide an abstraction that standardizes similar *software*, show key entities and relationships, address specific domains, and that they are technology-agnostic […]" (Murtaza y Shwan 2012, p. 59). Al no casarse con ningún tipo de tecnología, significa que se podrían aplicar a cualquier tipo de desarrollo de *software* en cualquier tipo de plataforma, desde microcomputadoras hasta *mainframes*. A pesar de ello, no encontré pruebas de que este modelo haya sido utilizado por otros investigadores o desarrolladores, aparte de la reseña que hacen en su trabajo Murtaza y Shwan, el cual estudiaremos a continuación.

Al terminar de presentar su ARMMS, Venkatesan explica el papel del arquitecto en el proceso de desarrollo de programas multilingües y entre las funciones que le atribuye es la función de mediador.

> [The a]rchitect plays the key role in the software development process. Multilingual software models can be used as communication vehicle by the architect to the stakeholders in order to present an overview about the project. […]
>
> (Venkatesan 2008, p. 90)

Su aliado perfecto puede ser, entonces, un localizador experto en informática que le sirva de apoyo para mediar entre las partes implicadas en la armonización de los requerimientos para crear programas multilingües.

2.3.4 Una colección de directrices prácticas, bien fundamentadas y organizadas: Murtaza y Shwan (2012)

Inicialmente, en su trabajo de fin de máster titulado "Guidelines for Multilingual Software Development" (2012), los autores buscaban maneras para hacer disponibles a todas las personas alrededor del mundo datos e información en línea y por medios electrónicos, pero según sus investigaciones progresaban, el enfoque fue cambiando hasta que terminaron creando una serie de directrices, extraídas de la literatura y del análisis de proyectos desarrollados, en su mayoría, en Suecia. Estas directrices, además de concienciar a los desarrolladores y auspiciadores de proyecto sobre el *software* internacionalizado, disminuirán la complejidad, los costos, la desconfianza y la incertidumbre del proceso de internacionalización y localización (*Ibidem*, pp. 20-21).

La capacidad de un *software* de funcionar en múltiples idiomas es una característica fundamental que mejora la competitividad de una empresa y la satisfacción de sus clientes. Si las pymes prestaran atención a este aspecto de sus productos y servicios, se verían beneficiadas de un enorme potencial de crecimiento, cosa que las grandes empresas de *software* ya han probado[159]. Además, el crecimiento del mercado de los dispositivos móviles acentúa esta necesidad puesto que la creación de *software* para estos equipos se desarrolla para que funcione

[159] "[…] If small and medium software publishers around the world […] start to internationalize their products and services, the potential for growth is great, and many of the larger software publishers have already proved this" (2012, p. 23), afirman los autores. Google y Apple enfatizan por medio de sus prestaciones para internacionalización (véase el apartado "Guías de redacción" en la página 211) y presencia global la importancia de internacionalizar las apps. La App Store abrió en 2008 con el lanzamiento del iPhone 3G y tenía apenas 500 apps. En tres meses ya contaba con 3000 apps y más de 100 millones de descargas, y para finales de 2014 contaba con 1,2 millones de apps y 85 000 millones de descargas. Aunque en con un comienzo más modesto, Google Play, la contraparte Android de la App Store, también comenzó a ofrecer apps en 2008, pero solo ofrecía un puñado. Sin embargo en la actualidad supera a la App Store en apps y descargas: 1,3 millones de apps y 70 % más descargas que la App Store (Sims 2015).

sobre varias plataformas a la vez (iOS, Android, etc.) y tiende a ser multiplataforma[160], si incluye, por ejemplo, una versión web. Introducir nuevos idiomas en las aplicaciones que usan estos equipos se puede convertir en una verdadera pesadilla multilingüe-multiplataforma, si no se considera el multilingüismo durante las etapas iniciales del desarrollo (*Ibidem*, p. 23).

Esta investigación intenta averiguar cuáles son las herramientas, técnicas, acercamientos y tecnologías disponibles para desarrollar *software* multilingüe y para modificar el *software* existente a fin de que brinde el apoyo necesario a culturas e idiomas adicionales y qué prácticas actuales en el desarrollo de *software* inhiben la posibilidad de que un programa se pueda localizar en el futuro. También buscan las razones por las cuales en la actualidad hay tanto *software* y sitios web monolingües que ignoran las necesidades de una gran cantidad de usuarios y de posibles mercados, y cómo esto puede cambiar. Finalmente investigan qué consideraciones y prácticas se pueden introducir en el SDLC[161] para facilitar el desarrollo de *software* multilingüe (*Ibidem*, p. 24).

Murtaza y Shwan inician su investigación presentando el trasfondo, la historia y el estado actual del *software* multilingüe. Evalúan la industria y comentan la relación del desarrollo de *software* multilingüe con las plataformas móviles, con los servicios de traducción automática en la web y con las metodologías ágiles. Presentan la definición de internacionalización y de localización y explican que la internacionalización tiene un efecto profundo en las actividades de diseño y codificación de la ingeniería de *software*. Explican la función del ingeniero de localización y enumeran sus competencias y responsabilidades. La gestión de proyectos multilingües, el aseguramiento de calidad, la traducción de documentación y de gráficos, y la tecnología en la traducción son otros temas que comentan brevemente en este capítulo (*Ibidem*, pp. 27-33).

[160] En inglés *cross-platform*. Son programas que pueden ejecutarse en varios sistemas de computadoras, como por ejemplo, los programas de Microsoft Office que se pueden usar tanto en las plataformas Mac como en las plataformas PC. También se puede aplicar a equipo, por ejemplo, un teclado o un ratón que funcione en Macs y PCs (Janssen 2016, s. v. cross platform).

[161] Véase el apartado “La era de metodología antigua“ en la página 46.

El capítulo tres lo dedican a explicar la metodología que usaron a lo largo de su investigación. Los investigadores hicieron primordialmente un estudio cualitativo combinado con un pequeño estudio cuantitativo para validar los resultados.

Para el estudio cualitativo llevaron a cabo una revisión de literatura exhaustiva con Google Scholar, IEEE Xplore y la ACM Digital Library usando un conjunto específico de palabras clave. Los artículos que encontraron los dividieron por categorías (económica y financiera, administrativa y de gestión, desarrollo de *software* multilingüe y SDLC, internacionalización que depende de la tecnología o proveedor, y misceláneos) que les ayudaron a organizar y extraer la información que cada artículo aportaba. De esta revisión nos interesa resaltar varias observaciones y cuestionamientos:

> […While] reviewing literature at the start of this project, it was realized that numerous technologies, tools, practices and strategies exist for multilingual software development. Therefore, the problem of developers ignoring the development of multilingual software is not due to a lack of solutions. Further, through our observation of sample projects and subsequent interviews with project owners and members, we came to realize that the problem was due to obliviousness of local software developers about the numerous solutions, tools and approaches for multilingual software development. […What] software developers truly need is awareness and straightforward and relevant guidelines in order to increase the number of international software […]
>
> (*Ibidem*, p. 36)

Y más adelante:

> It seemed strange that in spite of the numerous benefits […], so many software projects fail to consider internationalization.
>
> (*Ibidem*, p. 38)
>
> Now, the question remained, why so many software solutions are not internationalized? Why is it that only large software firms provide their software products and services in multiple languages? Is it a very expensive endeavor that only large firms can afford? Is it a low priority required that developers only consider after their businesses become truly global? Shouldn't

> all successful software applications be internationalized so a larger audience can be targeted from the start, which may help the company grow?
>
> (*Ibidem*, p. 39)

Entonces, el enfoque de su estudio cambia cuando concluyen que:

> The solution to the problem of ignoring software internationalization does not lie in a technical solution or integration of internationalization practices into software development methodologies, which already exist, but in providing relevant and pragmatic guidelines to decision makers and developers.
>
> (*Idem*)

La revisión de literatura la complementan con entrevistas en línea y personales a miembros que formaban parte de proyectos concretos a fin de extraer sus experiencias y sus motivaciones. Hicieron un análisis de dichos proyectos para ver si habían implementado internacionalización y localización, y, de haberla implementado, cómo. La información que extraen de este análisis los lleva a convencerse de que

> […] indeed the financial and technical uncertainties, or pure obliviousness of non-native users, led to software publishers either incompetently implementing multilingual software or simply developing monolingual software. […]
>
> (*Idem*)

Así pues, una guía con directrices para desarrollar *software* multilingüe que se presente de manera adecuada e incluya lo que implica cada una de ellas en relación con el desarrollo de *software* sería una herramienta útil para concienciar y estimular a los desarrolladores a internacionalizar y localizar sus proyectos puesto que reducirían la incertidumbre que causa el desconocimiento sobre el tema (*Ibidem*, pp. 39-40).

Una vez recopilan y redactan sus directrices, las agrupan en guías y desarrollan una herramienta de navegación ("Guideline Navigation Tool") que usarán como base para encuestar a expertos y obtener retroalimentación y datos cuantitativos sobre sus hallazgos. Los autores piensan que

> […] concise and pragmatic guidelines that are relevant to specific project and are presented in an easily accessible manner […facilitate…] software internationalization, regardless of what stage the software is in its lifecycle. […]
>
> (*Ibidem*, p. 40)

Sin embargo,

> It is not claimed that the guidelines presented in a navigable manner would eventually solve the problem of developers ignoring internationalization, but it will definitely raise awareness and encourage developers to internationalize the products by providing them with a bird-eye view of what needs to be considered and what solutions are available for their projects.
>
> (*Idem*)

El capítulo cuatro aborda la revisión de literatura, la cual los autores organizaron y agruparon en las áreas temáticas antes mencionadas (económica y financiera, administrativa y de gestión, desarrollo de *software* multilingüe y SDLC, internacionalización que depende de la tecnología o proveedor, y misceláneos). Comienzan ampliando la información que presentaron en el capítulo dos con la literatura que aporta información introductoria relacionada con el desarrollo e implementación de *software*. Explican las tres áreas que se deben tomar en consideración durante la internacionalización según las describe Abufardeh (véase página 85) y explican que el nivel de "internacionalidad" se puede medir verificando el nivel de segregación entre estas tres áreas y el núcleo de la aplicación (*Ibidem*, p. 45). Nombran cuáles son los factores y características de un proyecto que vaya a localizar un *software* existente (*Ibidem*, p. 46). Indican también que el equipo de desarrollo necesita estar familiarizado con los estándares, prácticas, terminología y *concerns*[162]. Mencionan y definen los conjuntos de caracteres y los estándares de intercambio de la industria de localización (TMX, XLIFF, además incluyen las memorias de traducción) (*Ibidem*, pp. 46-47). Subrayan la importancia de que los desarrolladores entiendan estos estándares:

[162] Véase nota 127.

> Developers involved in international software projects should clearly understand these standards and others as well as the related tools to efficiently and successfully implement internationalized software applications.
>
> (*Ibidem*, p. 47)

Además, explican que la localización no es traducción, sino que va más allá de la traducción y tiene otras implicaciones:

> Furthermore, it is important that developers today know that multilingual software development is far more than mere translation of the user interface. Translation of the user interface is just one of the issues that arise when developing multilingual software. The culture and norms in the target market give rise to other matters that software developers must consider.
>
> (*Idem*)

En la sección de la literatura relacionada con temas económicos y financieros, explican que los desarrolladores de *software* trabajan bajo presión de las partes implicadas y esto hace que enfoquen sus esfuerzos en desarrollar las funcionalidades del dominio y echen a un lado los requisitos no tangibles, como lo son los requisitos que afectarán el *software* a largo plazo o los requisitos no funcionales. Resaltan la importancia de entender que la posibilidad de localizar un *software* en distintos idiomas es fundamental para competir en un mundo globalizado. Desarrollar *software* resulta costoso y si el *software* desarrollado tiene éxito, lo más lógico es permitir que ese *software* pueda venderse en otros mercados. El éxito previo del producto aumenta la posibilidad de que ese mismo producto tenga éxito en otros mercados o culturas (*Ibidem*, p. 48).

Según la experiencia de expertos e investigaciones realizadas, la internacionalización y localización de *software* aumenta la accesibilidad en las culturas metas: "[…] usability and overall efficiency of software applications increase if the cultures of the target users are taken into consideration […]" (*Ibidem*, p. 49); sin embargo, estiman que el proceso de internacionalizar toma tiempo y es costoso. Indican los autores que los desarrolladores piensan que el costo adicional de desarrollar *software* multilingüe radica en el tiempo adicional que hay que invertir en el desarrollo, pero, se equivocan: "[…] the main cost component in many

projects is the cost of translating the content. […]" (*Idem*) y, como es lógico, este costo aumenta según se añaden idiomas (y culturas) adicionales.

En la sección que dedican a la literatura relacionada con la gestión y administración, Murtaza y Shwan nos explican que la localización e internacionalización exigen al gestor de proyectos tomar en consideración procesos y controles adicionales. Recomiendan que aun cuando el presupuesto no admita la posibilidad de localizar el *software*, este debe "[…] at least be designed and implemented as one that is internationalized (at least in theory) to facilitate future localizations" (*Ibidem*, p. 50), y que los equipos que desarrollan *software* multilingüe sean multinacionales, multiculturales y que puedan trabajar en equipo eficientemente. Los desarrolladores más pequeños, que por lo general no se pueden permitir este tipo de equipos, pueden optar por buscar proveedores de servicio en el extranjero, negocios locales que se alíen con ellos o también pueden crear equipos que abarquen personas en diversos puntos geográficos. En estos casos es importante entender que las diferencias culturales y las diferencias de hora pueden causar retrasos, falta de uniformidad en la calidad y, a la larga, aumentar los costos del proyecto (*Ibidem*, p. 51).

En cuanto a la literatura que trata sobre los procesos de desarrollo de *software* y el SDLC, los autores dividen la información recopilada según las etapas del SDLC pues consideran que lo que se discute con relación a cada una de estas etapas se puede incorporar a las distintas metodologías de implementación que existen. Estas son: estudio de viabilidad, ingeniería de requisitos, arquitectura y diseño, programación, control de calidad y mantenimiento posimplementación (*Idem*).

Durante el estudio de viabilidad, Murtaza y Shwan sugieren que se hagan análisis de mercado, análisis de los posibles escenarios para el desarrollo del *software* e investigación del *software* existente. También deben considerar la disponibilidad de herramientas de traducción y, para localización, traductores e ingenieros de internacionalización (*Idem*).

Para la fase de ingeniería de requisitos Murtaza y Shwan explican que es importante que las actividades que se llevan a cabo y los resultados de estas faciliten las actividades de localización e internacionalización del SDLC. Hacen referencia a los atributos de calidad que

menciona Venkatesan[163]. La internacionalización complica la cantidad de partes implicadas y aumenta la complejidad al extraer los requisitos de estas partes. No solo eso, sino también que lo que para una parte implicada puede ser un requisito fundamental, para otra parte puede que ni exista necesidad alguna de tomarlo en consideración[164]. A esto, hay que añadirle la importancia de tomar en consideración los aspectos culturales y los específicos al idioma. Explican los autores los niveles de internacionalización y localización según los presenta Abufardeh[165] (*Ibidem*, pp. 52-54).

La fase y actividades relacionadas con arquitectura y diseño no están claramente delimitadas en las metodologías o procesos ágiles en comparación con el modelo de cascada. Sin embargo, las consideraciones arquitecturales y de diseño son especialmente importantes al desarrollar proyectos multilingües. Abufardeh y Magel (2009) apoyan la idea de usar la programación orientada a *aspects*[166] para evitar que los *concerns* queden enmarañados y unificar los *concerns* que están dispersos[167]. Para identificar y separar los módulos y códigos que dependen del *locale* sugieren categorizarlos de la siguiente manera:

- Controles localizados de la interfaz del usuario
- Núcleo del producto
- Lógica específica para una cultura
- *Software* de terceros
- Procesamiento de datos (funciones relacionadas con las bases de datos)

[163] Los atributos de calidad se discuten en la página 101 y subsiguientes.

[164] Por ejemplo, una ley específica que sea aplicable únicamente a un *locale*.

[165] Véase página 82.

[166] Véase nota 142.

[167] *Idem*.

Explican que el desarrollo de *software* se puede clasificar basándose en dos tipos de acercamientos: el de programación y el arquitectural[168], y seguido presentan una descripción de ARMMS[169] como ejemplo del acercamiento arquitectural. Murtaza y Shwan apuntan a continuación que la parte más evidente de las aplicaciones multilingües es la interfaz del usuario y destacan temas de importancia relacionados con la IU (expansión/contracción de textos, idiomas bidi). La creación de prototipos ayudará a medir la respuesta de los usuarios a distintos diseños. También indican que es importante considerar el almacenamiento de datos en las bases de datos (esquemas de codificación de caracteres[170] que se pueden usar) (2012, pp. 55-60).

La sección que cubre la etapa de programación incluye recomendaciones sobre qué debe saber o tomar en consideración un programador sobre las propiedades de los distintos idiomas que afectan al *software*, el tipo de apoyo que brinda la plataforma de desarrollo que se haya escogido, la externalización de los textos, formatos de moneda, fecha, cifras, calendarios, etc., qué apoyo multilingüe ofrecen los depósitos de datos (sistema operativo, bases de datos), qué control ejerce el sistema operativo sobre la configuración de textos, formatos de fecha, etc. (*Ibidem*, pp. 61-63).

En la sección que dedican a la fase de pruebas y de aseguramiento de calidad los autores explican las pruebas de internacionalización que se deben realizar y las pruebas recomendadas durante y luego de la localización. Para lo que tiene que ver con mantenimiento posimplementación Murtaza y Shwan explican qué debe suceder luego de entregar el proyecto y cómo podría cambiar según la estrategia de implementación usada para el proyecto. En esta etapa se podrían añadir más idiomas a la aplicación. Explican, para proyectos ya desarrollados, las posibles alternativas para internacionalizar el *software* (*Ibidem*, pp. 63-67).

[168] Lo toman de (Venkatesan 2008, pp. 19-32)

[169] Véase página 117.

[170] Véase nota 197.

La literatura relacionada con internacionalización que depende de la tecnología o proveedor cubre aspectos relacionados con cómo afecta la selección de proveedores o de tecnología específica a la hora de crear *software* multilingüe. En el análisis de lenguajes de programación describen las prestaciones y limitaciones de cada uno. Se limitan a discutir los siguientes lenguajes, que solo cubren las plataformas Windows, Mac y servidores web:

Lenguaje	internacionalización	localización
Java	Sí - por medio de clases y paquetes que proveen la funcionalidad. Funciona en las plataformas *core, desktop, enterprise* y móvil.	Sí - añadir texto traducido, fuentes y métodos de *input*.
.NET *Framework* (Visual Basic, Visual C#, Visual F#, JavaScript	Sí - clases.	Sí - incluye gestor de recursos.
C, C++	No directamente, pero existen directrices.	No directamente, pero existen directrices; se pueden usar herramientas de localización.
JavaScript	No directamente, pero existen directrices.	No directamente, pero existen directrices; se pueden usar herramientas de localización.
Perl	Similar a C y C++.	Similar a C y C++.
PHP	Por medio de bibliotecas y *frameworks*.	Por medio de bibliotecas y *frameworks*.
Python	Con GNU gettext.	Con GNU gettext.

Lenguaje	internacionalización	localización
Ruby	Ruby i18n Framework.	Sí, con API[171] y programación.
Objective C (Apple)	Lista de pasos que hay que seguir.	Se pueden usar herramientas de localización.
Android SDK	Es una extensión de Java.	Es una extensión de Java.
ActionScript (Flash)	No directamente.	No directamente.

Tabla 2.3 - Lenguajes discutidos por Murtaza y Shwan

No todos los *frameworks* para desarrollar permiten la internacionalización y la localización, así que es importante seleccionar uno que sí permita la internacionalización y la localización (*Ibidem*, pp. 67-70). Explican, además, las herramientas de localización que existen y con qué lenguajes y plataformas funcionan (Alchemy Catalyst, SDL Passolo, GNU gettext y otros) (*Ibidem*, pp. 70-71).

En la sección de literatura miscelánea mencionan los atributos de calidad que afectan la internacionalización (seguridad y ejecutoria), la integración con aplicaciones de terceros y terminan diciendo:

> Translation of the content based on culture and language is one of the most challenging tasks of multilingual software development. Understanding Machine Translation technologies, other resources like Translations Memories,

[171] *Application Program Interface*. Son una serie de parámetros o requisitos para que dos programas se puedan "hablar" y hacer uso de sus funcionalidades. Por ejemplo, Google Maps tiene una serie de API que les permiten a otras aplicaciones incrustar sus mapas dentro de ellas. Así el programador puede aprovechar las funcionalidades de Google Maps sin tener que crear un programa que despliegue mapas. Las API permiten también mover información entre programas, como por ejemplo, las funciones de copiar y pegar de los sistemas operativos de las microcomputadoras. Las API "exponen" al mundo exterior, pero de manera limitada, algunas de las funcionalidades internas de los programas (Proffitt 2013).

> and knowing their limitations and utilizing human translators effectively can minimize cost and improve productivity. Utilization of localization vendors, freelance translators and approaches such as crowd-sourcing can provide additional solutions.
>
> (*Ibidem*, p. 73)

El capítulo que sigue cubre los resultados de la investigación, en específico las directrices que desarrollaron. Hacen un recuento detallado de las entrevistas que llevaron a cabo para averiguar las técnicas usadas para implementar (o no implementar) sitios web multilingües, las limitaciones que estos proyectos presentan y el proceso de toma de decisiones relacionado con estos sitios web. Observaron que en muchas ocasiones se seleccionaron sistemas comerciales o de código abierto existentes y que aun cuando uno de los requerimientos era facilitar el uso de interfaces multilingües, no se consideraba o no se discutía la capacidad que los productos evaluados tenían para ofrecer estas prestaciones. También indican que la traducción de contenidos a múltiples idiomas y la subsiguiente gestión de estos textos representaban un reto clave y un obstáculo a la hora de desarrollar y dar mantenimiento al *software* multilingüe. Muchos de los entrevistados pensaban que la tarea de internacionalizar un proyecto existente, sobre todo si se tenía disponible el código fuente y se podía acceder a los desarrolladores, era fácil. Consideraban también que los costos de traducción eran altos puesto que el contenido se modificaba constantemente. Las respuestas a las preguntas que se les formularon los llevaron a concluir que "internationalization was not a hurdle and the main challenge is the localization of the software into different locales, which requires cross-cultural expertise and professional translators" (*Ibidem*, pp. 74-79).

Las directrices que los autores desarrollan las presentan categorizadas y enumeradas por un código de la siguiente manera:

- Guías introductorias: IG01-IG04
- Guías económicas y financieras: EF01-EF05
- Guías administrativas y de gestión: AM01-AM07
- Guías para el estudio de viabilidad: FS01-FS04
- Guías para ingeniería de requisitos: RE01-RE06

- Guías arquitecturales y de diseño: AD01-AD12
- Guías de programación de *software*: PG01-PG09
- Guías para hacer pruebas al *software*: TG01-TG06
- Guías para la fase de posimplementación y mantenimiento: PR01-PR06

Estas directrices comprenden un total de 49 puntos o sugerencias. Al diseñar la herramienta de navegación de las guías (GNT[172]) determinaron que los proyectos de desarrollo de *software* multilingüe pueden estar en una de tres etapas: en el estudio de viabilidad (FS), en desarrollo (DV) o en posimplementación (PR). Cada una de estas etapas se coloca en tablas aparte que contienen las propiedades del proyecto (PP) que hay que considerar y los códigos que corresponden a las guías aplicables.

La GNT-FS incluye las propiedades de:

- Cantidad de idiomas meta
- Tamaño del contenido estático en palabras
- Ciclo del contenido (si cambia con regularidad o si principalmente es estático)
- Tipo de desarrollo (a la carta o COTS[173])
- Precio del *software*

La GNT-DV incluye:

- Metodología de implementación (iterativa o secuencial)
- Fase de implementación (información relacionada con cada fase)
- Estado actual del proyecto (si está adelantado, a tiempo o retrasado)
- Presupuesto disponible
- Cantidad de idiomas meta (igual que en GNT-FS)

[172] Esta y las siguientes, por sus siglas en inglés.

[173] *Commercial Off-The-Shelf* o aplicaciones comerciales que están disponibles para el público en general. Office es un COTS, por ejemplo.

- Tamaño del contenido estático en palabras (igual que en GNT-FS)
- Ciclo del contenido (igual que en GNT-FS)
- Precio del *software* (igual que en GNT-FS)

La GNT-PR incluye:

- Disponibilidad del código fuente
- Disponibilidad del equipo de desarrollo
- Internacionalización
- Localización
- Presupuesto disponible (igual que en GNT-DV)
- Cantidad de idiomas meta (igual que en GNT-FS)
- Tamaño del contenido estático en palabras (igual que en GNT-FS)
- Ciclo del contenido (igual que en GNT-FS)
- Precio del *software* (igual que en GNT-FS)

Terminan dando tres ejemplos prácticos de cómo usar la GNT (*Ibidem*, pp. 79-99).

El capítulo seis, de una sola página, explica cómo validaron la GNT y obtuvieron retroalimentación sobre el uso y aplicación de la GNT.

Terminan la investigación con un resumen de sus hallazgos y logros. Esta investigación "[…] contributes by compiling all previous works and presenting them in an accessible manner" (*Ibidem*, p. 101). Recalcan la necesidad de concienciar a la industria sobre los beneficios directos e indirectos de desarrollar *software* multilingüe. Las guías aportan una visión general en cuanto a las actividades técnicas y no técnicas de la internacionalización y la localización de *software*, con lo cual se facilita el "[…] multilingual software development by reducing the complexity and time required to understand the different aspects related to the topic […]" (*Idem*). Entre sus limitaciones (las de las guías) señalan que su investigación se limita al desarrollo de *software* multilingüe y que la aplicación de la internacionalización y la

localización en el ámbito del *software* embebido[174] está por estudiarse. Aclaran que resulta imposible cubrir todo tipo de herramientas y lenguajes de programación, pero que se han incluido en la investigación los principales. La selección poco diversa de proyectos usados como ejemplo (principalmente CMS[175]) provee una visión limitada de los retos que uno puede enfrentar al desarrollar *software* multilingüe, sin embargo, las entrevistas a profesionales de la industria han ampliado el entendimiento en cuanto a las preocupaciones, retos y motivaciones para desarrollar *software* multilingüe. Mencionan que hace falta que se desarrolle aún más el campo de la MT a fin de obtener traducciones de alta calidad y facilitar la traducción, que es uno de los principales obstáculos de la localización (*Idem*).

La GNT no ofrece la precisión deseada pues la idea original era tomar en consideración todas las posibles respuestas de las PP y sugerir directrices específicas, pero ello hubiese significado una enorme cantidad de combinaciones posibles. Por ello se optó por unas directrices más genéricas y por relacionarlas de manera general y simple con las PP. En el futuro, sugieren, se podría desarrollar una herramienta de *software* que tomara en consideración todas estas posibles combinaciones y que ofreciera resultados más a la medida. También señalan que algunas directrices se contradicen, dado que son muy generales, con lo cual el lector debe evaluarlas de manera subjetiva para escoger la opción que mejor le convenga al proyecto que está desarrollando (*Ibidem*, pp. 101-102).

[174] Los sistemas embebidos (también integrados, incrustados o empotrados) son "[…] a microprocessor-based system that is built to control a function or range of functions and is not designed to be programmed by the end user [… A] user can make choices concerning functionality but cannot change the functionality of the system by adding/replacing software. […]" (Heath 2003, p. 2) Una definición un poco más abarcadora la presenta Barr: "A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, as in the case of an antilock braking system in a car. […]" (2015, s. v. embedded system)

[175] Content Management System.

Las implicaciones para la industria son: que los desarrolladores pueden usar estas directrices para planificar e implementar el desarrollo de *software* multilingüe de manera más informada.

> Textbooks and technical tutorials seldom consider internationalization and localization and this has led to different tools and approaches on different platforms and in different implementation technologies.
>
> (*Ibidem*, p. 102)

Por esa razón, los compradores de *software* no saben cómo evaluar estos productos en cuanto a los requisitos de múltiples idiomas, los gestores de proyecto y empresariales no saben cómo incorporar el personal adecuado, y los desarrolladores que carecen de este tipo de experiencia no pueden hacer estimados de desarrollo adecuados o, peor aún, implementan mal este tipo de *software*. Las grandes empresas tienen el presupuesto para experimentar, pero para las pymes no es lo mismo ya que tienden a ignorar estos procesos, con lo cual incurren en pérdidas a corto y a largo plazo. Estas directrices pueden ayudar a la industria del desarrollo de *software* a tomar mejores decisiones que redundarán en beneficios y productos de mejor calidad (*Idem*).

Para la academia, esta investigación ayuda a organizar por categorías las publicaciones existentes relacionadas con el tema de la internacionalización y la localización. Al agruparlas se puede detectar qué áreas necesitan mayor investigación. Ponen como ejemplo la "[…] lack of high quality automated translation tools as a major hurdle to implement multilingual software […]", con lo cual "[…] alternative methods and improved automated translation tools are much needed and can add tremendous value towards multilingual software development". En cuanto a esta observación, pensamos que el valor añadido que pueda aportar la traducción automática no es tan importante como la disminución de la calidad del *software* traducido por la falta de contexto —que se puede solventar por medio de comentarios relevantes— que experimenta el traductor al recibir las cadenas de texto, o simplemente porque el programador, por no tener noción alguna de qué impacto tiene la construcción de la cadena, la programa de

forma tal que produce efectos no deseados en la interfaz[176]. También indican que han detectado factores clave que desalientan el desarrollo de *software* multilingüe, tema al cual la academia puede aportar soluciones (*Idem*).

2.3.4.1 Recopilación sobresaliente y asequible

A partir de la extracción, análisis y clasificación de la literatura disponible, los autores preparan una serie de directrices prácticas y un sistema para navegarlas que permite a las partes implicadas en el desarrollo de programas multilingües tomar decisiones informadas relacionadas con el tema de la internacionalización y la localización. Puesto que está basado en una recopilación y clasificación muy completa del tema, este trabajo de fin de máster se puede considerar documento fundamental para entender el impacto que tiene el proceso de internacionalización sobre un proyecto de desarrollo de *software*. Expresan los autores que "this thesis project contributes by compiling all previous works and presenting them in an accessible manner" (*Ibidem*, p. 101). Esta es una de las más importantes aportaciones de este trabajo investigativo.

En cuanto a las directrices que producen,

> […s]ome of the guidelines are general, others are for project owners and non-technical managers concerned with financial implications. Additionally, a large number of guidelines are provided for software project team members,

[176] Una app que registra actividades físicas llamada "Runkeeper" está plagadas de problemas de localización en su interfaz en castellano. Para dar un ejemplo, en las cadenas de texto que presenta en el historial de actividades realizadas por el usuario y sus amigos, "Tú caminaste 1,71 mi" y "Juan corriste 4,16 mi" resultan extrañas en castellano. Es evidente que surge de la construcción de la cadena de texto con variables de sustitución "%1 %2 %3" donde %1 sustituye a quien realizó la actividad (Tú/Juan), %2 sustituye el tipo de actividad, en este caso correr, y %3 sustituye la distancia recorrida durante la actividad. En inglés funciona sin problemas, "You ran 1,71 mi" o "Juan ran 4,16 miles", pero en castellano suena más natural si elides el sujeto en el primer caso, "Caminaste 1,71 mi" y, en el segundo caso, si cambias la persona del verbo, "Juan corrió 4,16mi". El programador parte de la premisa de que la construcción sintáctica del inglés la comparten todos los idiomas.

> who might need to develop multilingual software or modify existing monolingual software to support new languages […]
>
> (*Ibidem*, p. 18)

con lo cual incluyen conocimiento relacionado con el tema que interesa a todas las partes implicadas en este tipo de proyectos y así logran el fin que destacamos antes: proporcionar unas directrices que sean "relevant and pragmatic […] to decision makers and developers" (*Ibidem*, p. 39).

Similar a Abufardeh cuando habla de tener a disposición del equipo un experto en traducción, Murtaza y Shwan destacan las características y experiencias que tiene que tener el ingeniero de localización muy al principio de su investigación, pero fallan en hacerlo un participante activo en el proceso de desarrollo de programas. Incluso, si miramos la lista de términos y definiciones que acompañan el principio del texto, lo reducen a una "[…] person who is responsible for analysis and preparation of localization files and for localization and testing of GUI, online help, and web sites […]" (*Ibidem*, p. 15). En el próximo capítulo veremos más a fondo la descripción de estos autores del ingeniero de localización junto con nuestra propuesta de quién debe ser esta importante figura.

2.4 Un solo punto de vista, una sola visión

Apenas encontramos cuatro investigaciones que escudriñan el proceso de incluir la localización dentro del desarrollo de *software* y ninguna de ellas intenta acercar a este proceso al experto en el tema de la localización. Hablan de la importancia de que un *software* sea multilingüe, de integrar en el equipo de trabajo personas con conocimientos en otras lenguas y culturas, de que el *software* esté internacionalizado, pero todos los autores fallan en mirar más allá y realmente presentar una propuesta integradora y transdisciplinar.

Concurrimos de forma parcial con las observaciones que hacen Murtaza y Shwan en cuanto a que no hace falta inventar la rueda, pues ya está inventada:

> The solution to the problem of ignoring software internationalization does not lie in a technical solution or integration of internationalization practices into

> software development methodologies, which already exist, but in providing relevant and pragmatic guidelines to decision makers and developers.
>
> (*Ibidem*, p. 39)

Sí, la integración de las prácticas de la internacionalización en las metodologías de desarrollo de *software* ya existe; sí, las soluciones técnicas para solventar los problemas inherentes a la internacionalización ya están creadas. Sin embargo, no solo hacen falta directrices relevantes y pragmáticas, labor que estos autores ejecutan con profundidad y hasta la saciedad, sino que también es necesario divulgarlas, aplicarlas y, proponemos nosotros, integrarlas a otras áreas de conocimiento. El capítulo siguiente recoge nuestra propuesta integradora.

En consonancia con los objetivos A y B, y con las preguntas de investigación A y B, hemos presentado una breve reseña histórica sobre el surgimiento de la industria de la localización, los conceptos relacionados con ella, cómo se percibe esta área de estudios en los estudios de traducción y un resumen crítico de las investigaciones sobre localización de *software* que se han llevado a cabo hasta el momento.

Capítulo 3: El internacionalizador, un híbrido de la traducción y la informática

Las cuatro investigaciones que acabamos de reseñar aportan fundamentos básicos para entender el metalenguaje, el raciocinio lógico y los procesos relacionados con el desarrollo de *software* y la manera en que se percibe y se implementa la función del localizador a lo largo de este proceso. Su contenido es material valioso que explica distintas estrategias para abordar la internacionalización e integrarla dentro del proceso de desarrollo de *software*. Cualquier traductor que desee incursionar en el mundo de la localización de *software* tendrá una ventaja adicional si conoce y estudia estas propuestas, pues su estudio le permitiría posicionarse como participante activo dentro de este proceso. Sin embargo, esta posible participación se ve truncada ya que estos investigadores no presentan una visión más integradora del localizador y, con ello, pierden una valiosa pieza clave que puede facilitar el proceso de internacionalización: el traductor.

Dos de las investigaciones reseñadas mencionan la importancia de contar con la presencia de un traductor, figura a veces reducida a la de un *native speaker*, dentro del proceso de desarrollo de *software*. Las tareas que Abufardeh le atribuye están más bien relacionadas con la gestión de la interacción entre proyecto y localizadores (preparación de contenido y organización de los ficheros, entre otros), que sigue el modelo de caja negra que siempre se ha usado. Como un *además*, menciona que el "experto en traducción", como él lo llama, puede ayudar al equipo de desarrollo a integrar y comprender conceptos de internacionalización (que llama globalización cuando debería referirse a ello como internacionalización) en los requisitos del proyecto.

> […] Having a translation expert available for the development team will surely have great advantages. […] A translation expert can help in content preparation, organizing the source file structures, and in coordinating testing aspects of the globalized product. Furthermore, **the translation expert can help the development team understand and embed globalization requirements such as format and style needs of local languages. This can**

> **reduce the need for code modifications at later stages which are usually expensive and a cause for delays**.
>
> (Abufardeh 2008, pp. 113-114, énfasis añadido)

Los beneficios que se obtienen de esta inclusión no son fáciles de cuantificar, y quizás esta es una de las razones por las cuales no se incluye la participación de estos expertos en los proyectos de desarrollo. Integrar a un experto traductor a lo largo de todo el proceso se refleja en la cuenta de resultados del proyecto como un *costo adicional*. Los beneficios, lo que gana el proyecto, no se ve pues por lo general es algo que se ha evitado, un *cost avoidance*. Cuando se evitan retrasos y modificaciones en etapas posteriores porque el experto advierte sobre las consecuencias de una u otra decisión en etapas tempranas, se evitan gastos de reelaboración (*rework*). Cuando se optimiza la extracción e intercambio de archivos de localización porque el experto integra estrategias para facilitar y gestionar con eficiencia los procesos de localización según se ha ido desarrollando el *software*, se evitan gastos que se cuantifican en tiempo y, por extensión, en dinero. Cuando la información contextual que recibe el localizador es pertinente y este trabaja con mayor precisión y eficiencia, se evitan gastos que se cuantifican en tiempo, y se evita la reelaboración (de textos traducidos incorrectamente). Aún menos tangibles son otros beneficios, como, por ejemplo, una mejora en la calidad comunicativa del producto, un nivel de aceptación alto por la eliminación de las limitaciones lingüísticas que pudiera tener un usuario que no conoce completamente el idioma en que se muestra la interfaz.

Los aportes de la participación del localizador también los reduce a "requisitos de formato y necesidades de estilo" de la localidad en cuestión, pero sabemos que, para que una localización tenga buena acogida[177], hay que tomar en consideración a lo largo de todo el proceso de desarrollo otros factores que van más allá de estos requisitos y necesidades. Además

[177] Una buena acogida implica una total invisibilidad de los procesos de internacionalización y de localización. Los usuarios de la versión localizada no deben percibir que el producto sea una localización. Tienen que pensar y sentirse como si el programa hubiese sido una creación original para ellos. "The aim of localization is to give a product the look and feel of having been created specifically for a target market, no matter their language, culture, or location." (GALA 2016c)

de todo lo relacionado con el dominio del lenguaje que los investigadores mencionan como factores que hace falta integrar y tener en cuenta a la hora de desarrollar programas informáticos, los elementos culturales, las cuestiones jurídicas y los gustos y preferencias del mercado afectan la manera en que se recibe un programa de computadora. Ello comprende en parte el aval de competencias del nuevo localizador.

En ocasiones la figura del localizador no queda bien parada en estas investigaciones, a pesar de que sus autores hacen un análisis amplio y profundo de qué es y qué implica la internacionalización y la localización. En el caso de Abufardeh, si bien en el capítulo cuatro de su tesis hace referencia al experto en traducción que acabamos de mencionar y más o menos acierta en sus posibles aportaciones al proyecto, cuando llegamos a las recomendaciones que hace en el capítulo ocho se conforma con tener un "professional native speaker" en el proyecto:

> Having a professional native speaker of the targeted languages on site or as a consultant is always a plus. This helps by saving time and money, and it also provides the development team members with rapid feedback throughout the development process.
>
> (*Ibidem*, p. 257)

¿Qué cualifica a una persona como un "hablante nativo profesional"? ¿Qué conocimientos aporta al desarrollo de *software* un "hablante nativo profesional"? ¿Qué labores lleva a cabo un "hablante nativo profesional" en este entorno? ¿De qué manera esta persona puede ahorrar tiempo y dinero? ¿Cómo esta persona puede aportar su conocimiento y experiencia para crear un producto de mayor calidad y de mayor efectividad comunicativa? La figura del localizador sigue siendo desprofesionalizada y subutilizada en el proceso de desarrollo de programas informáticos.

El dúo Murtaza y Shwan otorga un cariz ingenieril a quien interviene en el desarrollo de *software* multilingüe y le llama *localization engineer*[178]. Con más conocimiento técnico (y,

[178] Los autores extraen esta descripción del *software localization engineer* de (Esselink 2000).

por lo tanto, con posibilidad de ser más relevante para las partes implicadas), los autores expanden en gran manera las posibles aportaciones que este puede hacer al proyecto.

> […T]eams developing multilingual software must ensure the existence of *localization engineers*. This is a *software engineer specializing in localization of software through experience and familiarity with practices and tools*. In our view, the larger the project, the more localization engineers should be present in the team and they must lead the internationalization and localization efforts. […] *Localization engineers are in essence software translators*. Just as a linguistic translator systematically translated text from one language to another, a localization engineer translates the software from one language to another. In particular, the graphical user interface (GUI or UI), or all elements that the end user would interact would, need to be translated or localized, and as mentioned before, this is not the mere translation of the textual content, but also taking care of numerous other aspects such as the backend and the database. UI elements such as dialog boxes, menus and strings need to be translated. Additional elements and aspects of the software that must be attended to in order to internationalize and localize a product successfully are considered and this includes the UI, database, the target platform and more.
>
> (Murtaza y Shwan 2012, p. 30, énfasis añadido)

Si bien la esencia de esta descripción la toman de Esselink, el hecho de que estos autores destaquen la importancia de que *los ingenieros de localización son en esencia traductores de software* cambia drásticamente el enfoque en que la industria de la localización trabaja y se acerca a nuestra propuesta desde la dirección contraria. Este localizador experto, a quien llamaremos *internacionalizador*[179], proviene del campo de la *traducción* (y no de la *informática*)

[179] En este sentido, puede haber paralelismos con el ámbito del comercio exterior y la internacionalización de las empresas puesto que desde las instituciones oficiales (en el caso de España, el Instituto de Comercio Exterior, la Agencia Andaluza de Promoción Exterior o Extenda, o las cámaras de comercio) se empieza tímidamente a alentar a las empresas y a las ONG que quieren internacionalizar su actividad a que incluyan en sus plantillas traductores e intérpretes con formación específica, entre otras, en comercio exterior. La tesis doctoral presentada por Álvarez García (2015) analiza la necesidad que existe de formar traductores altamente especializados en el ámbito del comercio exterior y describe las competencias que deben tener los titulados en traducción e interpretación en el marco del comercio exterior español. Por otro lado, el artículo de Aguayo Arrabal (2012) investiga la conexión entre las actividades de traducción e interpretación con el comercio exterior y estudia la (no-)inserción de estos profesionales dentro de este ámbito comercial.

y se acerca al espacio del desarrollo de *software* por medio de unos conocimientos específicos que lo capacitan en esta área de conocimiento. El *internacionalizador* puede, cómo no, provenir del campo de la informática y ser un conocedor en el desarrollo de programas informáticos que se educa para hacerse experto de la traducción y la mediación intercultural: un híbrido que se desplaza con soltura en estos dos ámbitos tratados como dispares pero que por necesidad tienen que trabajar en armonía para lograr un producto que funcione en múltiples idiomas y culturas. Para Murtaza y Shwan sigue siendo un *software engineer* que ha tenido experiencia vasta en localización y las herramientas que se usan en estos procesos, pero es importante el hecho de que hayan sido enfáticos en la importancia de su presencia durante el desarrollo de un proyecto multilingüe, la relevancia que les otorgan como líderes en el proceso de la internacionalización y de la importancia de las competencias lingüísticas que deben tener (aun cuando fallan en mencionar la importancia de las competencias de mediación intercultural de estos).

Abufardeh aboga por que, durante la etapa de diseño, primen los principios de flexibilidad (a fin de que el *software* sea fácil de adaptar a las variaciones locales), de extensibilidad (para posibilitar añadir funcionalidad futura) y de la "traductividad" (medido en términos del tiempo y esfuerzo necesarios para traducir)[180]. ¿Cómo se supone que un desarrollador de *software*, que no tiene la menor idea de qué implican las diferencias lingüísticas y culturales entre *locales*, entienda cuáles son las variaciones locales de las que habla el principio de flexibilidad y lo que implica una adaptación fácil en un programa informático? ¿Por qué y para qué es necesario que el *software* admita funcionalidad futura? ¿Qué tiempo y qué esfuerzo toma traducir cadenas de texto? ¿De qué manera puede tener el traductor mejor idea del contexto al cual el programador se refiere cuando se le transmite una serie de cadenas de texto? ¿Cómo la documentación y los comentarios que el programador añade tienen un impacto sobre este contexto? ¿Cuándo, por qué y para qué hay que adaptar algo que tenga relevancia cultural? El internacionalizador es quien tiene la experiencia y el peritaje necesarios para contestar estas preguntas.

[180] Véase página 87.

3.1 El internacionalizador como parte del proceso de desarrollo de *software*

A lo largo de nuestra investigación hemos sugerido en varias ocasiones que consideramos que, en la industria, la localización de *software* se percibe y se aborda como una caja negra que se invoca en el momento en que surgen cadenas de texto para traducir. Así, la necesidad de involucrar a los localizadores surge durante las etapas de desarrollo en que estas cadenas se comienzan a producir, es decir, durante la etapa de codificación, la cual se lleva a cabo al final del SDLC (por ejemplo). En este momento, las decisiones que pueden propiciar o no la internacionalización ya se han tomado y dar vuelta atrás implicará aumentar costos, como muchos de los investigadores han explicado. La necesidad, incluso, puede surgir tan tarde como luego del lanzamiento del producto, cuando se decide ampliar la base de ventas a otros mercados. Nos explica Manuel Mata Pastor que

> […] el simple hecho de no asumir desde las etapas iniciales del diseño y desarrollo de un producto la posibilidad de que este pueda localizarse ulteriormente a otros idiomas acaban [sic] por someter al traductor/localizador a un sinfín de dificultades que obstaculizan su tarea.
>
> (2016, pp. 31-32)

Y no solo el localizador termina siendo la víctima de este descuido: la calidad del producto se verá afectada y su impacto negativo afectará la imagen del producto en los mercados meta, pues el receptor percibirá "algo" que no le encaja, un elemento foráneo, que muchas veces le cuesta señalar en dónde está, porque no es tan evidente, pero lo percibe. La imagen del producto también se ensombrece porque cuando no hacemos uso de los recursos, de las metáforas, de las resonancias propias de la cultura de destino, el mensaje será menos efectivo. Asimismo, se verá perjudicada la orientación del usuario sobre el uso de la aplicación, su aprovechamiento al máximo de las funcionalidades o de los significados que sus prestaciones puedan evocar en él (Torres del Rey 2016). Es por ello que el internacionalizador tiene que formar parte del equipo de desarrollo de *software* desde las etapas más iniciales hasta las etapas finales del proceso, para ayudar a guiar al equipo en la dirección correcta en cuanto a qué hacer para internacionalizar el proyecto.

O'Hagan y Mangiron (2013) parten de la localización de videojuegos para hacer un llamamiento a implementar un proceso de desarrollo de *software* que sea *localization-friendly* que no se puede desoír.

> […] As promoted by the concept of GILT, companies are constantly advised to approach localization according to their wider global strategies by considering the implications of localizing their products at an early stage of product development through the process of internationalization […] In order to make localization economically efficient, it is critical to implement localization-friendly game development through internationalization at the outset. […] For complex products such as digital games, systematic fore-thinking is becoming a necessity.
>
> (*Ibidem*, p. 112)

Las implicaciones en términos de costos son enormes y precisamente por ello se aconseja a las empresas que tengan en cuenta la internacionalización durante las etapas tempranas del desarrollo del producto. Además, y como es lógico, según aumenta la complejidad de un proyecto, la necesidad se torna más imperante. Chandler y Deming (2012), también desde el punto de vista de los videojuegos[181], explican que este no es siempre el caso, aun cuando la perspectiva ha ido cambiando poco a poco.

> While the trend is slowly changing, most localization efforts on a game title currently happen later in the development cycle, which leaves little room for ensuring a completely thorough and effective localization. The better examples

[181] No solo desde la industria de los videojuegos se aboga por abordar el desarrollo de un producto partiendo de una perspectiva global. Dray y Siegel (2006) explican que el diseño centrado en el usuario aboga por que se comience a diseñar cualquier tipo de producto a partir de un entendimiento profundo de todos los usuarios del producto, de cómo el producto encaja en el contexto cultural, social, tecnológico y físico de sus usuarios, y que se consideren sus necesidades a lo largo de todo el proceso de desarrollo. En este artículo establecen, entre otras perspectivas, los paralelismos que existen entre los problemas que afrontan la localización y el diseño centrado en el usuario.

> of game localization are often the cases where localization considerations were addressed early on and throughout the project […].
>
> (*Ibidem*, p. 20)

No podemos perder de vista que los mejores ejemplos de localización los componen aquellos proyectos en que se tomaron en consideración los aspectos relacionados con la localización *desde las fases iniciales* y *durante* el desarrollo del proyecto. Estas autoras también hablan de lo importante que es crear código que sea "prolocalizable", como acabamos de señalar, aun cuando no se tenga la expectativa de que el programa vaya a ser lanzado en otros mercados.

> Localization-friendly code is developed to be easy to localize. When developing localization-friendly code, asset organization, asset integration, and linguistic testing are taken into account, along with international alphabets, user interface (UI) design, and compatibility between languages. Even if a developer is working on a game that does not have localized versions currently planned, they may be needed later or after the original game has shipped. If the code is localization-friendly to begin with, creating international versions is much easier and resources are used more efficiently. Developing localization-friendly code is a good practice for all game developers, because it saves many localization headaches in the long run.
>
> (*Ibidem*, p. 121)

Enfatizan, además, que es importante considerar todo esto antes de comenzar a codificar, pues los resultados pueden ser nefastos debido a la magnitud y complejidad de hacer esto *a posteriori*.

> Trying to retrofit code so that it is localization-friendly is not recommended; it is time-consuming, challenging, and has the potential to introduce a number of bugs. It can be done, but the time and risk must be carefully considered before attempting a code change of this magnitude.
>
> (*Ibidem*, p. 122)

Roturier subraya que la manera en que se internacionaliza un producto afecta los procesos ulteriores (como la traducción o la adaptación) y los hace, a la larga, más eficientes. (Roturier 2015, p. 186)

> From an internationalization perspective, should (and if so, how?) the developer be alerted when they write code in a locale-specific manner? One could argue that this type of work is the remit of quality assurance, but global efficiency gains may be realized if these issues were taken care of during the development process.
>
> (*Ibidem*, p. 188)

No hace falta discutir la necesidad de integrar la internacionalización en todo proyecto de desarrollo de *software* nuevo. Ya hemos visto la manera en que ha cambiado el ecosistema de la informática desde sus principios y hasta el presente, y cómo en la actualidad es un espacio amalgamado en el cual no podemos detectar límites entre plataformas. La ubicuidad de la internet hace de esto, para efectos prácticos, algo obligatorio. Y el camino continúa ampliándose en esa misma dirección: las "cosas" de la IoT[182] ya sobrepasaron, desde 2008, a los habitantes del planeta y se espera que para 2025 haya más de 50 millardos de "cosas" conectadas a la internet, un negocio a nivel mundial valorado en cerca de 11,1 billones USD anuales para esa misma fecha (Hurlburt 2015, p. 22). Jonathan Caras compara esto con el genio de Aladino, listo para complacer los deseos de su amo en todo momento, y, para complacer al amo, los aparatos inteligentes se caracterizan por los niveles de interconexión y personalización sin precedentes que manifiestan. "[…] A high-quality smart device 'knows you'. [… A] high-quality smart device can deliver a custom-tailored user experience that feels intelligent (and thus 'smart') to the end user" (Caras 2015, pp. 32-33). ¿Cómo pretendemos ofrecer esta experiencia personalizada si los programas que se valen de estas "cosas" para analizar la información que producen no están adaptados al idioma y a la cultura de quien los usa (o simplemente no conoce ni la cultura ni el idioma)? Por otro lado, la experiencia del usuario con estos aparatos inteligentes se facilita en función de cuán natural y humana se perciba la interacción. "[…T]he extent to which our devices can learn about us, and thus adapt to our needs, will enable us to achieve a more natural and human experience […]" (*Ibidem*, p. 34). El desconocimiento de los aspectos lingüísticos y culturales limitará, y es posible que hasta imposibilite, la adaptación de los aparatos a las preferencias de sus usuarios, entorpeciendo la

[182] Véase nota 60.

integración de la tecnología a quienes se valen de ella para ampliar su experiencia vital. "[…] Aligning human and computer input and output is critical in the advancement of technology" (*Idem*). En una interacción mediada principalmente por el idioma y, en consecuencia, por la cultura, una perspectiva miope de cara a la internacionalización no solo limitará las posibilidades que ofrece la IoT, sino que también retrasaría su expansión y progreso.

En el capítulo dos habíamos comentado que Murtaza y Shwan establecen en dónde radica la mayor partida en cuanto al costo de crear una aplicación multilingüe: "[…] the main cost component in many projects is the cost of translating the content […]". Sin embargo, los desarrolladores piensan que los costos aumentan por el tiempo adicional que hay que dedicar a cuestiones relacionadas con la internacionalización (Murtaza y Shwan 2012, p. 49). En la actualidad, y muy posiblemente por la importancia de internacionalizar, este argumento de los desarrolladores es cada vez más débil puesto que las prestaciones que las plataformas de desarrollo ofrecen son cada vez más "conscientes" de la necesidad de localizar y ofrecen interfaces automatizadas, basadas en estándares de amplia aceptación en la industria (XLIFF[183], por mencionar uno), que facilitan los procesos mecánicos de intercambio de cadenas de texto relacionados con la localización. También estas plataformas de desarrollo se valen de bibliotecas[184] estándares desarrolladas especialmente para manipular correctamente la información que es sensible a los *locales*. Localizar, sí, posiblemente sea el mayor costo del proceso de crear una aplicación multilingüe, pero si el producto está adecuadamente internacionalizado, se pueden aplazar la localización y el lanzamiento del producto a otros mercados, cuando ya el éxito del producto esté mejor establecido y el desarrollador se pueda permitir llevar a cabo una buena localización de la aplicación. Si el producto está bien internacionalizado, los costos, a largo plazo, se reducen.

[183] Véase nota 105.

[184] Véase nota 96.

También antes mencionamos que esta pareja de investigadores encuentra que los desarrolladores de *software* enfocan más sus esfuerzos en desarrollar las funcionalidades del dominio[185] que los requisitos no tangibles, como lo son los requisitos de internacionalización, o los requisitos no funcionales, como lo son la adaptación del producto a distintos modelos culturales. Las funcionalidades del dominio tienen presencia dominante en comparación con las relacionadas con la internacionalización pues son el objetivo primario del desarrollo que se está llevando a cabo y los programadores a cargo de un proyecto informático ni conocen ni entienden que todo *software* incluye una faceta comunicadora. El programa desarrollado se puede ejecutar con eficacia y lograr maravillas, pero si los usuarios no se enteran de lo que el programa está haciendo, no pueden interactuar con él, o no se les guía adecuadamente mediante lenguaje verbal y no verbal sobre lo que puede(n) hacer, de nada vale. Si a esta faceta comunicadora le añadimos que cada *locale* —incluso hasta podríamos hablar de *sublocales* si consideramos variantes regionales de un mismo *locale*— comprende un modelo conceptual (Norman 2013) al cual responde cada usuario internacional en particular, las necesidades de adaptación y los niveles de flexibilidad que debe presentar el *software* aumentan en función de los *locales* a los cuales la aplicación tiene la posibilidad de atender. El internacionalizador tendrá la capacidad de velar por que el *software* que se desarrolla incluya características que faciliten atender estas necesidades y facultar la localización de los programas, atendiendo una internacionalización consciente de estas prestaciones.

En muchas instancias, Microsoft, Google, Apple y las grandes empresas[186] que están relacionadas de una manera o de otra con la industria de la localización tratan de presentar la localización de un programa o de una app como un proceso simple. Estas empresas crean

[185] Véase nota 128.

[186] Tan reciente como hasta el 3 de abril de 2016 (según captura de archive.org, https://web.archive.org/web/20160403005731/https://developer.apple.com/internationalization/) Apple incluía una lista de proveedores de servicios de localización en su página de internacionalización, aunque en la actualidad ya no la incluyen (https://developer.apple.com/internationalization/). Google, por otro lado, dice que "Google Play App Translation Service can help you quickly find and purchase translations of your app" (http://developer.android.com/distribute/tools/localization-checklist.html#gp-trans).

expectativas falsas a los desarrolladores, en especial cuando dicen que localizar una aplicación es tan simple como extraer las cadenas de texto de la aplicación y enviarlas a un proveedor que las localice. Por muy bien que esté hecha una localización, sin una buena internacionalización la calidad del producto se verá afectada, pues es bastante difícil que un programador tenga la capacidad de manejar los requisitos lingüísticos y culturales que implica lograr calidad sobresaliente en un producto de *software*. Tampoco es muy probable que sea capaz de prever los usos, las interpretaciones y las interrelaciones gramáticas y sintácticas características de cada idioma, es decir, entre los elementos que dependen a nivel de la cultura, ni entre los distintos elementos lingüísticos, visuales o funcionales del idioma.

¿Puede el internacionalizador hacer las veces de un equipo multinacional y multicultural? Pensamos que sí, que un internacionalizador (o un grupo, dependiendo de la magnitud del proyecto) tendría la capacidad de aportar el conocimiento necesario y asistir al equipo de desarrollo en desarrollar un *software* completamente internacionalizado y adecuadamente documentado para facilitar su localización en el momento en que las partes implicadas decidan realizar este paso. De nuevo hacemos hincapié en lo que nos recomiendan Murtaza y Shwan, sin olvidar las palabras de Mata Pastor que mencionáramos antes, cuando explican que una aplicación debe "[…] at least be designed and implemented as one that is internationalized (at least in theory) to facilitate future localizations" (2012, p. 50).

3.2 Destrezas y conocimientos del internacionalizador

Ya mencionamos arriba lo ventajoso que puede ser para el internacionalizador que conozca las investigaciones previas que hemos reseñado en el capítulo dos pues todas ellas contienen claves que le permiten comprender el metalenguaje, el raciocinio lógico y los procesos relacionados con el desarrollo de *software* y la manera en que se percibe y se implementa la función del localizador a lo largo de este proceso. También resulta provechoso profundizar un poco en el desarrollo histórico de las computadoras para así entender el surgimiento de los lenguajes de programación y de las metodologías de desarrollo de *software*. Así, armado de una visión más amplia de su desarrollo histórico, le será más fácil formar parte

de los procesos de desarrollo de programas informáticos y de entender los principios que rigen estas estrategias y procesos.

Jiménez-Crespo (2013) señala dos components específicos a la localización que no se comparten con otras modalidades de la traducción: "[…] active co-operation between translators-localizers and development engineers, and the need for a comprehensive understanding of technological issues on the part of translators" (*Ibidem*, pp. 16-17). A tono con estas características especiales que exhibe la localización de *software* sugerimos a continuación una serie de destrezas y conocimientos que definen el perfil del internacionalizador.

Muchas de las estrategias de desarrollo de *software* se valen de herramientas visuales de documentación. Estas comprenden un lenguaje visual específico, con el cual es provechoso estar familiarizado a fin de entender no solo la documentación que se va generando a lo largo de las etapas de desarrollo, sino, además, las distintas funcionalidades que ofrecerá la aplicación, y asegurarse de que las consideraciones que afectan la internacionalización del producto se estén tomando en cuenta a lo largo de todo el proyecto. Para poder participar activamente durante todo el desarrollo de una aplicación informática, un internacionalizador debería conocer las principales herramientas visuales que se utilizan.

Otro asunto relevante que discuten todos los investigadores previos es entender la codificación de caracteres (Véase Young 2001, pp. 6-9; Abufardeh 2008, pp. 17-25; Venkatesan 2008, pp. 161-171; Murtaza y Shwan 2012, pp. 46-47), y en especial Unicode y sus prestaciones, pues es básico para asegurar una adecuada internacionalización. El internacionalizador estará en una posición de ventaja al conocer no solo el alcance y las limitaciones que impone la codificación de caracteres, sino también la *suite* de prestaciones que ofrecen las bibliotecas asociadas con este tema.

Elegir sobre qué lenguaje de programación se va a desarrollar una aplicación es fundamental a la hora de internacionalizar, pues no todos los lenguajes incluyen prestaciones que faciliten o permitan la internacionalización. El lenguaje de programación seleccionado también tendrá un efecto sobre el entorno que se pueda usar para desarrollar los programas que

conforman una aplicación internacionalizada. Así, conocer un poco sobre los lenguajes de programación y sus prestaciones disponibles en cuanto a la internacionalización le permitirá al equipo de trabajo tomar una decisión informada.

El internacionalizador, como experto en idiomas y en comunicación, puede jugar un papel importante en la redacción de las cadenas de texto de los programas a fin de facilitar la ulterior localización. Estudiar las directrices que proporcionan las guías de redacción le ayudará en esto y, a lo largo de este proceso se puede asegurar de que la documentación de estas cadenas de texto se adecue a las necesidades del localizador. Así se podrá mejorar la calidad del trabajo final. También debe poder entender qué implicaciones tiene la construcción y concatenación de cadenas, así, ayudará los programadores del proyecto a facilitar el proceso de localización y a lograr obtener un resultado que sea aceptable para el público receptor de la localización del producto final.

Todos estos puntos están en consonancia con los objetivos C y D en la página 11 de nuestra investigación y con las preguntas de investigación A y C en la página 12. A continuación, discutimos estos temas que capacitan al internacionalizador.

3.2.1 Herramientas visuales para la modelación de sistemas

Los desarrolladores de aplicaciones se valen de herramientas visuales que los ayudan no solo a documentar un sistema, sino también a entender mejor cómo funciona y a validar dicho entendimiento con los usuarios y dueños de procesos. Los principales acercamientos al analizar un sistema son desde el punto de vista de los procesos, desde el punto de vista de los datos y desde un punto de vista que analiza las partes del todo como objetos (acercamiento orientado a objetos). Cada uno de estos acercamientos se relaciona con distintas metodologías de análisis y diseño. Veamos de qué herramientas visuales se valen para describir los sistemas a fin de entender mejor el lenguaje y el metalenguaje del desarrollo de sistemas de información y aplicaciones.

3.2.1.1 Modelación de procesos

La modelación de procesos se basa en el principio de descomposición funcional que busca dividir un problema complejo en partes más pequeñas a fin de que, al dividirlo, las partes en las que se divide sean fáciles de entender. Su enfoque en procesos enfatiza el entendimiento de la lógica de cómo se lleva a cabo algo y así lograr que los componentes físicos del sistema no interfieran con el diseño. Los principales diagramas de los que se vale la modelación de procesos son los diagramas de descomposición funcional, los *data flow diagrams* y los diagramas entidad-relación, pero también existen otras técnicas y diagramas que no necesariamente son tan elocuentes en su expresión gráfica, como por ejemplo, los árboles y las tablas de decisiones, el inglés estructurado, los diagramas de acción, las matrices y los diagramas de acción[187] (Avison y Fitzgerald 2006a, pp. 109-111).

3.2.1.1.1 Diagrama de descomposición funcional

Este diagrama tiene como objetivo organizar de manera gráfica el producto de la descomposición funcional. La Figura 3.1 muestra un diagrama de descomposición funcional que explica el proceso para calcular una nómina semanal. Según se va bajando de nivel,

[187] Para más información sobre estas técnicas y diagramas véase (Avison y Fitzgerald 2006a, pp. 251-271).

aumenta el nivel de detalle de cada proceso; los niveles superiores son más abstractos que los niveles inferiores[188].

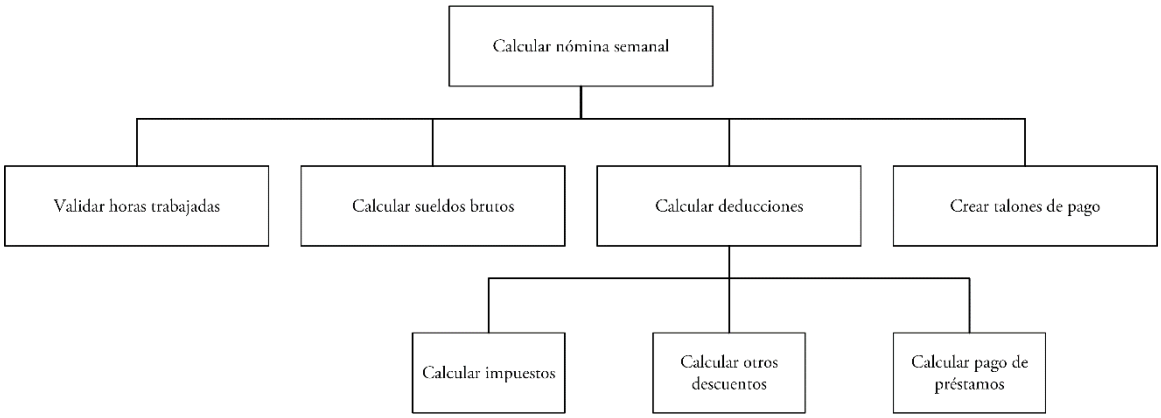

Figura 3.1 - Diagrama de descomposición funcional para el cálculo de nómina semanal (Avison y Fitzgerald 2006a, p. 110)

3.2.1.1.2 Data Flow Diagram (DFD)

Este tipo de diagrama ayuda a comunicar la noción de estructura, de cómo se ve el sistema desde el punto de vista lógico y explica más *qué* cosas va a hacer un sistema que *cómo* las va a hacer. Sirve no solo para comunicar información entre analistas y diseñadores, sino que también es comprensible para el usuario. El nivel de abstracción se puede controlar fácilmente, según el nivel de detalle que se escoja presentar (*Ibidem*, p. 243). En la Figura 3.2 vemos un DFD Nivel-0 de un hipotético sistema para ordenar comida. Las flechas indican el flujo o movimiento de datos, que nunca pueden quedar en el aire, es decir, que siempre tienen que conectar con uno de los siguientes elementos. Los rectángulos verticales numerados del 1.0 al 3.0 corresponden a los procesos o tareas que se ejecutan sobre los datos. Estos procesos cambian la estructura de los datos o generan información nueva a partir de ellos. Las cajas horizontales abiertas etiquetadas D1 y D2 son repositorios de datos temporales o permanentes. Estos repositorios no necesariamente son ficheros en una computadora; pueden ser archivos

[188] Véase el concepto de abstracción según se explica en el apartado "Arquitectura de *software*" en la página 98.

manuales, una hoja de papel con ítems enumerados o una carpeta con facturas, por ejemplo. Los rectángulos horizontales son entidades externas (al sistema documentado) que conforman la fuente de los datos o depósito final de estos. Los procesos se pueden "explotar" si descendemos al próximo nivel. En el DFD Nivel 1 del proceso *1.0 Ordenar comida* podríamos incluir los procesos internos de ordenar comida, por ejemplo: *1.1 Recibir orden de cliente, 1.2 Requisar ítems del inventario, 1.3 Rebajar ítems del inventario, 1.3 Preparar orden para la cocina* y *1.4 Enviar factura al cliente*. Estos procesos estarían recogidos dentro de su propio diagrama y servirían para dar una idea más clara de cómo funciona el proceso *1.0 Ordenar comida*, pero no se ven en el diagrama de Nivel 0. Se puede "explotar" cada uno de estos subprocesos si es necesario entrar en más detalles. Es de esta manera como se controla el nivel de abstracción que se desea presentar. A un nivel de mayor abstracción podemos preparar un "Diagrama de contexto" o "DFD de contexto", como el que aparece en la Figura 3.3, donde solo se representa el sistema sobre el cual se está trabajando y los actores o entidades externos. Si observamos bien, los cuatro rectángulos horizontales corresponden a las cuatro entidades externas que

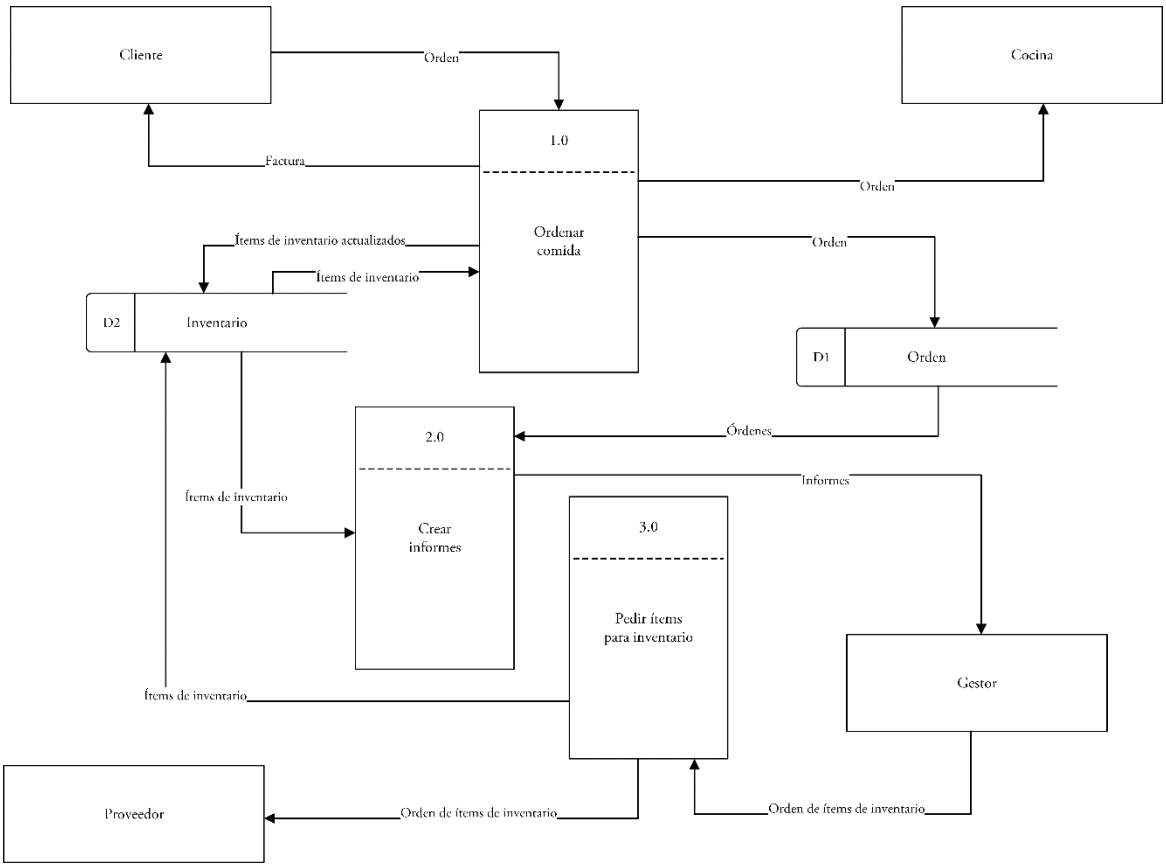

Figura 3.2 - Data Flow Diagram (DFD) *Nivel-0 (Visual Paradigm 2015)*

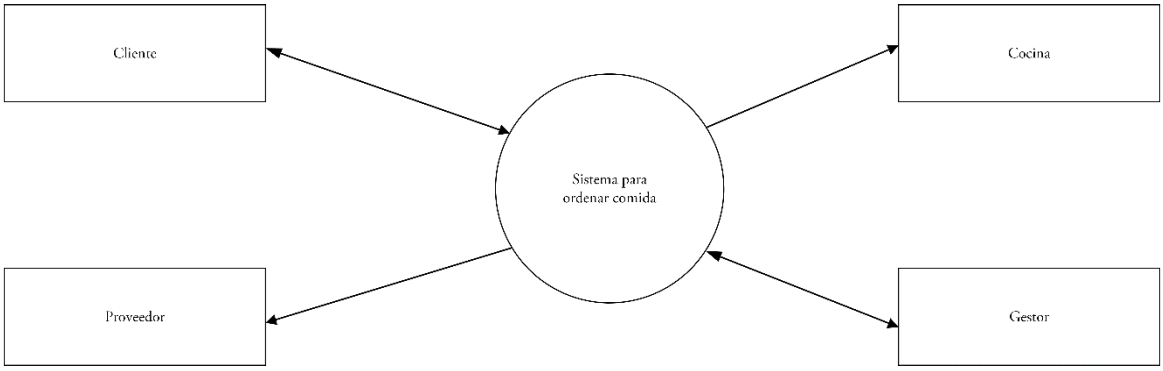

Figura 3.3 - Diagrama de contexto o DFD de contexto
(Visual Paradigm 2015)

interaccionan con nuestro sistema para ordenar comida y las flechas indican la dirección de dichas interacciones (*Ibidem*, pp. 244-251; Visual Paradigm 2015).

3.2.1.2 Modelación de datos

Este acercamiento a la modelación argumenta que la piedra angular de los sistemas son los datos y que, si se identifican los elementos de los que están compuestos y se comprende el significado de estos, sus relaciones y estructuras en profundidad, se puede generar un modelo lógico de un sistema que no depende de su implementación física. Las entrevistas a las personas que trabajan en la organización, el estudio de los documentos que circulan en esta y la observación directa ayudan a identificar, analizar y entender estos elementos de datos. La razón para escoger el análisis de datos sobre el de procesos es que sus proponentes argumentan que los datos son más estables, menos cambiantes que los procesos que se ejecutan sobre estos datos. La modelación de entidades y sus interrelaciones son la base principal de este acercamiento y el diagrama de entidad-relación es la herramienta visual que se usa para expresarlas (Avison y Fitzgerald 2006a, pp. 111-113).

3.2.1.2.1 Diagrama de entidad-relación

Los diagramas de entidad-relación muestran varios conjuntos de elementos de datos llamados *entidades* que competen a la organización o aplicación que se está estudiando. Las entidades están asociadas entre sí y estas asociaciones, que se expresan por medio de una línea, conforman una *relación*. La relación entre entidades tiene una *cardinalidad* específica: de uno

a uno, de uno a muchos o de muchos a muchos (*Ibidem*, pp. 217-229). En la Figura 3.4 vemos parte de un diagrama de entidad-relación de un sistema de gestión de matrícula.

- Cardinalidad de uno a uno (1:1) — La entidad "sección" se relaciona del modo "se reúne" con la entidad "aula" con una cardinalidad de uno a uno (ambas representadas en el diagrama por las dos barras perpendiculares a la línea). Esta relación se lee como sigue: una sección se ofrece exactamente en un aula, y un aula alberga una y solo una sección específica de un curso.
- Cardinalidad de uno a muchos (1:m) — La entidad "curso" se relaciona del modo "oferta" con la entidad "sección" con una cardinalidad de uno a muchos (representado por la pata de cuervo con la barra perpendicular). Esta relación se lee como sigue: un curso se oferta en una o más secciones; una sección pertenece a un solo curso. Como vemos, la cardinalidad en una dirección es de uno a muchos, pero en la dirección opuesta es de uno a uno. La relación sección-estudiante también es de 1:m.

- Cardinalidad de muchos a muchos (n:m) — Este sistema de gestión de matrícula permite que más de un profesor imparta parte de una misma sección y por eso la relación ente la entidad "profesor" se relaciona del modo "imparte" con la entidad "sección" con una cardinalidad de muchos a muchos: un profesor imparte una o varias secciones; una sección puede ser impartida por uno o más profesores. En este caso la cardinalidad es de uno a muchos en ambas direcciones.

Los diagramas de entidad-relación pueden evolucionar hasta describir cada entidad por

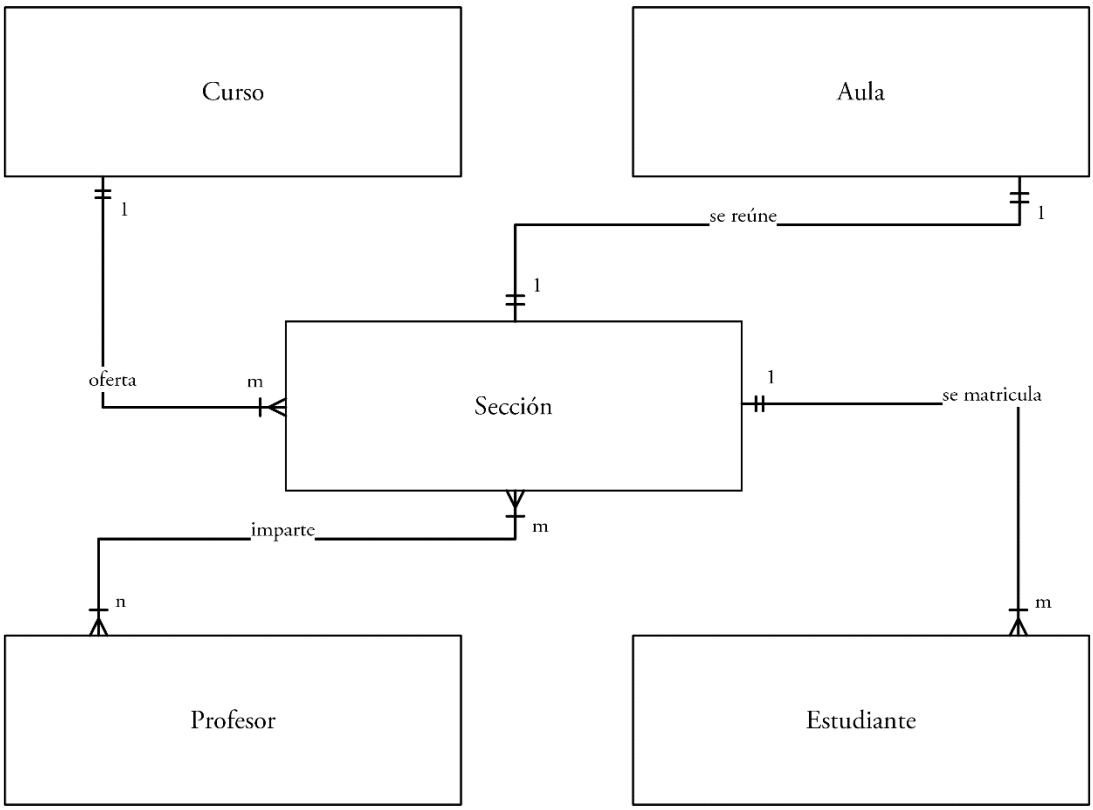

Figura 3.4 - Diagrama de entidad-relación (parcial) de un sistema de matriculación de estudiantes

medio de los elementos de datos que la componen. Esto se convierte en una herramienta de

gran utilidad al momento de desarrollar y normalizar[189] las bases de datos que el sistema o aplicación vaya a usar.

3.2.1.3 Modelación de objetos

En el apartado "La era de las metodologías"[190] hablamos de la segunda ola de las metodologías, las metodologías orientadas a objetos. Este acercamiento es distinto a los dos anteriores, pero, a la vez, similar. En la modelación de objetos se modelan los procesos y también los datos, pero se trata a ambos como un todo, un todo que se combina o encapsula dentro de lo que se conoce como un *objeto*[191](*Ibidem*, pp. 113-114). Avison y Fitzgerald citan a Coad y Yourdon (1991) cuando explican que

> […] object-oriented concepts are based on those we first learned as children; that is, objects and attributes, wholes and parts, classes and members. We learn by identifying particular objects, such as a tree, and then identifying their component parts (e.g. their attributes), for example, branches and leaves, and then to distinguish between different classes of objects, for example, those of

[189] Las reglas de normalización de las bases de datos están creadas para evitar anomalías al actualizar datos y evitar incongruencias en las bases de datos relacionales. Existen cinco formas normales, pero una base de datos se considera normalizada si se cumple con las primeras tres formas normales (Kent 1983, pp. 120-121).

[190] En la página 54 y subsiguientes.

[191] El creador del concepto *object-orientation*, Alan Kay, define las seis características de los objetos: todo es un objeto; los objetos se comunican por medio de mensajes; cada objeto tiene su propio espacio de memoria; cada objeto es una instanciación de una clase (que también es un objeto); la clase contiene los comportamientos que comparten sus instanciaciones; y las clases están organizadas bajo una estructura de árbol con una raíz única llamada la *jerarquía de herencias*, de manera que la memoria y los comportamientos asociados a una instanciación de una clase están disponibles a los descendientes de esta estructura de árbol (Budd 1993, p. 1).

> threes and rocks. This implies that the concepts should be simple and indeed the language Smalltalk[192] was originally developed for children to use.
>
> (Avison y Fitzgerald 2006a, p. 113)

También enumeran los beneficios de este acercamiento en relación con los conceptos orientados a objetos: unifican muchos aspectos del proceso de desarrollo de los sistemas de información, es decir, que todos los pasos del proceso de desarrollo de sistemas de información "hablarían" el mismo idioma puesto que se pueden realizar usando orientación a objetos; facilitan la reutilización del código y, por lo tanto, el desarrollo de aplicaciones es más rápido y robusto; integran los métodos de desarrollo de sistemas al contexto del sistema. Este último beneficio es especialmente importante pues enfatiza un cambio filosófico en el desarrollo de sistemas informáticos. El desarrollo de sistemas informáticos ya no busca desplazar al trabajador con un sistema mecanizado, sino que su fin es ampliar el apoyo a las personas para que puedan identificar y resolver problemas o mejorar la comunicación. El *lenguaje de modelado unificado* o UML, por sus siglas en inglés, es un lenguaje gráfico para modelar conceptos de análisis y diseño de sistemas (*Ibidem*, pp. 114-115, 279). Veamos los principales diagramas de los que se vale el UML para describir negocios y sistemas. Esta lista no pretende abarcar todos los tipos de diagramas UML existentes ni todas las posibles relaciones e interacciones que se pueden representar por medio de ellos[193].

3.2.1.3.1 Diagrama de casos de uso

Los diagramas de casos de uso ilustran las fronteras que delimitan un negocio o un sistema y describen, por medio de una secuencia de pasos, las interacciones entre el sistema y los distintos actores que se relacionan con él. Sirven de igual manera para documentar un sistema propuesto como para documentar un sistema existente (Daoust 2012, pp. 23-24). La

[192] Que discutimos en el apartado con el mismo nombre en la página 201.

[193] La fuente más importante de información sobre el UML es el consorcio OMG que publica los estándares para el UML (Object Management Group, Inc. 2016). El sitio web de Fakhroutdinov es posiblemente más asequible para aprender UML y como referencia (Véase Fakhroutdinov 2016).

Figura 3.5 muestra el diagrama de casos de uso para un sistema de trenes. En la esquina superior izquierda vemos "uc" (para *use case*, el nombre del diagrama en inglés) junto al nombre del diagrama. Los actores se representan con muñecos de palitos y se identifican debajo; los casos de uso son los óvalos, y el rectángulo que los contiene es la frontera del sistema. Las flechas o rayas se conocen como asociaciones y representan una relación entre dos nodos, en este caso entre actores y casos de uso. La cabeza de la flecha indica quién actúa sobre quién. Una raya

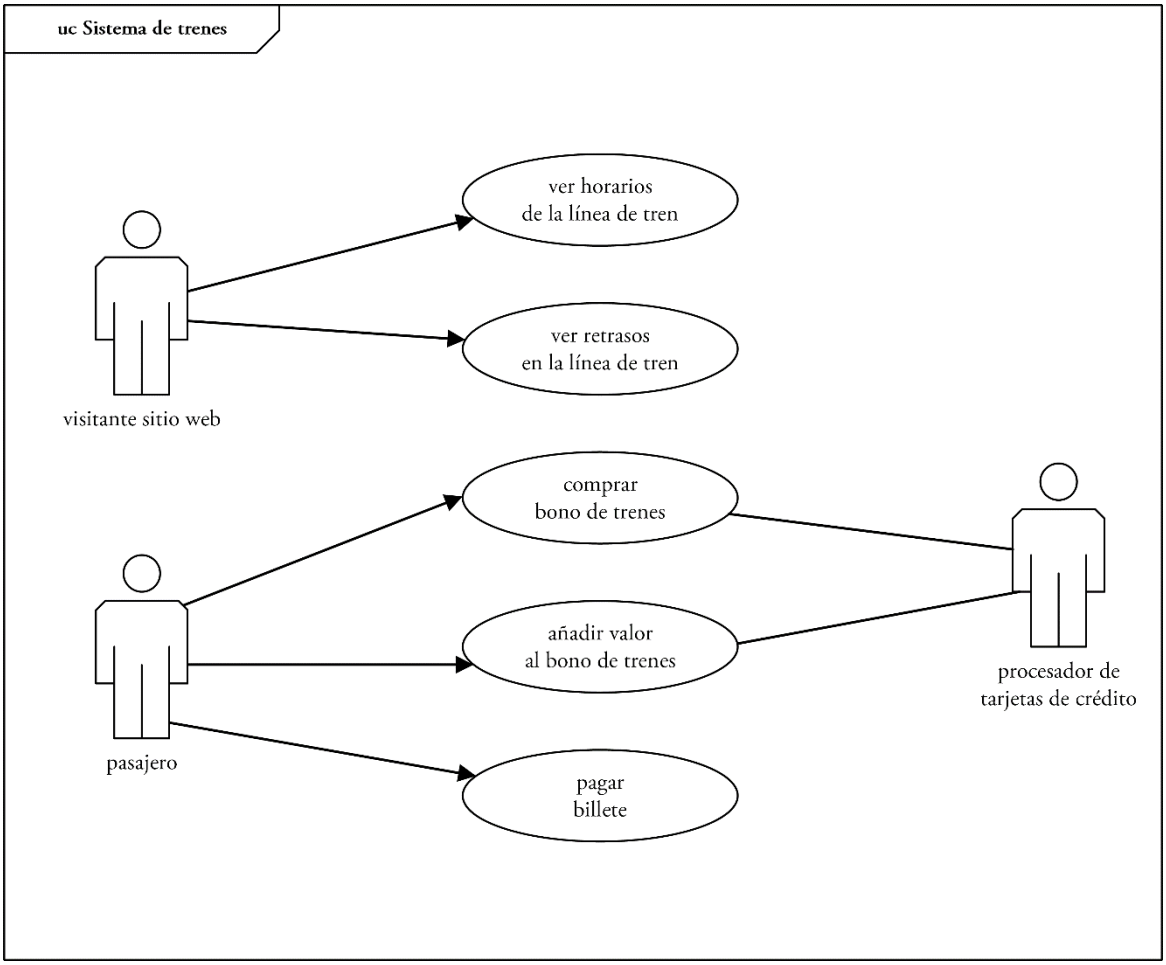

Figura 3.5 - Diagrama de casos de uso
(Daoust 2012, p. 54)

sin flechas implica que la cabeza de la flecha apunta hacia el caso de uso. En este sistema, el visitante del sitio web puede llevar a cabo dos acciones: ver los horarios del tren y ver los retrasos en las líneas. El pasajero puede hacerse con un abono, recargar el abono o comprar un billete solo. El procesador de tarjetas de crédito interactúa con las primeras dos opciones que tiene el pasajero, pero no con el caso de uso de comprar billetes. Esto quiere decir que los bonos se

pueden comprar y recargar con tarjeta de crédito, pero los billetes solo se pueden comprar con otro método de pago que no sea tarjeta de crédito.

3.2.1.3.2 Diagrama de clases

Los diagramas de clases muestran las entidades significativas para el negocio o la aplicación que se está desarrollando, las relaciones entre dichas entidades, la información que hace falta capturar en cada entidad (la que es relevante a la aplicación o negocio) y las acciones u operaciones que se ejecutan sobre dichas clases. Estos diagramas recogen información relacionada con los datos e información relacionada con los comportamientos en un mismo diagrama, con lo cual sirven para documentar aspectos de proceso y de datos a la vez. La Figura 3.6 presenta un diagrama de clases para un sistema de trenes que contiene cinco clases: tren, trayecto ferroviario, estación de tren, parada y recorrido de tren. El rectángulo que representa

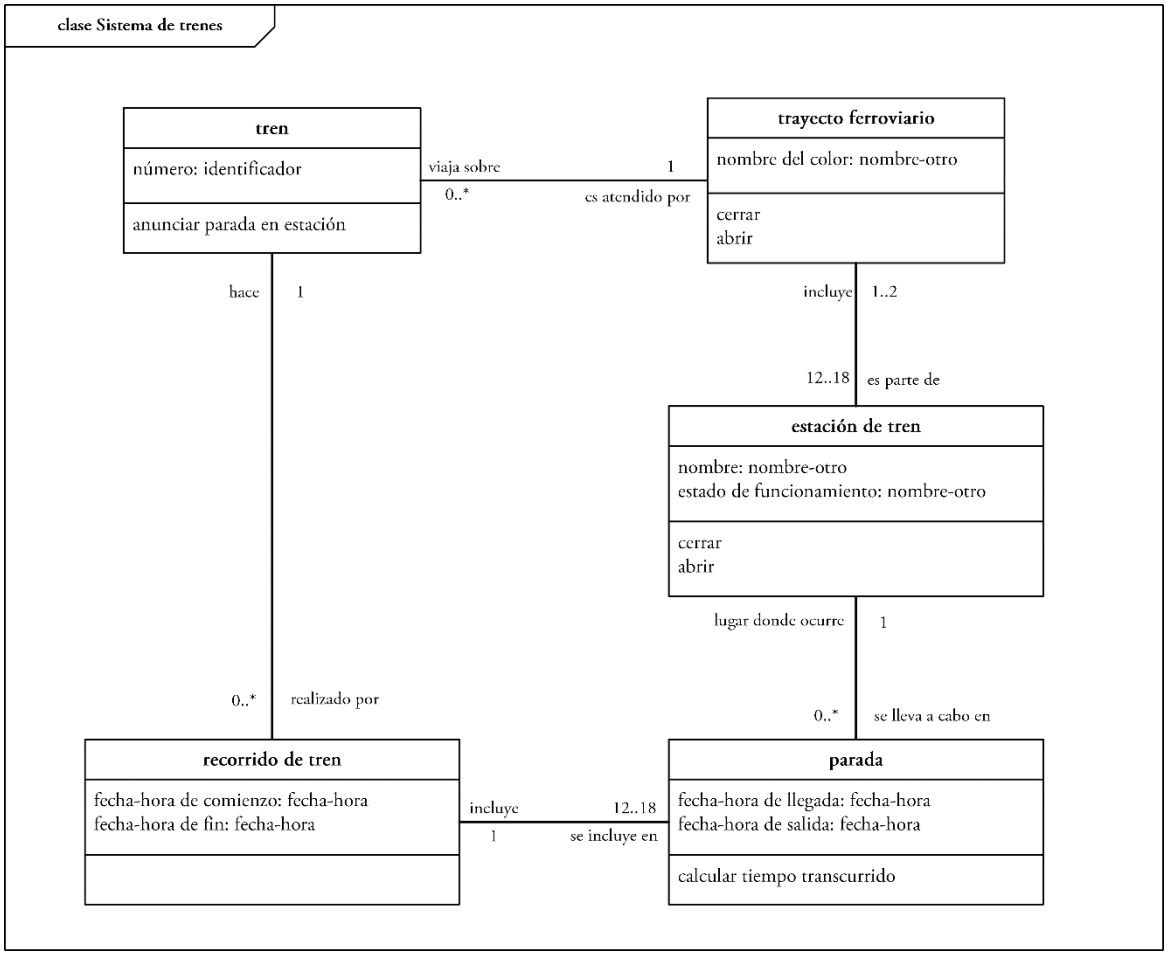

Figura 3.6 - Diagrama de clases
(Daoust 2012, p. 115)

a cada clase está dividido en tres secciones. La sección superior indica el nombre de la clase, la sección central enumera los atributos, es decir, las características que describen a la clase, y la sección inferior recoge las operaciones que se llevan a cabo sobre la clase. Las líneas que van entre clase y clase se llaman *asociaciones* y, al igual que los diagramas de entidad-relación[194], incluyen cardinalidad[195]. Veamos cómo se leen estas relaciones. En el caso de tren y trayecto ferroviario, un tren "viaja sobre" una y solo una vía (1), mientras que el trayecto ferroviario "es atendido por" entre cero y una cantidad infinita (0..*) de trenes. El trayecto ferroviario "incluye" de doce a dieciocho estaciones de tren (12..18), mientras una estación de tren "es parte de" entre uno y dos (1..2) trayectos ferroviarios. La estación de tren es el "lugar en donde ocurre" entre cero y una cantidad infinita de paradas; una parada, sin embargo, ocurre exactamente en una sola estación de tren, y así las restantes. Los atributos incluyen el nombre del atributo y su característica física (identificador o clave única, nombre-otro o campo alfanumérico, fecha-hora o campo que incluye la fecha y la hora en este ejemplo), y las operaciones describen qué acción se puede ejecutar sobre la clase. El trayecto y la estación se pueden abrir y cerrar; sabemos qué acción se puede ejecutar, pero para este fin específico no necesitamos saber cómo se lleva a cabo.

[194] Véase apartado "Diagrama de entidad-relación" en la página 159.

[195] Véase cardinalidad en la página 159.

3.2.1.3.3 Diagrama de actividades

El diagrama de actividades muestra la manera en que fluyen las acciones y las decisiones de un sistema y quiénes llevan a cabo dichas acciones o quiénes toman las decisiones (*Ibidem*, p. 55). La Figura 3.7 nos muestra el diagrama de actividades para el tercer caso de uso que se ilustra en la Figura 3.5, "comprar bono de trenes". Las actividades que llevan a cabo los tres

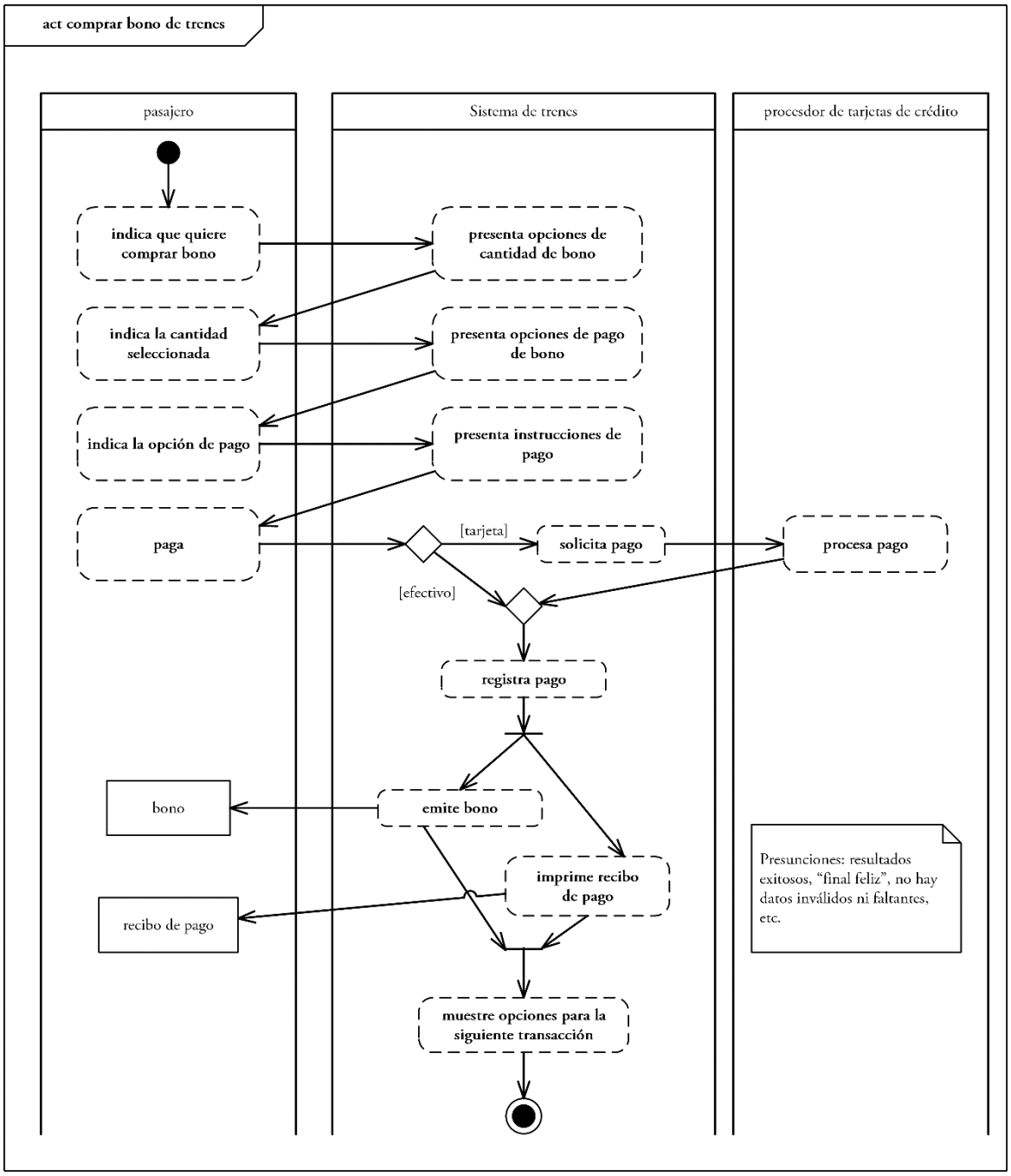

Figura 3.7 - Diagrama de actividades
(Daoust 2012, p. 77)

actores del sistema se recogen en los tres cajones verticales con la parte inferior abierta. El diagrama se empieza a leer a partir del punto sólido con un círculo de arriba llamado *nodo inicial* y termina en el nodo final, un punto sólido con dos círculos alrededor. Los rectángulos con los bordes redondeados son las acciones que se llevan a cabo y los que tienen las esquinas rectas (bono y recibo de pago) son objetos que se generan. Las flechas regulan y dirigen el flujo entre acciones y objetos, y se dividen o se unen en los nodos de decisión o de unión, respectivamente, que tienen forma de diamante. La barra horizontal donde se dividen o se unen flechas son nodos horquilla e indican acciones que ocurren a la vez, como es el caso de la emisión del bono y la impresión del recibo de pago. El rectángulo con la esquina doblada corresponde a una anotación o comentario, que, en el caso de este diagrama, nos indica que este es el escenario óptimo del proceso, donde no ocurren errores ni faltan datos para procesar la transacción.

3.2.1.3.4 Diagrama de máquina de estados

Los diagramas de máquina de estados[196] se usan para modelar el ciclo de vida de un objeto y muestra los cambios en estado que sufre ese objeto, qué los ocasiona y cuáles son las

[196] También se les llama diagramas de estado o mapas de estado (*statechart*).

restricciones para cambiar de un estado a otro. Por lo regular, estos cambios se rigen por las reglas del negocio o de la aplicación (*Ibidem*, p. 125). La Figura 3.8 muestra un diagrama de máquina de estados de un semáforo para peatones. Al igual que el diagrama de actividades, comienza con un nodo inicial y termina con un nodo final. Una vez el semáforo se instala, está inactivo, pero al ser sometido a una transición de estado ("activar"), su estado cambia a activo. Este estado de activo se conoce como un superestado, pues está compuesto por dos estados simples. En el estado activo el semáforo transiciona entre estos dos estados simples, indicar "caminar" e indicar "no caminar". Obsérvese que es imposible desactivar el semáforo cuando está indicando "caminar", pues ocasionaría una situación de peligro. Las gafas en la esquina derecha inferior del superestado indican que es un estado compuesto, es decir, que el diagrama indica ciertos estados pero que en realidad es más complejo de lo que se indica. En el caso de este ejemplo, el semáforo para peatones trabaja en concierto con el semáforo para automóviles y sus cambios de estados están relacionados y sincronizados de una manera específica, pero esa sincronía, estado y relaciones no se ven en este diagrama (posiblemente se vean en otro). Solo cuando el semáforo retorna al estado de "inactivo" es que se puede desinstalar.

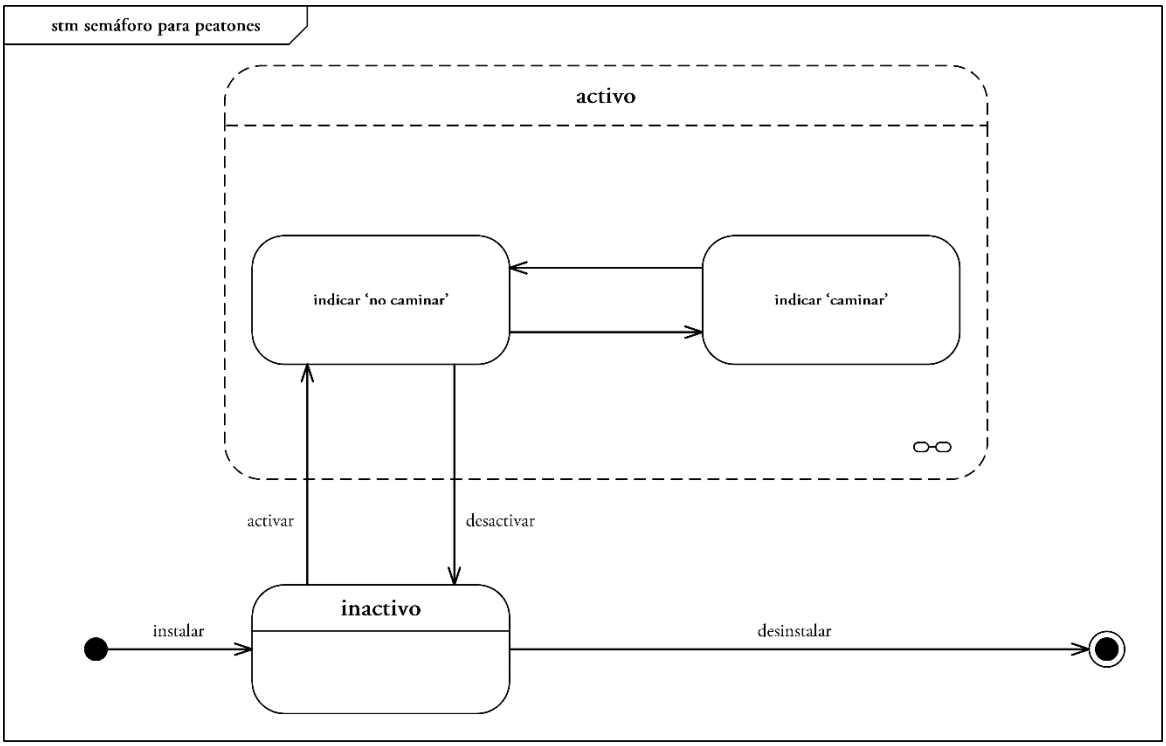

Figura 3.8 - Diagrama de máquina de estados
(Daoust 2012, p. 131)

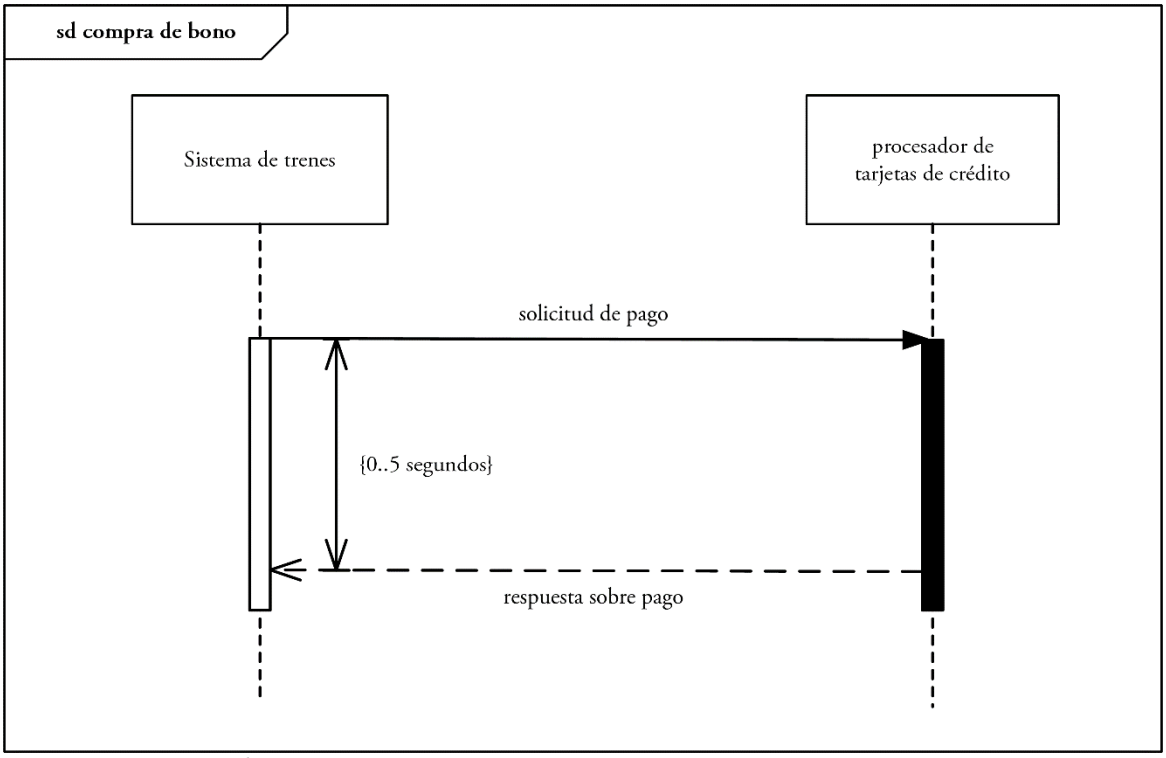

Figura 3.9 - Diagrama de secuencias
(Daoust 2012, p. 159)

3.2.1.3.5 Diagrama de secuencias

Los diagramas de secuencias ilustran la secuencia de intercambio de información entre los actores en un espacio de tiempo. Los diseñadores de *software* a menudo usan este tipo de diagrama para especificar el intercambio de mensajes entre objetos al momento de ejecución de un programa. La Figura 3.9 nos muestra un escenario de un caso de uso en específico: la interacción entre el sistema de trenes y el procesador de tarjetas de crédito que vimos primero en la Figura 3.5 y luego expandimos en la Figura 3.7. En este ejemplo, el sistema de trenes envía un mensaje, "solicitud de pago", al procesador de tarjetas de crédito, que, transcurridos entre cero y cinco segundos, devuelve el mensaje "respuesta sobre pago" al sistema de trenes.

3.2.2 Codificación de caracteres, Unicode y bibliotecas relacionadas

ASCII, siglas de *American Standard Code for Information Interchange*, fue uno de los primeros sistemas de codificación de caracteres[197] que adoptaron los manufactureros de computadoras y establece una relación entre un carácter y un código numérico cuya representación en formato binario permite que la computadora lo despliegue o ejecute operaciones sobre él. Como fue creado en los Estados Unidos, solo considera el conjunto de caracteres necesarios para escribir en inglés estadounidense, dejando fuera tildes, cedillas, acentos y muchos otros tipos de signos diacríticos (sin mencionar ideogramas) necesarios para escribir en otros idiomas. Para resolver este problema se fueron creando distintas páginas de códigos para cada idioma o región que permitían, aunque para ciertos idiomas de manera limitada, ampliar los espacios de la tabla original (que solo permitía 128 caracteres). Estas páginas de códigos no solo se encargaban de definir la equivalencia entre un código binario del repertorio de caracteres de la computadora y su correspondiente ítem textual, sino que también permitían definir la ordenación de estos caracteres y la conversión entre mayúsculas y minúsculas. Vemos que el uso de esta colección persiste en la actualidad:

> Because coded character sets evolved along with the technological advances in the computer hardware, software, and communications, there are many of them. The most recent and popular coded character set is Unicode, which contains all of the characters needed to write all of the languages in current use, and many other languages as well. While Unicode is recommended for all text representation, especially in today's on-demand world, several older character

[197] La codificación de caracteres la tratamos un poco más adelante, en la página 172.

> sets are still used to represent data in databases, Web page collections, or user interfaces.
>
> (International Business Machines 2013b, sec. Executive Overview)

Al llegar la era de las comunicaciones este sistema presentaba problemas graves al momento de intercambiar datos, pues había que saber de antemano qué página de códigos se estaba usando para codificar los ficheros que se intercambiaban. Estas tablas funcionan perfectamente dentro del entorno donde se usan: sus usuarios ven los caracteres esperados y tienen disponibles los que necesitan para su *locale*. Sin embargo, al momento de intercambiar datos, los puntos (o valores binarios internos) que definen los caracteres no necesariamente coinciden. Por ejemplo, el punto que ocupa el símbolo "#" en la tabla de caracteres ASCII lo ocupa el de "£" en la tabla del ISO 7-bit del Reino Unido. El punto que corresponde a "Ñ" en la ISO 7-bit de España lo ocupa "]" en ASCII. Para evitar que los datos se corrompan al intercambiarlos o se presenten incorrectamente al usuario, los manufactureros y consorcios de la industria han desarrollado subrutinas[198] de conversión entre los esquemas de codificación de caracteres que se encargan de mantener la integridad de los datos al mover información entre sistemas. También indican estados de error cuando no es posible hacer dicha conversión (porque el carácter no existe dentro del repertorio de uno de los sistemas entre los que se convierte) o proveen servicios de encapsulamiento para proteger los datos no transformables (porque no hay equivalencia directa, por ejemplo, entre caracteres orientales y occidentales). Los *International Components for Unicode* (ICU), una biblioteca que incluye servicios extendidos de Unicode y de manipulación de textos que veremos más adelante en la página 184, también brindan subrutinas de conversión entre esquemas de codificación de caracteres.

La solución a la multiplicidad de páginas de códigos y los problemas que ello conlleva es Unicode, un sistema de codificación estandarizado desarrollado a finales de la década de 1980 que provee espacio para codificar al menos 1 114 112 *code points* o posiciones de código, y permite expansión, de ser necesaria. El sistema de codificación Unicode

[198] Una subrutina (también llamada procedimiento, función, rutina, método o subprograma) es un módulo de código independiente que sirve para llevar a cabo una tarea en concreto (Janssen 2016, s. v. procedure).

> "[…] is a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages and technical disciplines of the modern world. In addition, it supports classical and historical texts of many written languages".
>
> (Unicode Consortium 2013)

La diferencia clave de este sistema es que dichas posiciones de código asignan una representación numérica en una tabla a un concepto y no a un carácter, por tanto, el concepto de una "Latin Small Letter a with Ring Above and Acute" corresponde a la posición número 507 de la tabla de Unicode o a el código U+01FB (en hexadecimal) y se ve así: ǻ. Esta es una definición estática y fija, un cajón que guarda un concepto único asociado a un carácter, y se podrá ver si se usa una fuente que incluya en su repertorio el carácter correspondiente al código U+01FB[199].

Al hablar de la codificación de caracteres es importante distinguir entre los conjuntos de caracteres (*character set*), los conjuntos de caracteres codificados (*coded character set*) y los esquemas de codificación (*encoding scheme*). Los conjuntos de caracteres son el repertorio de símbolos gráficos que se usan para un fin en específico, como, por ejemplo, el abecedario en español.

> "A *character set* contains the letters, numbers, symbols, and special characters used to represent data in a particular locality or line of business. A given character set can be useful for many environments, such as different countries, different languages, or different lines of business. For example, the LATIN1

[199] Para más información sobre cómo se desarrollaron los sistemas de codificación de caracteres véase Searle (2004). Wideburg (1998) recoge datos interesantes sobre los inicios del desarrollo del Unicode. Spolsky (2003) y Zentgraf (2015) explican la importancia de entender el impacto de los sistemas de codificación a la hora de programar.

> character set contains the characters used for business applications in most Western European countries"
>
> (Unisys 2013c, p. 1.6).

Un conjunto de caracteres codificados representa un conjunto de caracteres a los cuales se les ha asignado un número único (*code point* o posición de código) que identifica a cada uno de los miembros del conjunto (también se les llama páginas de códigos).

> "A *coded character set* is a character set in which each character is assigned a code value. The same character set can exist with a variety of encodings. For example, the LATIN1-based character set can be encoded in an International Organization for Standardization (ISO) format or an EBCDIC format. Although the system uses EBCDIC codes, it also recognizes other format codes and can map characters from one coded character set to another with a coded character set mapping table.
>
> (*Idem*)

Los esquemas de codificación especifican la manera en que ese número único se representa internamente en una computadora. Unicode usa tres esquemas de codificación: UTF-8, UTF-16 y UTF-32 (Ishida 2010). Los demás conjuntos de caracteres (ASCII, por ejemplo) tienen esquemas de codificación propios[200].

Para desarrollar aplicaciones internacionalizadas, las plataformas de desarrollo y los lenguajes de programación que se utilicen tienen admitir Unicode, que no solo es el estándar internacional que busca digitalizar todos los sistemas de representación gráfica de idiomas actuales, pasados y futuros, sino que, por medio de su gestor, el Unicode Consortium, también está a cargo de mantener el repositorio de datos de localización llamado CLDR y los ICU, que veremos enseguida. Estos dos recursos son fundamentales en la expansión de las funcionalidades de apoyo a la internacionalización y el mantenimiento uniforme de datos relacionados con los *locales* que sirven a los desarrolladores de *software*. Para la industria de videojuegos que, como dijéramos arriba, es más consciente de que sus productos sean

[200] Ishida (2010) ofrece una explicación más detallada.

prolocalizables, esto es muy importante. Chandler y Deming (2012) recomiendan siempre mantenerse al día e implementar la última versión de Unicode en su *software*: "[…] Implementing the latest version of Unicode in the game engine allows for easy localization of all languages and helps pave the way for expansion into new territories […]" (*Ibidem*, p. 111). Un producto que se va a usar en una diversidad de mercados con *locales* variados no puede darse el lujo de no poder admitir el repertorio de caracteres que Unicode pone a disposición del desarrollador.

3.2.2.1 Una fuente común de datos relacionados con la localización

El 16 de enero de 2004 el comité de dirección de Openi18n, cuyos miembros fundadores fueron IBM, Sun y OpenOffice.org, aprobó la versión 1.0 del CLDR, siglas de *Common Localization Data Repository* (Unicode Consortium 2015a). En esos momentos:

> Most operating systems and many application programs currently maintain their own repositories of locale data to support [… language or region conventions on the formatting of dates, numbers, time, currency, etc., … b]ut such data are often incomplete, idiosyncratic, or gratuitously different from program to program. In the age of the internet, software components must work together seamlessly, without the problems caused by these discrepancies.
>
> (Unicode Consortium 2005)

Con el objetivo de subsanar estos problemas crean el CLDR, y así brindar "[…] a general XML format for the exchange of locale information for use in application and system development, and to gather, store, and make available a common set of locale data generated in that format" (*Idem*). Este repositorio central, que a partir de su versión 1.1 está bajo el control del Unicode Consortium (Unicode Consortium 2015a), contiene:

- Patrones específicos para cada *locale* para el formato de
  - fechas, horas y husos horarios, incluidos calendarios no-gregorianos;
  - números y moneda, incluidos números que no son ASCII.
- Traducción de los nombres de:
  - idiomas, países, regiones establecidas por la ONU, alfabetos y escritura;

  - símbolos y nombres de monedas, incluidas las correspondientes formas plurales;
  - los días de la semana, meses, eras, períodos de días, todos en formatos cortos, abreviados y sin abreviar;
  - los husos horarios, ciudades que corresponden a ciertos husos horarios; y
  - las unidades de tiempo en formato corto y largo, en presente, pasado y futuro, incluidas las correspondientes formas plurales.
- Información sobre los idiomas, alfabetos y escritura:
  - caracteres que usa el idioma, más caracteres especiales y una lista de los caracteres que conforman un índice[201];
  - formación de plurales de sustantivos o expresiones unitarias y ordinales;
  - asignación del género de una lista[202];
  - reglas sobre mayúsculas y minúsculas;
  - reglas de ordenación y para hacer búsquedas en texto;
  - información sobre el alfabeto o escritura, por ejemplo, direccionalidad del idioma;
  - reglas de transliteración entre alfabetos o escritura e idiomas;
  - reglas para deletrear cifras; y
  - reglas sobre cómo segmentar el texto en caracteres, palabras, líneas y oraciones.
- Información sobre el país:
  - uso de idiomas, alfabetos y escritura en los territorios, incluidas lenguas oficiales;
  - posibles países para un idioma y posibles idiomas para un país;
  - moneda, número de decimales según la moneda, redondeo de la moneda y estado histórico de la moneda (nombres según las fechas de uso);

[201] *Index characters* en inglés. "Index characters are an ordered list characters for use as a UI [*user interface*] 'index', that is, a list of clickable characters (or character sequences) that allow the user to see a segment of a larger "target" list" (Unicode Consortium 2009).

[202] Para determinar el género de una lista de nombres. Véase http://unicode.org/reports/tr35/tr35-general.html#List_Gender

  - preferencias del calendario, rangos de las eras, comienzo de la semana; y
  - códigos de teléfono.
- Otros:
  - códigos ISO[203] para países e idiomas;
  - organización de territorios y regiones según la ONU[204];
  - extensiones del BCP 47[205];
  - mapas de teclado para idiomas en distintas plataformas; y
  - sistemas de numeración tradicionales, como por ejemplo, números romanos (Unicode Consortium 2015b).

Usando un repositorio de datos, las principales empresas informáticas, como Apple, Google e IBM, o que se sirven de la informática[206] se aseguran de brindar mayor uniformidad al

[203] Se refiere a la norma ISO 3166 que contiene los códigos de dos y tres letras para países.

[204] M49; véase http://unstats.un.org/unsd/methods/m49/m49regin.htm

[205] El Grupo Especial de Ingeniería de Internet (IETF, por sus siglas en inglés) se encarga de gestionar las RFC. Las RFC o *peticiones de comentarios* son los documentos que rigen las especificaciones de los protocolos, procedimientos, funcionamiento y todo aspecto que tiene que ver con cómo opera la internet. Este es el mecanismo que se usa para crear los estándares oficiales de la red. Como el IETF no tiene otra manera de endosar información técnica relevante, creó las BCP o *mejor práctica actual* (Postel, Rekhter y Li 1995). La BPC 47 especifica las normas a seguir al acompañar el atributo de idioma con etiquetas informativas adicionales. "Sometimes language tags are used to indicate additional language attributes of content. For example, indicating specific information about the dialect, writing system, or orthography used in a document or resource may enable the user to obtain information in a form that they can understand, or it can be important in processing or rendering the given content into an appropriate form or style. This document specifies a particular identifier mechanism (the language tag) and a registration function for values to be used to form tags. It also defines a mechanism for private use values and future extensions" (Phillips y Davis 2009).

[206] El sitio web del CLDR indica que Apple (OS X, iOS, aparatos móviles de Apple en iTunes para Windows, entre otros), Google (búsquedas, Chrome, Android, Adwords, Google+, Google Maps, Blogger, Google Analytics, entre otros), IBM (DB2, Lotus, Websphere, Tivoli, Rational, AIX, i/OS, z/OS, entre otros) y Microsoft (Windows, Windows Phone, Bing, Office, Visual Studio, entre otros) se sirven de este repositorio. También mencionan a las siguientes empresas: ABAS Software, Adobe, Amazon (Kindle), Amdocs, Apache, Appian, Argonne National Laboratory, Avaya, Babel (Pocoo library), BAE Systems Geospatial eXploitation

momento de presentar ciertos datos que son sensibles a los *locales* y que se usan con frecuencia a partir una fuente centralizada, uniforme y universal. No solo eso, sino que también facilita la labor de los programadores porque "we don't have to do as much research, we don't have to find out specifically how each thing should be formatted in these different languages and these different countries, instead we can just follow the standard […]" (Patch 2014), es decir, que con tan solo seguir el estándar del CLDR, el programador tiene la posibilidad de dar formato correcto y aplicar toda la información relacionada con cada *locale* que el repositorio contiene.

Pero no solo el programador se beneficia de esta base de datos. A partir del CLDR se han desarrollado proyectos que se valen de este repositorio para dar mejor apoyo de internacionalización a los lenguajes de programación que carecen de este tipo de soporte. La *Closure Library* de Google[207] es una biblioteca modular y multinavegador de JavaScript que lo usa, al igual que la biblioteca de internacionalización de Ruby, TwitterCLDR, que a partir de 2012 forma parte de los proyectos de código abierto en GitHub[208].

Products, BEA, la Biblioteca del Congreso de los EE.UU., BluePhoenix Solutions, BMC Software, Boost, BroadJump, Business Objects, caris, CERN, Debian Linux, Dell, Eclipse, eBay, EMC Corporation, ESRI, Firebird RDBMS, Free BSD, Gentoo Linux, GroundWork Open Source, GTK+, Harman/Becker Automotive Systems GmbH, HP, Hyperion, Inktomi, Innodata Isogen, Informatica, Intel, Interlogics, IONA, IXOS, Jikes, jQuery, Mathworks, Mozilla, Netezza, OpenOffice, Oracle (Solaris, Java), Lawson Software, Leica Geosystems GIS & Mapping LLC, Mandrake Linux, OCLC, Perl, Progress Software, Python, QNX, Rogue Wave, SAP, Shutterstock, SIL, SPSS, Software AG, SuSE, Symantec, Teradata (NCR), ToolAware, Trend Micro, Twitter, Virage, webMethods, Wikimedia Foundation (Wikipedia), Wine, WMS Gaming, XyEnterprise, Yahoo!, Yelp (Unicode Consortium 2015c).

[207] En https://developers.google.com/closure/.

[208] Véase (Aniszczyk 2012).

3.2.2.2 *International Components for Unicode*

Los *International Components for Unicode* (ICU) son bibliotecas desarrolladas para C, C++ y Java[209] que brindan servicios extendidos de Unicode y de manipulación de textos. Aunque estos lenguajes proveen cierto apoyo para la internacionalización en su configuración base, las funcionalidades son limitadas. Estas bibliotecas proveen los siguientes servicios:

- Conversión entre páginas de códigos y Unicode[210];
- Ordenación y comparación de cadenas de texto, según las convenciones y estándares de un idioma en particular a partir del CLDR y las reglas de ordenación de Unicode;
- Formato de números, fechas, hora y moneda, incluidas la traducción de meses y días, sus abreviaturas y ordenación, a partir de los datos del CLDR;
- Cálculos de tiempo y fechas que toman en consideración los husos horarios y los distintos tipos de calendarios;
- Apoyo al estándar de Unicode en cuanto a las propiedades de los caracteres y operaciones relacionadas con estos;
- Expresiones regulares[211] con apoyo a Unicode;
- Apoyo a lenguajes bidi; y
- Ubicación de los puntos que delimitan palabras, oraciones y párrafos, y los lugares en donde es apropiado cambiar de renglón al desplegar texto, entre otros.

La licencia de los ICU es de código abierto no restrictiva, con lo cual se puede usar para desarrollar programas comerciales, gratuitos y de código abierto. En la página de los ICU hay una lista de todas las empresas que se sirven de esta biblioteca de subrutinas (Unicode Consortium 2016).

[209] Las bibliotecas se conocen como ICU4C para C y C++, e ICU4J para Java.

[210] Para la relación entre páginas de código y Unicode véase Sherer (2000).

[211] Una expresión regular o *regex* es una cadena de texto especial que se usa para describir un patrón de búsqueda, una suerte de buscador con comodines. No solo sirven para hacer búsqueda y reemplazo, sino que también sirven para validar campos (Goyvaerts 2015).

3.2.3 Lenguajes de programación - descripción y soporte a la internacionalización

Esta breve descripción de los lenguajes y de las prestaciones de internacionalización que tienen los lenguajes será útil para el internacionalizador a la hora de interactuar con los desarrolladores de *software* y les servirá de pie de apoyo al momento de comenzar a trabajar en un proyecto de desarrollo de *software* con capacidad de ser internacionalizado. Si, en el caso de un lenguaje de programación, se menciona una plataforma en específico y otra no se menciona, significa que al momento de la investigación no se encontró apoyo a la internacionalización para la plataforma no mencionada. Si no se menciona ninguna plataforma en específico, la funcionalidad encontrada debería servir para cualquier implementación del compilador. Los entornos de desarrollo integrados, es decir, las herramientas en donde se codifican algunos de estos lenguajes de programación se reseñan aparte en el Apéndice A.

Para elaborar esta lista de lenguajes de programación, tomamos como punto de partida el árbol genealógico de los lenguajes de programación desarrollado por Lévénez (2015), descartamos los lenguajes que funcionan sobre los servidores web (como por ejemplo, Perl, PHP, Ruby, etc.) y nos aseguramos de que estuvieran los principales lenguajes que se usan para desarrollar aplicaciones sobre las plataformas *mainframe*, *midrange*, microcomputadoras y servidores. El índice TIOBE, que prepara un ranking mensual de los lenguajes de programación más mencionados en la internet, recoge todos los lenguajes que discutimos a continuación, con excepción de ALGOL y Pascal, entre su *top 100*[212]. El resto, salvo OCaml, PL/1, Smalltalk y Tcl/Tk, figura entre los *top 50* (TIOBE Software 2016). La Tabla 3.1 enumera, en orden alfabético, los principales lenguajes de programación disponibles para la plataforma *mainframe*, según los distintos sistemas operativos que funcionan sobre cada una, mientras que la Tabla 3.2 hace lo mismo para las plataformas *midrange*, microcomputadoras y servidores.

[212] El índice de popularidad de cada lenguaje se calcula todos los meses haciendo búsquedas automatizadas a través de la web del *query* '+"<lenguaje> programming"', donde <lenguaje> corresponde a la lista de lenguajes de programación calificados (véase http://www.tiobe.com/tiobe_index?page=programminglanguages_definition para más información).

Lenguaje ↓	*Mainframes*						
	IBM			Fujitsu		Unisys	
Sistema operativo→	zOS	Linux[213]	zVSE	OSIV/ XPS	MSP	Clearpath OS 2200	Clearpath MCP
Ada		x					
ALGOL[214]		x					x
Ensamblador	x	x	x				
C	x	x	x	x	x	x	x
C++	x	x					
C#[215]		x					
COBOL	x	x	x	x	x	x	x
Common Lisp		x					

[213] Distribución Redhat, Suse y Ubuntu.

[214] Para Linux, Algol 68 Genie Project en http://jmvdveer.home.xs4all.nl/algol.html.

[215] Bajo el entorno de desarrollo integrado MonoDevelop se pueden desarrollar aplicaciones en Linux y OS X o portar (convertir) aplicaciones de .NET a Linux y a Mac. Este entorno permite desarrollar aplicaciones en C#, F#, Visual Basic .NET, C/C++ y Vala (http://www.monodevelop.com/). .NET Core (https://dotnet.github.io/) es una implementación multiplataforma de .NET de código abierto lanzada en noviembre de 2015 (Lander 2015) que permite desarrollar aplicaciones sobre Linux bajo el *framework* de .NET.

Lenguaje ↓	*Mainframes*						
	IBM			Fujitsu		Unisys	
Sistema operativo→	zOS	Linux[213]	zVSE	OSIV/ XPS	MSP	Clearpath OS 2200	Clearpath MCP
Delphi		x					
Fortran	x	x		x	x	x	x
Haskell		x					
Java EE[216]	x					x	x
Java ME							
Java SE[217]		x					x
Objective-C							
OCaml		x					
Pascal		x					x
Perl		x					
PHP		x					

[216] J2EE (Eclipse IDE) para Clearpath OS 2200 y JEE7 para Clearpath MCP.

[217] Java SE versiones 7 y 8.

Lenguaje ↓	*Mainframes*						
	IBM			Fujitsu		Unisys	
Sistema operativo→	zOS	Linux[213]	zVSE	OSIV/ XPS	MSP	Clearpath OS 2200	Clearpath MCP
PL/1	x		x	x	x		
RPG			x				x
Smalltalk		x					
Swift		x					
Tcl/Tk		x					
Visual Basic .NET		x					

Tabla 3.1 - Lenguajes de programación principales disponibles para las plataformas mainframe

Lenguaje ↓	*Midrange*			Microcomputadoras y servidores		
	IBM				PC	Mac
Sistema operativo→	Linux[218]	AIX[219]	IBMi	Linux	Windows	OS X
Ada	x			x	x	x
ALGOL[220]	x			x	x	x
Ensamblador	x			x	x	x
C	x	x	x	x	x	x
C++	x	x	x	x	x	x
C#[221]	x			x	x	x
COBOL	x	x	x	x	x	x
Common Lisp	x			x	x	x

[218] Distribución Redhat, Suse y Ubuntu.

[219] AIX es la distribución de IBM de Unix.

[220] Para OS X, Algol68 en http://algol68.sourceforge.net, para los demás sistemas operativos, Algol 68 Genie Project en http://jmvdveer.home.xs4all.nl/algol.html.

[221] El IDE MonoDevelop permite desarrollar aplicaciones en Linux y OS X o portar (convertir) aplicaciones de .NET a Linux y a Mac. También se puede usar .NET Core en estas plataformas. Véase nota 215. Para la plataforma Windows se usa Visual Studio, pero también se puede usar .NET Core.

Lenguaje ↓	*Midrange*			Microcomputadoras y servidores		
	IBM				PC	Mac
Sistema operativo⟶	Linux[218]	AIX[219]	IBMi	Linux	Windows	OS X
Delphi[222]	x!			x!	x	x
Fortran	x	x		x	x	x
Haskell	x			x	x	x
Java EE			x			
Java ME						
Java SE	x			x	x	x
Objective-C						x
OCaml	x			x	x	x
Pascal	x			x	x	x
PL/1	x	x		x		

[222] x! - El IDE de Embarcadero (https://www.embarcadero.com) permite desarrollar aplicaciones en Delphi y C++ sobre las plataformas Windows y OS X. Según Aasenden esta plataforma pronto estará disponible también para Linux (Aasenden 2015).

Lenguaje ↓	*Midrange*			Microcomputadoras y servidores		
	IBM				PC	Mac
Sistema operativo→	Linux[218]	AIX[219]	IBMi	Linux	Windows	OS X
RPG[223]	x!!	x!!	x	x!!	x!!	
Smalltalk	x			x	x	x
Swift[224]	x			x		x
Tcl/Tk	x	x		x	x	x
Visual Basic .NET[225]	x			x	x	x

Tabla 3.2 - Lenguajes de programación principales disponibles para las plataformas midrange, servidores y computadoras personales

Roturier nos recuerda que "[… l]ike natural languages, it is obviously impossible for any individual to master all of them [programming languages], but it is possible to learn key concepts and be able to get some understanding of what the code is supposed to achieve […]" (2015, p. 19). Bajo esta premisa y sin pretender cubrir todo todos los lenguajes de programación existentes ni todas sus prestaciones, presentamos esta selección de lenguajes de

[223] x!! - Infinite i (http://www.infinitecorporation.com/infinite-i) es un IDE que permite compilar *software* desarrollado en RPG para la plataforma IBMi y convertirlo en código nativo que ejecute sobre Red Hat Linux, Windows Enterprise Server, HP-UX for Ithanium, AIX y Solaris. También funciona para COBOL.

[224] Para Linux en Ubuntu.

[225] Véase nota 221.

programación, hablamos brevemente de sus aplicaciones en el mundo práctico y describimos las prestaciones de internacionalización que ofrecen, si alguna.

3.2.3.1 Ada

Ada[226] fue diseñado para crear aplicaciones de gran tamaño y duraderas, y se usa particularmente en sistemas embebidos[227], en especial en aplicaciones donde la fiabilidad y la eficiencia son elementos fundamentales. Por ello se utiliza a nivel mundial en aplicaciones de aviónica, control de tráfico aéreo, sistemas ferroviarios y aparatos médicos. Es un lenguaje que posee características para funcionar como lenguaje orientado a objetos, pero no es necesario usar estas características («Ada Overview» 2016). Según el cronograma y árbol genealógico que desarrolló Lévénez (2015), es un lenguaje que deriva de Pascal, y el historial de versiones es Ada 79, Ada 83 ANSI, Ada ISO (1987), Ada 95, Ada 2005 y la más reciente, Ada 2012.

Ada carece de funciones avanzadas para manejar la internacionalización, sin embargo, el componente *League* del *framework* Matreshka[228] ofrece funcionalidades básicas para internacionalizar aplicaciones desarrolladas en Ada: conversión de variables de texto (incluye esquemas de codificación de caracteres), cursores[229], conversión entre mayúsculas y minúsculas, ordenamiento, cálculos de fechas, funciones para la localización de cadenas de texto, entre otras.

[226] El nombre de este lenguaje no es un acrónimo, sino un homenaje a Augusta Ada Lovelace, la hija del poeta Lord Byron, matemática y considerada como la primera programadora por su trabajo junto con Charles Babbage («Ada Overview» 2016).

[227] Véase nota 174.

[228] En http://forge.ada-ru.org/matreshka/wiki/League.

[229] Los cursores se usan para moverse a lo largo de los elementos que componen una variable con texto. El movimiento entre caracteres varía según la codificación de caracteres y el idioma, por eso la necesidad de este tipo de funcionalidad. Permite desplazamiento entre caracteres o conjuntos de grafemas. Los conjuntos de grafemas son caracteres que, para el usuario, se visualizan como un solo carácter pero que en realidad su representación interna comprende varios caracteres. Los emoticones son un ejemplo de esto (Véase SJ 2015).

Convierte representaciones internas y externas desde y hacia varios esquemas de codificación de caracteres.

3.2.3.2 ALGOL

Su nombre se deriva de ALGOrithmic Language y, similar al FORTRAN, se distingue por ser un lenguaje de tipo algebraico diseñado específicamente para resolver problemas numéricos (Gries 1978). De hecho, según Lévénez (2015) es derivado, por conducto de IAL[230], de FORTRAN I. Aunque existen compiladores para Linux, Mac y PC, su uso en nuestros días se debe a que este fue uno de los principales lenguajes usados en los *mainframes* de alta gama de Burroughs, una de las empresas precursoras de Unisys. En la actualidad Unisys ofrece este compilador en sus sistemas Clearpath con sistema operativo MCP.

El manual de referencia para la programación en ALGOL de la Unisys (2013a) dedica una sección extensa para explicar las prestaciones de internacionalización de su compilador, que las proporciona la biblioteca CENTRALSUPPORT del entorno MLS (*Multi Language System*) de la Unisys. Este entorno es la base para internacionalizar aplicaciones en los sistemas de Unisys y permite:

> Translate online screens, forms, menus, help text, and messages [… ;d]efine standard formats for the presentation of dates, times, numbers, and monetary data based on the needs of the application system user [… ;i]nstall and run translated application systems [… ;d]isplay text using the language, characters, symbols, and formatting relevant to the individual users of your application system [… ;p]resent the application system in several languages or conventions at the same time on the same system [… ;d]ynamically change the language or convention specifications so that users can access the application system using the language and formatting most useful to them[… ;m]ake changes to a

[230] *International Algebraic Language*, el nombre original de ALGOL 58 en el informe preliminar sobre el lenguaje (Kinnersley [sin fecha], s. v. IAL).

> previously translated application system and then translate only those changes, without retranslating the entire user interface.
>
> (Unisys 2013c, p. 1.4)

Por su parte, la biblioteca CENTRALSUPPORT incluye procedimientos para que el programa pueda:

> […d]etermine the default settings on the host computer […; i]dentify and validate character sets[231] and ccsversions[232] […; o]btain information about a coded character set or a ccsversion […; c]heck data against ccsversion data classes […; t]ranslate data using ccsversion mapping tables […; c]ompare data […; m]anipulate text […; t]ranslate characters using ccsversion escapement […; o]btain information about the convention contents […; a]dd, modify and delete conventions […; f]ormat date and time […; f]ormat monetary and numeric data […; o]btain default characteristics for hard copy output.
>
> (Unisys 2013a, p. 10.12-10.13)

Así, esta biblioteca permite internacionalizar completamente las aplicaciones desarrolladas en ALGOL sobre esta plataforma[233].

[231] Véase nota 197.

[232] "A *ccsversion* designates the way a coded character [véase nota 197] set is processed to meet the needs of a particular natural language, culture, or line of business. The ccsversion includes rules concerning the way data is presented, collated, capitalized, and so on, which can vary from language to language. The rules also define the data classes to which characters belong, including alphabetic, numeric, and space. The ccsversion can address the needs of users who speak the same language but require different methods for processing data because of regional differences in the language or requirements for a specific line of business. Therefore, several ccsversions can be based on the same coded character set." […] A *coded character set mapping table* is a table that maps characters from one coded character set to another coded character set" (Unisys 2013c, p. 1.6).

[233] De hecho, no solo permite internacionalizar aplicaciones programadas en ALGOL, sino que también apoya aplicaciones desarrolladas en C, COBOL, NEWP, Pascal y RPG sobre la plataforma Unisys (*Ibidem*, p. 2.5).

3.2.3.3 Ensamblador

A diferencia de todos los lenguajes que mencionamos, un ensamblador o *assembler* funciona para un microprocesador en específico, con lo cual, cada máquina tendrá su propio lenguaje de ensamblaje según su microprocesador y arquitectura. En general, los ensambladores se valen de comandos de nivel bajo y están más cerca del lenguaje de máquina, en comparación con los lenguajes compilados[234]. Salomon se vale de estas dos definiciones, "[…a]n assembler is a translator that translates source instructions (in symbolic language) into target instructions (in machine language), on a one to one basis […]" y "[… a]n assembler is a *translator* that translates a machine-oriented language into machine language" (1993, p. 1) para explicar en palabras simples cómo funciona un ensamblador. Por su cercanía con el lenguaje binario de las computadoras, es mucho más eficaz al ser ejecutado en el procesador y por ello se usa sobre todo cuando el programador busca mayor velocidad de ejecución de un programa. También es el lenguaje que se usa en *software* incrustado o integrado[235]. Quizás por ser un lenguaje de bajo nivel único para cada arquitectura de procesador es que no existen bibliotecas ni técnicas que den apoyo a la internacionalización del ensamblador.

3.2.3.4 C

El lenguaje C fue creado en los Laboratorios Bell entre 1969 y 1973, y su creación se desarrolla en paralelo con la creación del sistema operativo Unix. Dennis Ritchie toma del lenguaje B (y B lo toma de BCPL) la mayor parte de su sintaxis y crea C. C es un lenguaje polivalente, y aunque está considerado de alto nivel, es lo suficientemente cercano al *hardware* para hacerlo eficaz y rápido al ejecutarlo, mas se distancia lo suficiente de las características computacionales del equipo como para que se pueda portar a otras máquinas sin necesidad de alterarlo. Es un lenguaje orientado a procedimientos que brinda acceso directo a la memoria y a sus registros, y sus objetos compilados resultan muy compactos. Según las opciones y posibilidades de optimización del compilador, los programas en C compilan prácticamente en

[234] Véase la explicación sobre los compiladores y los lenguajes compilados en la página 26 y subsiguientes.

[235] Véase nota 174.

código ensamblado y muchas veces ejecutan tan rápido como si fueran ensamblador (Ritchie 1996, pp. 672-673, 681, 686; Thakur 2015; Mundargi 2014). Durante la década de 1980 su uso se popularizó y se crearon compiladores de C para prácticamente todas las arquitecturas y sistemas operativos: "[…] it became popular as a programming tool for personal computers, both for manufacturers of commercial software for these machines, and for end-users interested in programming […]" (Ritchie 1996, p. 681). El lenguaje se usa para crear *kernels*[236], crear compiladores e interpretadores de lenguajes, crear controladores de dispositivos (*device drivers*), crear aplicaciones para las telecomunicaciones, programar redes, crear aplicaciones de procesamiento de señales digitales, crear gestores de bases de datos y editores de texto (Mundargi 2014).

La biblioteca GNU C ofrece prestaciones limitadas en cuanto a la internacionalización, razón por la cual se recomienda en su lugar usar las bibliotecas de ICU4C que provee los ICU[237]. La plataforma Unisys provee prestaciones integradas al lenguaje por medio de subrutinas externas o bibliotecas de servicios de internacionalización[238].

3.2.3.5 C++

C++ surgió del desarrollo de un lenguaje intermedio entre este y C llamado *C with Classes*. Fue creado en la década de 1980 por Bjarne Stroustrup de AT&T Bell Laboratories. Stroustrup se cita a sí mismo en su artículo "A History of C++":

> '[…] C++ was designed primarily so that the author and his friends would not have to program in assembler, C, or various modern high-level languages. Its

[236] Un *kernel* o núcleo es el mediador entre las aplicaciones y el *hardware*. Al encender un equipo y cargar un sistema operativo, lo primero que carga es el *kernel* y se queda activo hasta que se apaga la máquina. El *kernel* se encarga de sincronizar el uso de los recursos de la computadora por medio de la gestión de discos, de la memoria y de las tareas a ejecutarse (Janssen 2016, s. v. kernel).

[237] Véase el apartado "International Components for Unicode" en la página 177.

[238] Véase (Unisys 2011) y el apartado 3.2.3.2 para más información sobre estas prestaciones.

> main purpose is to make writing good programs easier and more pleasant for the individual programmer […]'' (1993, p. 284).
>
> C++ made object-oriented programming and data abstraction available to the community of software developers that until then had considered such techniques and the languages that supported them […] with disdain and even scorn: ''expensive toys unfit for real problems.'' […] C++ made object-oriented programming and data abstraction cheap and accessible.
>
> (*Ibidem*, pp. 293-294)

La biblioteca Boost.Locale (Beilis 2012) cuenta con una colección completa de funciones para internacionalizar los programas desarrollados en C++. Está basada en los ICU para ofrecer servicios de conversión entre mayúsculas y minúsculas, ordenación de caracteres, formato y cálculo de fechas, horas y husos horarios, análisis de límites de caracteres, palabras, oraciones y cambios de línea, formatos numéricos y de moneda, ortografía, conversión de conjuntos de caracteres, y traducción de cadenas de texto (usando GNU gettext[239]), entre otros.

3.2.3.6 C#

Microsoft lanzó el lenguaje C#[240], que desarrollaron Anders Hejlsberg, Scott Wiltamuth y Peter Golde para esta empresa, a mediados de 2000 como parte del ".NET *Framework*"[241]. Al diseñar el lenguaje, los objetivos del equipo de desarrollo eran que C# fuera

[239] Véase nota 106.

[240] El nombre se lee en inglés como "c sharp" (IPA: si ʃɑrp) pues la almohadilla en realidad es un signo de sostenido (♯vs. #), aunque por lo regular se representa usando el carácter de almohadilla. Aparenta ser un juego de palabras a partir del ++ de C++. "In C, ++ can, depending on context, be read as 'next,' 'successor,' or 'increment' though it is always pronounced 'plus plus'." (Stroustrup 1993, p. 279). Entonces lo que sigue la progresión C, C++ sería, en lenguaje musical, medio tono más arriba, es decir, sostenido, y de ahí C# o do sostenido.

[241] El .Net *Framework* brinda un entorno de programación orientado a objetos que funciona de manera uniforme sin importar en dónde se ejecuta el código objeto (local, distribuido o remoto). Busca minimizar los conflictos entre plataforma y objeto que ocurren al lanzar versiones nuevas, crear un entorno de ejecución de programas seguro y libre de problemas de desempeño, un entorno uniforme para que el desarrollador pueda crear

un idioma orientado a objetos que fuera simple, moderno y polivalente, y que permitiera desarrollar componentes de *software* que pudieran ser implementados en entornos distribuidos. También la portabilidad del código fuente fue un elemento importante en el diseño, así como la facilidad de aprenderlo, especialmente para los programadores que ya conocen C y C++. De igual manera, al concebir y expandir este lenguaje, se tomaron en consideración y se les dio especial importancia a los siguientes elementos: dar apoyo a la internacionalización, que fuera un lenguaje económico en relación con los requisitos de memoria y procesamiento, y que fuera adecuado para crear sistemas alojados y embebidos[242] (Ecma International 2006, p. xix).

La página "Globalizing and Localizing .NET Framework Applications" de la MSDN contiene todas las prestaciones para internacionalizar que ofrece el .NET *Framework*. La sección "Globalization" (que realmente se refiere a internacionalización) explica con detalles y ejemplos cómo atender el manejo de cadenas de texto, fechas y horas, valores numéricos, y configuraciones relacionadas con la cultura. La sección "Localizability Review" explica que el *framework* no apoya los formatos de números de teléfono, las direcciones postales, los tamaños de papel y las unidades de medida. También incluye una sección dedicada a las prácticas recomendadas al desarrollar aplicaciones *world-ready*. Todas estas recomendaciones aplican no solo a C#, sino también a Visual Basic y a todos los demás lenguajes que se sirven del .NET *framework*.

3.2.3.7 COBOL

Las especificaciones técnicas del lenguaje COBOL comenzaron a desarrollarse en 1959,[243] pero, a pesar de ello, sigue siendo un lenguaje vigente en la actualidad: "[…] nearly 15 percent of all **new** enterprise application functionality is written in COBOL, [… it] powers

aplicaciones basadas en Windows y web indistintamente, y que funcione sobre una base de estándares de comunicación de aceptación general para facilitar la integración con otro código (Microsoft Developer Network 2016).

[242] Véase nota 174.

[243] Para sus orígenes, véase página 29.

many everyday services such as ATM transactions, check processing, travel booking, and insurance claims. [… There are] more than 200 billion lines of COBOL code being used across industries such as banking, insurance, retail and human resources […]" (International Business Machines 2013c, énfasis añadido) anunciaba la *Big Blue* en 2013 al ampliar las prestaciones de su compilador de COBOL para *mainframe*. Si observamos la Tabla 3.1 y la Tabla 3.2, los únicos dos lenguajes que tienen apoyo a lo largo de todas las plataformas que presentamos son C y COBOL. No solo existen compiladores comerciales disponibles, sino que también hay tres proyectos de código abierto, GnuCOBOL, OpenCobolIDE y Cobol Bridge for Node.js, para desarrollar programas en COBOL (Holm 2015).

COBOL para las *mainframes* de IBM permite el uso de Unicode, pero solo con el esquema de codificación UTF-16[244] (a nivel interno del programa[245]) (International Business Machines 2013d). No ofrece más prestaciones de internacionalización, sin embargo, si se invocan subrutinas en C o C++ se debería poder tener acceso a la biblioteca ICU4C y se pueden añadir las funcionalidades que brinda esta biblioteca. La plataforma Unisys provee prestaciones integradas al lenguaje por medio de subrutinas externas o bibliotecas de servicios de internacionalización[246].

3.2.3.8 Common Lisp

John McCarthy, pionero en la investigación sobre inteligencia artificial, inventó LISP a finales de la década de 1950 como parte de sus investigaciones en el *MIT Artificial Intelligence Project,* un lenguaje que ha derivado en muchísimos dialectos. "My desire for an algebraic list-processing language for artificial intelligence work […]" fue lo que inició sus investigaciones y su convicción de que "[…r]epresenting sentences by list structure seemed

[244] Véase nota 197.

[245] A nivel externo, el sistema de gestión de bases de datos del equipo se encarga de hacer la conversión entre UTF-8 y UTF-16, pero esto afecta el desempeño del programa (International Business Machines 2013d).

[246] Véase (Unisys 2012a, 2012b, 2013b) y el apartado 3.2.3.2 para más información sobre estas prestaciones.

appropriate—it still is—and a list-processing language also seemed appropriate for programming the operations involved in deduction—and still is […]" (McCarthy 1981, pp. 174, 177). Common Lisp es un dialecto de LISP estandarizado por ANSI, polivalente, de uso industrial y, por medio del *Common Lisp Object System (CLOS)*, con apoyo completo a programación orientada a objetos (Association of Lisp Users 2016).

El apoyo a internacionalización de Common Lisp se limita a cage2 / cl-i18n[247], un *framework* similar a Gettext para la gestión de cadenas de textos en otros idiomas. La versión gratuita CLISP[248] apoya Unicode, pero solo como sistema de codificación de caracteres, es decir, que no ofrece la colección de prestaciones necesarias para una verdadera internacionalización. LispWorks[249], un entorno de desarrollo multiplataformas de Common Lisp tiene el mismo apoyo limitado de Unicode. CLiki, la Wiki de Common Lisp, dedica una página a internacionalización[250], pero los proyectos a los que hace referencia no se han actualizado hace tres o cuatro años, o están abandonados.

3.2.3.9 Delphi

Este lenguaje está basado en Object Pascal y fue desarrollado durante la década de 1990 por Borland, la empresa creadora de Turbo Pascal[251]. El manual del lenguaje lo describe como "[…] a high-level, compiled […] language that supports structured and object-oriented design. […I]ts benefits include easy-to-read code, quick compilation, and the use of multiple unit files for modular programming […]". Este lenguaje junto con su entorno de desarrollo integrado favorece la metodología RAD (de hecho, el entorno se llama "RAD Studio") para desarrollo acelerado de aplicaciones (Embarcadero Technologies 2014).

[247] En https://github.com/cage2/cl-i18n.

[248] En http://clisp.cons.org/.

[249] En http://www.lispworks.com/.

[250] En http://www.cliki.net/internationalization.

[251] Véase el apartado "Pascal" en la página 198.

La documentación dedica un breve capítulo a internacionalización y localización de las aplicaciones, pero, aparte de permitir el uso de Unicode, ni el lenguaje ni la plataforma proveen funciones avanzadas, como por ejemplo las que ofrecen los ICU, para facilitar la internacionalización. De hecho, aunque se pueden desarrollar aplicaciones para Windows, iOS y Android, el manual solo habla de las limitadas prestaciones que toma de Windows.

3.2.3.10 Fortran

Fortran fue el primer lenguaje de alto nivel desarrollado[252] y es el que se usa, aun en la actualidad, para desarrollar aplicaciones de ciencia e ingeniería. En un artículo publicado en 2015, Franklin enumera algunas de las versiones activas del compilador y, si nos dejamos llevar por su lista, podemos observar que está disponible para todas las plataformas principales, incluidas las *mainframes* de IBM: GNU Fortran (Windows, Linux y OS X), Oracle Solaris Studio, IBM (z/OS, z/VM, AIX, Blue Gene y Linux), Intel (Windows, Linux y OS X), Fortran for Android, entre otros (Franklin 2015).

No existe una manera sencilla de internacionalizar aplicaciones desarrolladas en Fortran. Este lenguaje solo maneja conjuntos de caracteres ASCII y permite el uso del conjunto de caracteres UCS-4 según especificado en ISO 10646, pero depende de la implementación que haga el compilador, pues no es obligatorio, y de que sea la especificación Fortran 2008 (JTC 1 2010, sec. 3.1.1, 4.4.3.1). Aun así, existe una especificación técnica de ISO/IEC a fin de promover "[…] portability, reliability, maintainability, and efficient execution of programs containing parts written in Fortran and parts written in C, for use on a variety of computing systems […]" (JTC 1 2012, sec. 1). Usando las prestaciones de integración entre estos dos

[252] Véase la sección dedicada a este tema en el apartado "Los primeros lenguajes de programación" (página 28).

lenguajes, es posible implementar la biblioteca ICU4C en lenguaje C que permite llevar a cabo operaciones de internacionalización sobre las cadenas de texto y cifras, entre otros[253].

3.2.3.11 Haskell

Haskell es un lenguaje funcional puro[254] que se comenzó a desarrollar a finales de la década de 1980 debido a que "[…] there had come into being more than a dozen non-strict, purely functional programming languages, all similar in expressive power and semantic underpinnings […]" y la carencia de un lenguaje común estaba limitando el uso generalizado de este tipo de lenguajes (Hudak et al. 2007, p. 12.1). Según la wiki de Haskell, con Haskell se puede aumentar considerablemente la productividad del programador; el código es más breve, más claro y se le puede dar mantenimiento con mayor facilidad; se cometen menos errores y el programa es más fiable; la "brecha semántica" entre el programador y el lenguaje es menor; y los plazos para crear programas son más cortos (Haskell Wiki 2013).

La página de la wiki[255] proporciona una lista de opciones para internacionalizar los programas desarrollados en Haskell, pero se limita solo a la parte textual, es decir, la que trata la internacionalización como traducción únicamente. El paquete text-icu[256] usa las bibliotecas de los ICU para brindar funcionalidades como acceso a los metadatos en la base de datos de caracteres de Unicode (UCD); funciones para separar caracteres, palabras, oraciones y detectar

[253] Véase el apartado "*International Components for Unicode*" en la página 177 para las prestaciones de los ICU.

[254] Los lenguajes de programación funcionales son los que están basados en programación funcional. Los programas funcionales se ejecutan evaluando *expresiones*, a diferencia de los programas que usan lenguajes imperativos, que se componen de *enunciados* cuyo estado cambia según se ejecuta cada renglón del programa. Si el lenguaje no permite que las expresiones generen *efectos secundarios*, es decir, si no permiten que devuelvan un valor al ser evaluadas, se dice que es un lenguaje funcional *puro* (Haskell Wiki 2014).

[255] En https://wiki.haskell.org/Internationalization_of_Haskell_programs.

[256] En https://hackage.haskell.org/package/text-icu.

cambios de líneas; funciones de ordenamiento de caracteres; funciones de conversión entre esquemas de codificación de caracteres; normalización de representación en Unicode (para poder compararlos correctamente); y búsqueda y reemplazo usando expresiones regulares[257].

3.2.3.12 Java

Java es un lenguaje polivalente, basado en clases y orientado a objetos. Es lo suficientemente simple como para que un programador logre fluidez en el lenguaje en poco tiempo, y es muy similar a C y a C++. Los programas compilan en un formato binario destinado a ser ejecutado sobre la *Java Virtual Machine* (JVM) (Gosling et al. 2015, p. 1). Esta plataforma se diseñó en sus orígenes a fin de solucionar los problemas que surgen al desarrollar programas para aparatos de consumo: ejecución sobre múltiples plataformas y entrega segura de la aplicación a la plataforma en cuestión. No solo esto, sino que el programa también debía poderse transportar sobre cualquier medio, funcionar sobre cualquier cliente[258] y asegurarle al cliente que la aplicación es segura de ejecutar. La JVM es la pieza clave de la plataforma Java, pues funciona como una máquina abstracta[259] que responde a las instrucciones compiladas de Java y gestiona los recursos del equipo sobre el cual se está ejecutando el programa. También se encarga de imponer limitaciones sintácticas y estructurales estrictas para fines de seguridad (Lindholm et al. 2015, pp. 1-2). Java tiene distintas ediciones con diversos fines. La *Java Platform, Standard Edition* (Java SE), permite desarrollar e implantar aplicaciones que exhiben ejecutoria, versatilidad, portabilidad y seguridad sobresalientes en clientes, servidores y entornos embebidos[260] (Oracle Technology Network 2016). El desarrollo de la *Java Platform, Enterprise Edition* (Java EE), está regido por su comunidad de usuarios, los *Java User Groups*,

[257] Véase nota 211.

[258] Cliente se refiere al sistema anfitrión donde se ejecuta el programa. Sobre el concepto cliente-servidor, véase nota 58.

[259] Véase el apartado “Abstracción“ en la página 114.

[260] Véase nota 174.

quienes actúan guiados por el *Java Community Process*. Esta comunidad de usuarios está compuesta por "[…] industry experts, commercial and open source organizations […], and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity […]" (Oracle Technology Network 2015a). La *Java Platform, Micro Edition* (Java ME) está dirigida a brindar un entorno flexible para crear aplicaciones que se ejecuten sobre aparatos embebidos[261] y móviles para la IoT[262]: microcontroladores, sensores, portales (*gateways*), teléfonos móviles, cajas de TV e impresoras, entre otros. Las aplicaciones que se desarrollan usando Java ME mantienen su portabilidad entre aparatos, pero permiten aprovechar las prestaciones nativas que cada equipo ofrece (Oracle Technology Network 2015b).

Java incluye prestaciones propias de internacionalización, cuya implementación se detalla ampliamente en la página web titulada "Java Internationalization"[263]. También se puede usar la biblioteca de los ICU para Java, ICU4J[264].

3.2.3.13 Objective-C

Objective-C es un lenguaje orientado a objetos, superconjunto del lenguaje C, que se usa para desarrollar aplicaciones para OS X e iOS. La App Store de Apple está disponible en más de ciento cincuenta países, razón por la cual esta empresa brinda apoyo extenso a la internacionalización y localización. Cuenta con más de cuarenta idiomas hacia los cuales se pueden localizar los productos desarrollados (Apple Inc. 2014, 2016b).

[261] *Idem*.

[262] Véase nota 60.

[263] En http://www.oracle.com/technetwork/java/javase/tech/intl-139810.html.

[264] En http://site.icu-project.org/home.

La página principal que recoge las prestaciones de internacionalización que ofrece Apple a quienes desarrollan sobre su plataforma[265] incluye vídeos, documentación y muestras de código con ejemplos para iOS y OS X. El *Foundation framework*, que usa los ICU, incluye todas las prestaciones necesarias para internacionalizar una aplicación desarrollada en Objective-C.

3.2.3.14 OCaml

La primera implementación de Caml, un acrónimo de *Categorical Abstract Machine Language*, aparece en 1987 y continuó desarrollándose hasta 1992. Para 1996 se le añade capacidad de programación orientada a objetos y lo renombran Objective Caml, OCaml para acortar. Es un lenguaje funcional polivalente con énfasis en la expresividad y en la seguridad, muy adecuado para enseñar programación, que además tiene prestaciones de lenguaje imperativo[266] («A History of OCaml» 2014, «What is OCaml» 2014).

La capacidad para internacionalizar OCaml se limita a una implementación básica de GNU gettext[267].

3.2.3.15 Pascal

En 1971 uno de los creadores originales de ALGOL, el Dr. Niklaus Wirth, publicó las especificaciones de un nuevo lenguaje de programación estructurado[268], muy similar a ALGOL, que bautizó Pascal en honor al filósofo y matemático francés. La estructura de Pascal es libre,

[265] En https://developer.apple.com/internationalization.

[266] Véase nota 254.

[267] En https://opam.ocaml.org/packages/gettext/gettext.0.3.5/.

[268] La programación estructurada busca desarrollar programas más fáciles de comprender por medio de la creación de subrutinas que agrupan procesos lógicos y el uso de construcciones que se limitan a estructuras secuenciales, es decir, una serie de comandos que se ejecutan uno tras otro, estructuras de selección —las estructuras IF-THEN-ELSE son un ejemplo— y estructuras iterativas o de bucle para las repeticiones.

es decir, que no está restringida por columnas[269], lo que hace que el código sea fácil de leer pues se lee como un idioma natural. El profesor Ken Bowles, de la Universidad de California en San Diego, adaptó el compilador para que pudiera funcionar en la Apple II, y, gracias a ello, lo convirtió en poco tiempo en el lenguaje de programación más usado en las universidades de EE. UU., pues el costo de este equipo era considerablemente menor si se compara con una *mainframe*, que era lo único que había disponible en las universidades de esa época para ejecutar programas. Al surgir las PCs, la empresa Borland lanzó Turbo Pascal, un compilador que, fiel a su nombre, generaba programas a velocidades exorbitantes para la época y que se convirtió en el compilador por excelencia para desarrollar aplicaciones sobre el sistema operativo DOS de la PC. La popularidad de Pascal decayó cuando AT&T comenzó a distribuir de manera gratuita a las universidades el sistema operativo Unix y los estudiantes comenzaron a aprender C. La popularidad de los lenguajes orientados a objetos, la creación de C++ y el hecho de que Microsoft adoptara C como el estándar para desarrollar aplicaciones para Windows terminó de rematarlo. A pesar de ello, sigue siendo un lenguaje muy utilizado, sobre todo en la enseñanza de la programación a nivel universitario, sobre todo fuera de los EE. UU. (Yue 2011).

El compilador "Free Pascal"[270] ofrece localización de cadenas de texto usando GNU gettext, pero, aunque en su wiki de documentación indica que da apoyo a funciones para manipular fechas, cadenas de texto y lenguajes RTL, el apoyo no es completo. La plataforma

[269] Cuando comenzaron a desarrollarse lenguajes de programación, muchos idiomas tenían restricciones en cuanto a la forma en que se escribían, pues el *input* estaba limitado por las ochenta columnas que cabían en una tarjeta perforada estándar y los creadores de lenguajes de programación necesitaban establecer parámetros para designar el uso de las columnas a fin de aprovechar al máximo la limitación de ochenta caracteres. En el lenguaje RPG, que repasaremos en el apartado con el mismo nombre en la página 200, por ejemplo, la columna 6 define el tipo de información que contiene la tarjeta. Si tiene una "C", corresponde a una sección de cálculos donde se puede llevar a cabo acciones de aritmética, lógica o de manipulación de datos. Esta especificación denominaba también la configuración de las restantes columnas, qué valores aceptan y qué cálculos o decisiones se tomaban según la información contenida en ellas. Las versiones más nuevas de este lenguaje han flexibilizado estas restricciones.

[270] En http://www.freepascal.org/.

Unisys provee prestaciones integradas al lenguaje por medio de subrutinas externas o bibliotecas de servicios de internacionalización[271].

3.2.3.16 PL/1

A principios de la década de 1960, las líneas divisorias entre programadores, programa y equipo científicos, comerciales y especializados estaban mucho más demarcadas. Con el pasar del tiempo, esta división comenzó a cerrarse pues las necesidades de estos grupos de usuarios se ampliaron de forma tal que "invadían" espacios[272]. Estos cambios en necesidades son una de las razones por las cuales IBM desarrolla el System/360[273] y, junto con el grupo de usuarios SHARE[274], forma el proyecto "Advanced Language Development Committee of the SHARE FORTRAN", de donde surgen las especificaciones para PL/1 (Radin 1978, p. 227). Es un lenguaje polivalente que se usa para resolver problemas en campos tan variados como el comercio, la ciencia, las matemáticas, la ingeniería y la medicina, entre otros. Adecuado para principiantes, el código es fácil de leer pues incluye buenas prestaciones para documentación y, por su estructura y sintaxis flexibles, se autodocumenta (Vowels 2010).

[271] Véase (Unisys 2009) y el apartado 3.2.3.2 para más información sobre estas prestaciones.

[272] El autor citado usa varios ejemplos para explicar las nuevas necesidades que iban surgiendo. Los usuarios comerciales, cuyas necesidades en un principio se limitaban a cálculos básicos de suma, resta, multiplicación y división, comenzaban a hacer regresiones lineales y análisis factorial sobre sus datos, con lo cual cambiaban sus necesidades originales en cuanto a precisión numérica de aritmética decimal simple a cálculos complejos que necesitan hacer uso de coma flotante, antes solo usadas por los científicos. Por otro lado, los científicos, que se conformaban con un listado enigmático de números, comenzaban a apreciar el valor de informes claramente formateados (Radin 1978, p. 227).

[273] Véase el apartado "La revolución de las *mainframes*" en la página 31.

[274] SHARE es un grupo de usuarios, originalmente de las *mainframes* de IBM, formado en 1959 que aún en la actualidad apoya a la comunidad informática (http://www.share.org/about).

PL/1 para las *mainframes* de IBM permite el uso de Unicode, pero solo con el esquema de codificación UTF-16[275] (a nivel interno del programa[276]) (International Business Machines 2013e).

3.2.3.17 RPG

Cuando IBM presenta su sistema IBM 1130 en 1965, describe al *Report Program Generator*, mejor conocido como RPG, como un lenguaje orientado a resolver problemas que, por medio de técnicas eficaces y fáciles de usar, genera programas que pueden leer datos de uno o varios ficheros, hacer cálculos sobre estos datos e imprimir informes a partir de ellos (International Business Machines 2003a). Originalmente concebido como un lenguaje limitado a usar ciertas columnas en cada renglón con reglas específicas sobre qué comando o código va en qué columna, las versiones más recientes permiten un formato libre para especificar las distintas secciones que conforman un programa en RPG (Paris y Gantner 2013).

La plataforma de IBM i es la que ofrece más prestaciones para internacionalizar programas en RPG. La base de datos de todos los equipos IBM permiten el uso de Unicode y la *Character Data Representation Architecture*[277] se asegura de que se hagan las conversiones de caracteres adecuadas entre las distintas interfaces (bases de datos, ficheros, equipo, programas, etc.) o al transmitir datos entre sistemas con esquemas de codificación de caracteres distintos. Tiene una biblioteca de APIs[278] (*National Language Support-related APIs*[279]) para gestionar

[275] Véase nota 197.

[276] A nivel externo, el sistema de gestión de bases de datos del equipo se encarga de hacer la conversión entre UTF-8 y UTF-16, pero esto afecta el desempeño del programa (International Business Machines 2013e).

[277] "Character Data Representation Architecture (CDRA) is an IBM architecture that defines a set of identifiers, services, supporting resources, and conventions to achieve consistent representation, processing, and interchange of graphic character data in data processing environments" (International Business Machines 2013a).

[278] Véase nota 171.

[279] En https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/apis/nls2.htm

funciones relacionadas con internacionalización como, por ejemplo, movimiento entre los elementos textuales, conteo de elementos textuales, conversión entre mayúsculas y minúsculas, ordenación de caracteres, captura de datos sobre el *locale* y transformación de datos. Carece de funciones para gestionar los formatos numéricos, de fechas, direcciones, etc., pero puede invocar las API de los ICU para C y C++, con lo cual, por medio de estas permite la internacionalización[280]. Las presentaciones de Milligan (2014a, 2014b) explican en detalle cómo internacionalizar (él lo llama *globalizar*) los programas en RPG del IBM i, dar apoyo a Unicode y qué hacer para migrar a sistemas que usen solo Unicode. Estas explicaciones también funcionan para programas desarrollados en COBOL sobre esta plataforma. Para los equipos Unisys, la versión de RPG para MCP apoya la internacionalización, principalmente por medio de APIs. El manual de referencia dedica el capítulo 5 a explicar las prestaciones para internacionalizar[281] que ofrece esta plataforma (Unisys 2008).

3.2.3.18 Smalltalk

Smalltalk no es un lenguaje de programación específico, sino que se refiere más bien a una familia de lenguajes de programación que fueron desarrollados en Xerox PARC aproximadamente en la década de 1970 (Budd 1993, p. 1). Su creador fue también el creador del paradigma de diseño conocido como *object-oriented* que surge del concepto de que todo puede ser descrito y representado por "the recursive composition of a single kind of behavioral building block that hides its combination of state and process inside itself and can be dealt with only through the exchange of messages" (Kay 1993, p. 512), es decir, por una composición de "cajas negras" (que en efecto son abstracciones[282]). Los objetos tienen seis características[283]: todo es un objeto; los objetos se comunican por medio de mensajes; cada objeto tiene su propio espacio de memoria; cada objeto es una instanciación de una clase (que también es un objeto);

[280] En http://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/apis/icu.htm?lang=en

[281] Véase también el apartado 3.2.3.2.

[282] Véase el apartado "Abstracción" en la página 114.

[283] Estas características las mencionamos también en la nota 191 de la página 161.

la clase contiene los comportamientos que comparten sus instanciaciones; y las clases están organizadas bajo una estructura de árbol con una raíz única llamada la *jerarquía de herencias*, de manera que la memoria y los comportamientos asociados a una instanciación de una clase están disponibles a los descendientes de esta estructura de árbol (Budd 1993, p. 1).

Squeak[284] es una plataforma de código abierto para desarrollar programas en Smalltalk sobre Windows, Mac y Linux. En la documentación no queda claro si ofrece apoyo a la internacionalización o no. GNU Smalltalk[285] también ofrece apoyo limitado a la internacionalización (traducción de cadenas de texto, formato de fechas, monedas y cifras). La internacionalización de Cincon Smalltalk[286] está basada en el CLDR. VA Smalltalk[287] lleva varios años informando que dará apoyo a internacionalización por medio de los ICU, pero aún no lo incluye en su plataforma.

3.2.3.19 Swift

En diciembre de 2015 Apple lanzó Swift como un proyecto de código abierto en GitHub. Es una colección de proyectos que comprenden el compilador de Swift, una biblioteca estándar, un conjunto de bibliotecas nucleares (*core*), un depurador y un gestor de paquetes Swift. Se puede usar para desarrollar aplicaciones para todas las plataformas que ofrece Apple más Linux, y se espera que, por ser un proyecto de código abierto, la comunidad de programadores genere implementaciones que funcionen sobre Windows. Swift es un lenguaje que tiene un acercamiento actualizado hacia la seguridad y el desempeño, y usa patrones de diseño de *software* modernos. Los creadores también lo describen como un lenguaje muy expresivo que tiene características que lo hacen fácil de leer y escribir (Apple Inc. 2016a).

[284] En http://squeak.org/.

[285] En http://smalltalk.gnu.org/.

[286] En http://www.cincomsmalltalk.com/.

[287] En http://www.instantiations.com/.

Swift ofrece apoyo completo a la internacionalización de las aplicaciones que se desarrollan en este lenguaje por medio de las API del *Foundation framework* de Objective-C, basadas en los ICU.

3.2.3.20 Tcl/Tk

El desarrollador de Tcl, siglas de *Tool Control Language*, creó ese lenguaje a principios de la década de 1980 a raíz de los trabajos de investigación suyos y de su grupo de investigadores sobre circuitos integrados que realizaron en la Universidad de California en Berkeley. El objetivo era crear un lenguaje extensible, es decir, que cada aplicación pudiera añadir sus propias prestaciones a las prestaciones básicas del lenguaje y que estos añadidos parecieran ser parte del lenguaje original. El lenguaje tenía que ser simple y genérico a fin de que no restringieran las prestaciones de la aplicación desarrollada y tenía que integrar todas las extensiones que se le hicieran con facilidad y fluidez. Sobre todo, el lenguaje tenía que poder ser embebido[288] . Por otro lado, el interés generalizado (y el suyo) durante esta década en las GUI lo estimuló a desarrollar una extensión para añadir un componente gráfico para este lenguaje, la cual llamó Tk (Ousterhout [sin fecha]).

Tcl representa los datos internos en formato UTF-8, con lo cual, en principio, debería dar apoyo amplio a la internacionalización. Sin embargo, las prestaciones de internacionalización que ofrece se limitan a la conversión entre esquemas de codificación de caracteres y manejo de cadenas de caracteres («How to Use Tcl 8.1 Internationalization Features» [sin fecha]). Por otro lado, es un lenguaje que permite invocar programas en C, con lo cual debería ser posible integrar las ICU4C y ampliar el apoyo a la internacionalización.

3.2.3.21 Visual Basic .NET

A fines de la década de 1980 desarrollar aplicaciones para las PC con Windows implicaba programar en C y C++. A fin de simplificar el desarrollo de programas en esta

[288] Véase nota 174.

plataforma, Microsoft expande el popular lenguaje BASIC[289] y lo hace funcionar sobre Windows cuando presenta Visual Basic 1.0 en mayo de 1991. Ya para el año 2003 Visual Basic iba por la versión 6.0 —la que más éxito tuvo desde que se lanzó el producto— cuando Microsoft presenta el .NET *Framework*[290]. Visual Basic .NET tiene prestaciones totalmente basadas en objetos y está integrado por completo al *framework* .NET. El enfoque de esta plataforma es en el desarrollo rápido de aplicaciones, característica que lo hace un lenguaje muy usado (Max Visual Basic 2010; HitMilL 2006). Las prestaciones de internacionalización las provee el .NET *framework*[291].

3.2.4 Entornos de desarrollo integrados

Los entornos de desarrollo integrados (IDE por sus siglas en inglés) son programas que facilitan la creación de otros programas. Son entornos gráficos que incluyen prestaciones que le permiten al programador trabajar con más eficacia e incluso, en algunos casos, a trabajar en equipo. Los IDE proveen información contextualizada en relación con el lenguaje de programación que se está usando, incluyen funcionalidad para hacer depuración de errores de los programas, gestionar las versiones que se generan, y permiten expandir y ampliar sus

[289] BASIC es el acrónimo de *Beginners All-purpose Symbolic Instruction Code*, un lenguaje de programación simple creado por Thomas E. Kurtz y John J. Kemeny en Darmouth College entre 1963 y 1964 (Lewis 1978). Es el lenguaje más utilizado por su simplicidad y porque fue el primer lenguaje para el cual se creó un interpretador (lo crearon Bill Gates y Paul Allen) que funcionaba en las primeras computadoras personales que se crearon. "If ordinary persons are to use a computer, there must be simple computer languages for them. BASIC caters to this need by removing unnecessary technical distinctions […] and by providing defaults […] where the user probably doesn't care. BASIC also shows that simple line-oriented languages allow compact and fast compilers and interpreters, and that error messages can be made understandable" (Kurtz 1981, p. 535)

[290] Véase nota 241.

[291] Véase la sección sobre internacionalización del apartado " C#" en la página 190.

prestaciones por medio de *plugins*[292]. La similitud entre las interfaces del usuario que ofrecen los IDE ayudan a optimizar la labor del programador y a mejorar su productividad (Janssen 2016, s. v. Integrated Development Environment [IDE]). Algunos IDE ya comienzan a incluir, entre las prestaciones que ofrecen, servicios para facilitar la localización por medio de intercambio de archivos usando formatos estándar[293] o de interfaces (y prestaciones) similares a las las herramientas de TAO.

A continuación, presentamos una selección de IDE a fin de que el internacionalizador se familiarice con ellos. Los IDE que reseñamos brevemente solo incluyen algunos de los IDE que se usan para desarrollar programas en las tres plataformas principales que hemos estudiado a lo largo de esta investigación[294].

Eclipse

Eclipse surgió como una iniciativa de IBM que buscaba reducir la gran cantidad de entornos de desarrollo incompatibles que ofrecía a sus clientes y estimular la reutilización de los componentes que estos entornos tenían en común. La idea era que todos trabajaran sobre un *framework* que les permitiera a los integrantes de equipos de desarrollo de *software* aprovechar los componentes ya desarrollados sobre distintas plataformas, que pudieran tener un alto nivel de integración y que los desarrolladores pudieran desplazarse entre distintos tipos de proyectos con facilidad. Desarrollaron una nueva plataforma integradora extensible que permitía ampliar las prestaciones básicas que ella ofrecía por medio de *plugins* y luego la convirtieron en un proyecto de código abierto bajo el auspicio de la Eclipse Foundation, una empresa sin fines de lucro que se encarga de gestionar la infraestructura de TI y la propiedad

[292] Los *plugins* son elementos que se añaden a un programa o aplicación para ampliar la funcionalidad o agregar prestaciones nuevas y hacen que los productos que los admiten sean más versátiles y se adapten a los gustos y necesidades del usuario (Janssen 2016, s. v. plug-in).

[293] Como por ejemplo, XLIFF (véase nota 105) en la IDE de Apple (Apple Inc. 2015b)

[294] No incluimos XCode, la IDE de Apple, porque solo genera aplicaciones para equipos Apple ni Android Studio pues genera aplicaciones solo para equipos que funcionan bajo el SO de Android.

intelectual, dar apoyo a la comunidad de desarrollo y desarrollar el ecosistema por medio del mercadeo y la promoción (Veys y Laffra 2006; Milinkovich 2005).

Eclipse es un entorno de desarrollo integrado con herramientas para gestionar el entorno de trabajo; crear, lanzar y depurar programas; y compartir artefactos con un equipo de trabajo y gestionar las versiones. El mecanismo que tiene para descubrir, integrar y ejecutar *plugins* permite al usuario expandir la IU y ejecutar una tarea específica (Roy, Veys y Laffra 2007). Eclipse está disponible en las plataformas Mac, Windows y Linux, y viene con paquetes preconfigurados para trabajar con lenguajes de programación específicos (Java y C/C++, entre otros).

Embarcadero RAD Studio

Embarcadero es un IDE que permite desarrollar aplicaciones para Windows y Mac OS X, para las plataformas móviles iOS y Android, y además permite desarrollar soluciones innovadoras para la IoT[295]. Si se quieren desarrollar apps para la Mac, el entorno tiene que estar instalado en una Mac y usar una máquina virtual con Windows 10. Admite los lenguajes C++ y Object Pascal como lenguajes de desarrollo (Embarcadero Technologies 2016).

IBM Rational IDE

Este es el entorno de desarrollo integrado de IBM construido sobre Eclipse que sirve para desarrollar aplicaciones para *mainframe*, *midrange* y web. Está disponible en varias ediciones y permite desarrollar aplicaciones en COBOL, RPG, Java, Javascript, PL/1, C/C++ y JCL sobre Linux y Windows[296] (International Business Machines 2016).

[295] Véase nota 60.

[296] Según el sistema sobre el cual se ejecutará la aplicación que se está desarrollando. El IDE para Linux genera aplicaciones en Linux y el de Windows para todas las demás plataformas que funcionan sobre los sistemas IBM.

MonoDevelop

Mono es una plataforma de desarrollo de código abierto basada en el *framework* de .NET de Microsoft y auspiciada por esta misma empresa, la cual se rige por los estándares de Ecma[297] en cuanto al lenguaje C# y el entorno de ejecución "Common Language Runtime[298]" (Mono Project 2016a). El IDE que usa esta plataforma se conoce como MonoDevelop y permite desarrollar aplicaciones en C#, F#, Visual Basic .NET, C/C++ y Vala sobre Windows, OS X y Linux (MonoDevelop Project 2016).

NetBeans IDE

Es el IDE oficial para desarrollar aplicaciones en Java, PHP, Java FX, C/C++ y JavaScript, entre otros. Es un proyecto de código abierto que, al igual que Eclipse, admite *plugins* para expandir el apoyo que ofrece la IU y funciona sobre Windows, OS X y Linux (Oracle Corporation 2016).

Visual Studio

Es el IDE de Microsoft para desarrollar aplicaciones sobre Windows en C#, JavaScript, C++, Visual Basic y F#, entre otros. Acepta también *plugins* para ampliar su funcionalidad y, usando Xamarin[299], permite desarrollar apps en C# que comparten el código fuente y funcionan en iOS, Android y Windows (Microsoft 2016a, 2016b, 2016c).

---

[297] Ecma Internacional es una asociación de industrias fundada en 1961 que se dedica a la estandarización de las TIC y de los dispositivos electrónicos. Hasta 1994 se conoció como la *European Computer Manufacturers Association* (de ahí Ecma) pero luego cambió a *Ecma International*, nombre que refleja el carácter más globalizado hacia el cual se enfocó la asociación. Su TC49 se encarga de estandarizar C#, Eiffel, la CLI y las extensiones de C++ para la CLI (Ecma International 2016c, 2016a, 2016b).

[298] El "Common Language Runtime", una implementación de la ECMA CLI (véase nota 297), es una máquina virtual que permite ejecutar programas que son compilados conforme a estas especificaciones sobre las plataformas Linux, Mac (OS X e iOS), Sun Solaris, BSD, Windows, Nintendo Wii y Sony Playstation (Mono Project 2016b). El principio de la máquina virtual es igual a la JVM (véase el apartado "Java" en la página 195).

[299] Véase https://www.xamarin.com/.

3.2.5 Las cadenas de texto

Antes de repasar las guías de redacción, debemos detenernos en la manera en que los programadores construyen estas cadenas de texto, pues ello afecta a su contenido y, a fin de cuentas, a su redacción. Además, la unión de cadenas de texto o concatenación es una de las prácticas más comunes de los programadores y su uso sin entender las repercusiones en el proceso de localización ocasionan errores y problemas más adelante cuando se comienza a traducir.

> […] Reuse is such a pervasive element in the software development lifecycle that it has a major impact on […certain] aspects of a global application. […For example,] some of the text strings used to create the user interface may be reused from one place to another in order to save precious time and lines of code. […T]his strategy can work well in some cases, but it may have serious consequences when the context changes. […]
>
> (Roturier 2015, p. 52)

Es importante que un internacionalizador entienda por qué los programadores construyen las cadenas de texto de la manera que las construyen y cómo los pueden ayudar a mejorarlas. De igual manera, es muy posible que en el punto en donde un programador redacta una cadena de texto pueda proveer al localizador información contextual y de lógica relacionada con dicha cadena de texto. La intervención del internacionalizador a este respecto puede optimizar, mejorar y facilitar el proceso de localización. De hecho, Chandler y Deming (2012) sugieren que esta labor sea un trabajo de equipo entre el localizador y el programador.

> […] Have programmers and translators work together to design the basic mechanics for text and dialogue. This will help them best determine the formulae and algorithms that can account for the syntax in different languages and guarantee natural and believable dialogue.
>
> (*Ibidem*, p. 112)

Esta sugerencia la hacen en el contexto de desarrollo de videojuegos, pero la finalidad que persiguen aplica de igual manera al desarrollo de *software* multilingüe. Las autoras, incluso, desalientan el uso de texto concatenado.

> Avoid using concatenated text, which is created by pulling two separate text strings from the game code and displaying them as a single sentence in the game. This can cause a lot of difficulty in localizations, especially with verb tenses and gender. Although concatenated strings may display grammatically correct in the source language version, it may be difficult to program code to display the strings grammatically correct in other languages.
>
> (*Ibidem*, p. 127)

La concatenación de cadenas de texto puede muy tentadora para los programadores pues reduce la cantidad de texto que hay que mecanografiar (al momento de redactar la cadena en sí). Por otro lado, aparenta favorecer la localización pues, si se usa una TAO, una cadena reusada significa que no hay que volverla a traducir. Sin embargo, cada idioma tiene una sintaxis particular y al concatenar cadenas de texto podemos terminar con resultados incorrectos a nivel sintáctico. No solo eso, la concatenación de cadenas no discrimina cuestiones de género y número, lo que añade más problemas de incorrección gramatical. (Roturier 2015, pp. 65-66) El internacionalizador tiene la capacidad de trabajar mano a mano con el programador para poder optimizar las tareas de codificación que involucren reutilización de cadenas de texto (por eso se usan, porque se pueden reutilizar, como menciona Roturier). Lo ayudará a que estas cadenas se constituyan de forma tal que puedan concatenarse adecuadamente y en consideración de las "(de)limitaciones" (como género y número) que imponen otras lenguas hacia las que podría localizarse la aplicación.

A la par con la creación de las cadenas de texto está la tarea de documentación[300] y creación de comentarios. Añadir comentarios para comunicar el contexto al localizador es una forma de agilizar el proceso de localización, sin embargo, en muchas plataformas incluir este tipo de comentarios es totalmente opcional, con lo cual es más fácil para el desarrollador omitirlos. Roturier (*Ibidem*, pp. 64-65) dice que esta omisión se puede deber a que el desarrollador (o el autor o el editor de las cadenas de texto) no tiene tiempo de escribirlos o

[300] Aquí en el sentido de la documentación interna, dentro del programa, para referencia del programador o de quien modifique el programa en un futuro, no de la documentación que se produce para consumo de los usuarios.

que no se siente cualificado para escribirlos ni para dar una recomendación a los traductores. No se puede, sin embargo, resaltar lo suficiente cuán importante son los comentarios para el localizador, muy en especial si tenemos en cuenta que este se encuentra en un espacio descontextualizado y, muchas veces, imposibilitado de accederlo.

Quizás esto esté cambiando poco a poco y veamos en el futuro más concienciación hacia las necesidades del localizador. Durante la presentación de las nuevas prestaciones disponibles para la internacionalización productos Apple en el congreso para desarrolladores de Apple de 2015, Nat Hillard nos explica que para la función que gestiona la localización de cadenas de textos en Swift (véase el apartado 3.2.3.19 Swift en la página 210) es obligatorio incluir un comentario para fines de proveer información contextual al localizador.

> "[…] Here's where NSLocalizedString comes into play. […] It takes five parameters, three of which have default values. Interestingly, in Swift, we've made it such that the comment argument is non-optional. This is really emphasizing this is a critical argument. It provides context for your translators. A given word may be ambiguous in certain contexts and this comment parameter will allow you to customize that. So, the key is the string you wish to translate, and the comment is the comment explaining it. […]"
>
> (Hillard 2015, 07:34-08:07)

Esto, desde luego, no garantiza que el comentario añada la información que el localizador necesita, pero el hecho de que esta información se pueda comunicar proporciona la posibilidad de facilitar la localización y mejorar la calidad del producto localizado. Para Roturier, los posibles factores que contribuyen a la falta de acceso al contexto son, primero, que el desarrollador tiene la expectativa de que el traductor está ya familiarizado con el contexto y, por tanto, no lo necesita; segundo, por cuestiones de confidencialidad no le facilita al traductor una versión ejecutable para referencia o pantallazos, aun mediando un acuerdo de confidencialidad[301]; y, tercero, complejidad derivada por el trabajo adicional que implica

[301] Esto sucede también en la traducción audiovisual, donde a los traductores no se les da en muchas ocasiones el vídeo con el programa o película que tienen que traducir, por temor a las filtraciones o la piratería, lo cual redunda, necesariamente, en una traducción mala o errónea (Fuentes Luque 2016).

facilitarle al traductor esta información que el desarrollador considera innecesaria. (Roturier 2015, pp. 64-65)

Otras consideraciones importantes relacionadas con las cadenas de texto son usar nombres de variables descriptivos y prestar especial atención a las consideraciones relacionadas con el género y el número de las palabras. Los nombres de las variables no tienen ninguna relación directa con la localización, pero si el localizador tiene acceso a ellas y el programador usa un nombre descriptivo, podrían proveer contexto adicional para la traducción. (*Ibidem*, pp. 66-67)

3.2.6 Guías de redacción

Como hemos visto, parte de la labor del internacionalizador es asegurarse de que las cadenas de texto que serán sometidas a localización estén bien construidas y contengan datos contextuales relevantes y claros. También debe asegurarse de que estén bien redactadas para que el localizador pueda realizar un trabajo de mejor calidad. La manera en que se redacta el texto influye el producto final de la traducción. O'Hagan y Ashworth (2002, pp. 30-32) explican varias maneras en que se produce un texto para fines de localización y cómo este texto es recibido por el lector. El primer caso tiene como resultado una "[…] 'afterthought translation' [which] means that the source language text is not prepared with translation in mind. This may result in awkward translation for the target language Receiver in large parts resembling the source language (translatese) […]". Un texto fuente que ha sido internacionalizado será comprensible al receptor que conoce el lenguaje fuente y permitirá al traductor crear una versión traducida comprensible para el lector que no conoce para nada el lenguaje fuente.

> […] the source language is modified by being internationalized, and is understandable both to a Receiver who speaks the source language and to a Receiver for whom it is a second language. […] the internationalized text if further localized to become intelligible to other Receivers, who do not know

> the source language. The internationalization process makes the Message more amenable to the subsequent translation into the Receiver's language. […]
>
> (*Ibidem*, pp. 31-32)

Los autores sugieren seguir una guía de redacción para que el texto fuente sea comprensible no solo a los lectores con el idioma fuente como lengua materna, sino también a los lectores del idioma fuente como segundo idioma.

Las multinacionales Google, Apple, Microsoft e IBM han desarrollado guías con directrices de redacción que son de gran ayuda para los internacionalizadores a la hora de entender la importancia de la redacción de las cadenas de texto. Existen otras guías, pero nos enfocaremos en las que producen estas cuatro multinacionales puesto que contienen muchísima cantidad de información, suficiente como para que el internacionalizador contrastarlas y escoger las más relevantes para el proyecto en el cual esté trabajando.

Microsoft resume el impacto de la redacción de textos en el usuario explicando las observaciones que hacen en sus laboratorios de usabilidad.

> It's often with mixed emotions that we watch customers who are unable, or only partially able, to use a new feature simply because the words on the user interface aren't easy to follow or descriptive enough. We know the words in these early prototypes were probably created hastily by an overworked engineer using nerdy language that was not intended for primetime.
>
> (Microsoft Corporation 2012, p. xvii)

El internacionalizador, como mediador y conocedor de las culturas meta podrá revisar y optimizar la calidad del producto generado por el programador, incluso elaborar glosarios, especificar terminología y amplar las explicaciones y, además, asegurarse de que su redacción sea clara antes de que llegue a manos del localizador. La información que contienen estas guías es importante referencia, no importa sobre qué plataforma se esté desarrollando la aplicación en cuestión.

3.2.6.1 Google

Google mantiene una guía de diseño de materiales para crear a lo largo de toda la marca Google un lenguaje visual coherente y afín a los principios del buen diseño. Esta guía, que se actualiza constantemente, recoge las características que la empresa persigue en el diseño de interfaces de sus productos (Google 2016b). A lo largo de esta guía encontramos varias secciones que se dedican a brindar información que de una manera o de otra facilita la localización. La sección que dedican a la redacción de los textos subraya la importancia de crear apps que contengan textos fáciles de entender sin importar su cultura ni su idioma. Los redactores de textos deben dirigirse a los usuarios en segunda persona excepto cuando sea necesario enfatizar la titularidad del contenido o la responsabilidad de llevar a cabo una acción en la aplicación. Deben ser concisos, usar la voz activa, escribir en presente, evitar terminología técnica, omitir frases innecesarias y buscar uniformidad en el uso de verbos. La redacción debe estar centrada en el usuario, debe sonar familiar pero respetuosa. La guía explica el uso de las mayúsculas, de los signos de admiración y de los numerales. También ofrece avisos importantes relacionados con la localización: no usar metáforas cargadas de implicaciones culturales ni expresiones idiomáticas, evitar la mención de direcciones (derecha/izquierda) relacionadas con la interfaz y exhorta al redactor a evitar ambigüedad en el género, a fin de facilitar la labor del localizador. También hace hincapié en la importancia de que los programadores escriban en los comentarios de las cadenas de texto descripciones claras sobre el uso de estas para que el localizador pueda tener una mejor idea del contexto dentro del cual está traduciendo (Google 2016c). Además, incluye una sección que trata sobre los aspectos que afectan a la interfaz cuando se localiza para dar apoyo a idiomas bidi o que usan sistemas de numeración distintos (Google 2016a).

3.2.6.2 Apple

La guía de estilo principal de Apple está dirigida a la generación de textos y documentación técnica de la marca, no especialmente a la localización. Aun así, la sección titulada “International Style” explica brevemente que la redacción debe evitar expresiones idiomáticas y coloquialismos, que las estructuras de las oraciones deben ser simples y que se deben usar estándares y convenciones internacionales al expresar moneda, cifras, fechas y horas,

idiomas, números de teléfonos y medidas a fin de ayudar la labor del localizador (Apple Inc. 2013). Para el diseño de interfaces, se deben usar las directrices de interfaz humana para iOS y para OS X. Ambas incluyen una sección titulada con el mismo nombre, "Terminology and Wording", pero con contenido variado que explica los parámetros que hay que seguir. La guía dirigida a iOS es más breve y exhorta a quien redacta a

- "[…u]se terminology that you're sure your users understand […]",
- "[…u]se a tone that's informal and friendly, but not too familiar […]",
- "[…t]hink like a newspaper editor, and watch out for redundant or unnecessary words […]".

Estos esfuerzos más bien quedan fuera del ámbito de conocimiento de un programador. También aconseja usar nombres cortos o usar iconos para las etiquetas de los controles[302], a prestar atención al describir fechas[303] y a aprovechar la oportunidad que ofrece la App Store para comunicarse con sus posibles usuarios (Apple Inc. 2016c). La guía para OS X, un poco más extensa, hace las mismas sugerencias sobre evitar el uso de jerga y de términos técnicos, explica el uso adecuado de "terminología Apple", el uso de mayúsculas, abreviaturas, elipsis y dos puntos (Apple Inc. 2015c).

3.2.6.3 IBM

Un recorrido del índice de *The IBM Style Guide: Conventions for Writers and Editors* (DeRespinis y International Business Machines Corporation 2012) refleja que este libro contiene recomendaciones sobre idioma y gramática, puntuación, formato y organización del texto (encabezados, listas, procedimientos, figuras y tablas, estructura (para información temática, libros y *white papers*), referencias (bibliográficas y en notas al pie de página), números

[302] Un control es un ítem en una interfaz, como por ejemplo, un espacio para escribir el nombre, que está, por lo regular, acompañado de una etiqueta que lo identifica.

[303] "Ayer", "hoy" y "mañana" cambian en función del huso horario en que se encuentre el usuario. Algo que está sucediendo en España a las 2 h corresponde en ese momento a "hoy" para el usuario que está en España, pero a "mañana" para el usuario que está en Puerto Rico, por ejemplo.

y medidas, interfaces (comandos y su sintaxis, elementos de la GUI, instrucciones de menú y de navegación, mensajes), cómo escribir para una audiencia diversa (accesibilidad y audiencia internacional), y cómo preparar glosarios e índices. Ya que IBM publica muchísima documentación y documentos informativos, este texto contiene un enfoque principalmente dirigido a esto, pero también contiene secciones dirigidas a la redacción de textos dirigidos a públicos internacionales.

3.2.6.4 Microsoft

Por su enfoque dirigido a la computadora personal y a los aparatos móviles, *Microsoft Manual of Style* (Microsoft Corporation 2012) posiblemente contenga más material útil para un internacionalizador. El primer capítulo recoge el estilo y voz de Microsoft donde se enfatiza "[…] the shift toward a lighter, friendlier tone in Microsoft content, with succinct guidelines for writing in the Microsoft voice […]" (*Ibidem*, p. xx). El segundo está dirigido a crear contenido para la web, mientras que el tercero enfatiza la creación de contenido para un público a nivel mundial. El manual incluye ahora "[…] substantial information about writing for a global audience. 'International considerations' sections throughout the manual call attention to issues of localization, global English, and machine translation […]" (*Idem*). El capítulo que trata sobre accesibilidad explica cómo describir las prestaciones de accesibilidad que presenta el *software* y el *hardware*, además, similar a las consideraciones de internacionalización que menciona a lo largo del texto, también hacen referencia a lo largo del texto a las consideraciones importantes para mejorar la accesibilidad. El capítulo sobre la IU incluye directrices específicas para escribir contenido para distintas plataformas y, además, contiene una lista de comprobación (*checklist*) para redactar texto para la IU. Hay otro capítulo que recoge recomendaciones prácticas relacionadas con problemas de estilo comunes, como por ejemplo, el formato de encabezados y títulos, de listas y de números. Termina con una lista de acrónimos y abreviaturas, directrices sobre gramática y puntuación, y un diccionario técnico con sobre 1000 entradas.

Conclusiones y recomendaciones

A lo largo de nuestra investigación hemos estudiado, desde el punto de vista histórico, la manera en que interactúa la industria de desarrollo de *software* con la industria de la localización, y hemos presentado las maneras, incluidos los *frameworks*, las metodologías, las estrategias, los diagramas y los lenguajes de programación, en que se desarrollan los programas de computadora (objetivos A y B de nuestra investigación; preguntas de investigación A y B). Con esto, hemos determinado el conocimiento y competencias que necesita el localizador para que, como proponemos en este trabajo, pueda servir como mediador durante todo el proceso de desarrollo de *software* y no solo durante la fase final (objetivos C y D de nuestra investigación; preguntas de investigación A y B). También hemos incluido y discutido con enfoque crítico las propuestas de investigación previa que se han realizado en torno a este tema. (objetivos A y B de nuestra investigación; preguntas de investigación A y B) Luego de esto, hemos propuesto una revisión del concepto actual que se tiene del localizador y del papel que juega a lo largo del proceso de internacionalización del *software* multilingüe (objetivos C y D de nuestra investigación; preguntas de investigación A, B y C). Este *nuevo* localizador, a quien hemos bautizado con el nombre de *internacionalizador*, posee las competencias necesarias en cuanto a traducción y mediación intercultural junto con conocimientos informáticos que le permiten participar durante todo el proceso de desarrollo de *software* y no solo en la etapa que tiene que ver con la traducción. A continuación, ofrecemos una serie de recomendaciones profesionales y formativas basadas en el análisis y descripción de las competencias mencionadas y en nuestra experiencia como especialistas en informática y traducción (objetivo de investigación E).

Recomendaciones profesionales y formativas

- Un proyecto de desarrollo de *software* que se cuestione la viabilidad de internacionalizar limitaría las posibilidades de expansión hacia los mercados mundiales. No internacionalizar, como una decisión consciente de un equipo de trabajo de un proyecto, es cerrar las puertas a mercados que podrían expandir los beneficios económicos del programa en desarrollo y ampliar el ciclo de vida del producto.

- El localizador debe conocer las posibles plataformas, metodologías de desarrollo, lenguajes y medios que están disponibles en la actualidad a fin de posibilitar su participación durante el proceso de desarrollo de *software*, no solo como mediador intercultural, sino también como mediador interplataforma e interlenguaje. Sus destrezas no solo como traductor y gestor cultural, sino también como conocedor del lenguaje y del metalenguaje con el que se comunica el programador, de las posibilidades y prestaciones para el desarrollo de programas multilingües existentes, le ayudarán a lograr que el producto quede adecuadamente internacionalizado, a fin de que pueda ser localizado durante el proceso de desarrollo mismo o en algún momento en el futuro. Roturier (2015, p. 185) nos recuerda que los localizadores necesitan estar expuestos a los conceptos clave de desarrollo de *software* para que se sientan más cómodos al enfrentarse a los aspectos técnicos inherentes a la programación durante el proceso de localización. Esto también es importante para los localizadores con más experiencia técnica porque conocer sobre este tema mejora su desempeño y su productividad.
- La localización y la traducción continúan siendo una caja negra y el localizador poco más que un traductor que gestiona cadenas de texto, con lo cual la labor del localizador está totalmente subordinada al proceso de desarrollo de *software* y solo se le considera como participante *ad hoc* en su producto terminado: la traducción. Hace falta establecer propuestas integradoras a nivel académico y dentro de la industria que acerquen o integren al localizador a lo largo de todo el proceso de desarrollo de *software*, a fin de permitirle aportar sus conocimientos como traductor y mediador intercultural durante los procesos que solo manejan los desarrolladores, y facilitar una internacionalización adecuada del *software*. La academia es el espacio óptimo para llevar a cabo proyectos interdisciplinares con importantes elementos de transversalidad que integren las disciplinas de la informática, del desarrollo empresarial y la traducción en la gestión de nuevos desarrollos de *software* y funcionen como campos fértiles para estudiar y optimizar la relación entre ambas ramas de estudio. Las nuevas titulaciones, como por

ejemplo el nuevo Doble Grado en Traducción e Interpretación y Derecho[304] que ofrece la Universidad de Salamanca, son un claro ejemplo de ello.

- La figura del localizador sigue siendo desprofesionalizada e infrautilizada en el proceso de desarrollo de programas informáticos. En un proyecto de desarrollo de *software* las partes implicadas siguen pensando que solo necesitan a un "professional native speaker", en lugar de incluir a expertos en traducción y mediación intercultural. ¿Qué cualifica a una persona como un "hablante nativo profesional"? ¿Qué conocimientos lingüísticos y sobre la cultura podría aportar al desarrollo de *software* un simple "hablante nativo profesional"? La participación de un *profesional de la traducción especializado en localización* no solo aporta conocimiento lingüístico y cultural al proyecto, sino que también permite conformar un equipo con un "efecto multinacional", es decir, un equipo que, sin ser necesariamente internacional, se convierte en tal cuando cuenta con la presencia de uno o más internacionalizadores expertos. Los internacionalizadores permiten que el equipo entienda el impacto de los requisitos de internacionalización sobre el proyecto e integre las necesidades culturales y lingüísticas a lo largo de todo el proceso. Esto puede reducir los costos y retrasos que surgen en el proyecto por no tomar en consideración estos requisitos desde el principio.
- Se debe aumentar la divulgación limitada que tiene este tema, sobre todo desde el punto de vista de la localización, en los espacios que ocupan los programadores de *software*.
- Se deben desarrollar más programas académicos que fomenten la interacción entre los programadores de *software* y los traductores, y estudiar más a fondo cómo pueden colaborar a fin de mejorar la calidad de los productos multilingües que se programen.

Líneas de investigación futuras

- Qué impacto económico tiene la inclusión de un equipo de internacionalizadores en el desarrollo de un proyecto de *software*.

[304] Véase http://exlibris.usal.es/index.php/doble-grado-en-traduccion-e-interpretacion-y-derecho.

- Cómo facilita la labor del internacionalizador la futura localización de un proyecto de *software* adecuadamente internacionalizado.
- Cómo se expande el ciclo de vida del producto y se multiplican las posibilidades de expansión, distribución y penetración en los mercados al considerar e integrar la internacionalización a lo largo del desarrollo de un sistema informático.
- Cómo puede un localizador experto trabajar como parte de un equipo de desarrollo de *software* que se basa en la arquitectura de *software* para desarrollar las aplicaciones.
- Aplicación práctica del modelo de arquitectura de *software* propuesto por Venkatesan (2008) en idiomas bidireccionales.

Conclusión

Desde los inicios de la informática, las grandes empresas como Apple, IBM y Microsoft, entre muchas otras, han probado la importancia de tener presencia en otros mercados. Este no es el caso de las pymes y los desarrolladores más modestos. Si estos prestaran mayor atención a este aspecto de sus productos y servicios, se verían beneficiados con un enorme potencial de crecimiento. Y no solo es que sus productos y servicios estén internacionalizados y localizados, es que estén internacionalizados y localizados siguiendo estándares que proporcionen un alto nivel de calidad, porque lo menos que quiere un cliente es sentir que se le habla (que es lo que las interfaces hacen) con falta de corrección y sin considerar sus gustos, preferencias, convenciones y factores lingüísticos y culturales, e incluso de protocolos de cortesía si lo consideramos más a un nivel discursivo.

Esperamos que esta propuesta para revisar el concepto que tenemos del localizador y de su papel en el proceso de internacionalización de *software* multilingüe logre redefinir su función en esta industria, no solo para mejorar la calidad de los productos multilingües, sino también para fomentar el acceso a estos productos a todos los habitantes de la aldea global.

Bibliografía

Textos citados

A History of OCaml. [en línea], 2014. [consulta: 11 de marzo de 2016]. Disponible en: http://ocaml.org/learn/history.html#FinalQuote.

AASENDEN, J.L., 2015. Delphi for Linux imminent. *Jon Aasenden Pers. Website* [en línea]. [consulta: 20 de enero de 2016]. Disponible en: https://jonlennartaasenden.wordpress.com/2015/09/18/delphi-for-linux-immanent/.

ABUFARDEH, S.O., 2008. *A Framework for the Integration of Internationalization and Localization Activities into the Software Development Process*. Tesis doctoral. Fargo, Dakota del Norte: North Dakota State University.

ABUFARDEH, S.O. y MAGEL, K., 2009. Software Internationalization: Crosscutting Concerns across the Development Lifecycle. *Int. Conf. New Trends Inf. Serv. Sci.* S.l.: s.n., pp. 447-450. DOI 10.1109/NISS.2009.202.

Ada Overview. *Ada Inf. Clear. House* [en línea], 2016. [consulta: 23 de enero de 2016]. Disponible en: http://www.adaic.org/advantages/ada-overview/.

AGAY, A. y VAJHOEJ, A., 1998. A Brief History of FORTRAN/Fortran. *User Notes FORTRAN Program. Open Coop. Pract. Guide* [en línea]. [consulta: 12 de octubre de 2014]. Disponible en: http://www.ibiblio.org/pub/languages/fortran/ch1-1.html.

AGUAYO ARRABAL, N., 2012. El traductor-intérprete en el comercio exterior: ¿realidad o necesidad? *Entrecult. Rev. Trad. Comun. Intercult.*, vol. 5, pp. 57-74. ISSN 1989-5097.

ALLAN, R.A., 2001. *A History of the Personal Computer: The People and the Technology* [en línea]. S.l.: Allan Publishing. [consulta: 26 de enero de 2015]. ISBN 0-9689108-3-1. Disponible en: http://epe.lac-bac.gc.ca/100/200/300/allan_publishing/history_personal_computer/index.html.

ALLIANCE FOR TELECOMMUNICATIONS INDUSTRY SOLUTIONS, 2011. ATIS Telecom Glossary. *ATIS Telecom Gloss.* [en línea]. [consulta: 12 de octubre de 2014]. Disponible en: http://www.atis.org/glossary/default.aspx.

ÁLVAREZ GARCÍA, M. del C., 2015. *El acceso de los traductores e intérpretes al conocimiento experto en materia económica: especialización en comercio exterior* [en línea]. Tesis doctoral. Sevilla: Universidad Pablo de Olavide. [consulta: 28 de mayo de 2016]. Disponible en: https://www.educacion.es/teseo/mostrarRef.do?ref=1098033.

ANACLETO, V.A., 2005. Introducción a la UML 2.0. *epiwiki* [en línea]. [consulta: 8 de febrero de 2016]. Disponible en: http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=15.

ANISZCZYK, C., 2012. TwitterCLDR: Improving Internationalization Support in Ruby. *Twitter Blogs* [en línea]. [consulta: 16 de febrero de 2016]. Disponible en: https://blog.twitter.com/2012/twittercldr-improving-internationalization-support-in-ruby.

APPLE INC., 2013. Introduction to the Apple Style Guide. *Apple Style Guide* [en línea]. [consulta: 5 de abril de 2016]. Disponible en: https://help.apple.com/asg/mac/2013/#apsg1eef9171.

APPLE INC., 2014. About Objective-C. *Mac Dev. Libr.* [en línea]. [consulta: 10 de marzo de 2016]. Disponible en: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html.

APPLE INC., 2015a. About Internationalization and Localization. *Int. Localization Guide* [en línea]. [consulta: 16 de septiembre de 2016]. Disponible en: https://developer.apple.com/library/prerelease/ios/documentation/MacOSX/Conceptual/BPInternational/Introduction/Introduction.html#//apple_ref/doc/uid/10000171i-CH1-SW1.

APPLE INC., 2015b. Localizing Your App. *Int. Localization Guide* [en línea]. [consulta: 16 de septiembre de 2016]. Disponible en: https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPInternational/LocalizingYourApp/LocalizingYourApp.html.

APPLE INC., 2015c. Terminology and Wording. *OS X Hum. Interface Guidel.* [en línea]. [consulta: 4 de abril de 2016]. Disponible en: https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/TerminologyWording.html#//apple_ref/doc/uid/20000957-CH15-SW1.

APPLE INC., 2016a. About Swift. *Swift.org* [en línea]. [consulta: 18 de marzo de 2016]. Disponible en: https://swift.org/about/.

APPLE INC., 2016b. Build Apps for the World. *Int. - Apple Dev.* [en línea]. Disponible en: https://developer.apple.com/internationalization/.

APPLE INC., 2016c. Terminology and Wording. *IOS Hum. Interface Guidel.* [en línea]. [consulta: 4 de abril de 2016]. Disponible en: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/FeedbackCommunication.html#//apple_ref/doc/uid/TP40006556-CH56-SW1.

ASSOCIATION OF LISP USERS, 2016. Lisp Resources. [en línea]. [consulta: 6 de marzo de 2016]. Disponible en: http://www.alu.org/alu/res-lisp#common-lisp.

AVISON, D. y FITZGERALD, G., 2003. Where Now for Development Methodologies? *Commun. ACM*, vol. 46, n.º 1, pp. 79-82.

AVISON, D. y FITZGERALD, G., 2006a. *Information Systems Development: Methodologies, Techniques and Tools*. 4.ª ed. Londres: McGraw-Hill. ISBN 0-07-711417-5.

AVISON, D. y FITZGERALD, G., 2006b. Methodologies for Developing Information Systems: A Historical Perspective. En: D. AVISON, S. ELLIOT, J. KROGSTIE y J. PRIES-HEJE (eds.), *Past Future Inf. Syst. 1976-2006 Beyond*. Boston: Springer, IFIP International Federation for Information Processing, pp. 27-38.

BACKUS, J., 1981. The History of FORTRAN I, II, and III. En: R.L. WEXELBLAT (ed.), *Hist. Program. Lang. I*. Nueva York: ACM New York, pp. 25-74.

BARR, M., 2015. Embedded Systems Glossary. *Barr Group* [en línea]. [consulta: 24 de enero de 2016]. Disponible en: http://www.barrgroup.com/Embedded-Systems/Glossary.

BASS, L., CLEMENTS, P. y KAZMAN, R., 2013. *Software Architecture in Practice*. 3.ª ed. Upper Saddle River: Addison-Wesley. SEI series in software engineering. ISBN 978-0-321-81573-6.

BASS, L., JOHN, B.E. y KATES, J., 2001. Achieving Usability Through Software Architecture. [en línea]. Technical Report. Pittsburg, Pensilvania: Software Engineering Institute, Carnegie Mellon University. [consulta: 2 de noviembre de 2014]. CMU/SEI-2001-TR-005. Disponible en: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5605.

BECK, K., BEEDLE, M., BENNEKUM, A. van, COCKBURN, A., CUNNINGHAM, W., FOWLER, M., GRENNING, J., HIGHSMITH, J., HUNT, A., JEFFRIES, R., KERN, J., MARICK, B., MARTIN, R.C., MELLOR, S., SCHWABER, K., SUTHERLAND, J. y THOMAS, D., 2001a. Manifesto for Agile Software Development. *Manif. Agile Softw. Dev.* [en línea]. [consulta: 14 de agosto de 2016]. Disponible en: http://agilemanifesto.org.

BECK, K., BEEDLE, M., BENNEKUM, A. van, COCKBURN, A., CUNNINGHAM, W., FOWLER, M., GRENNING, J., HIGHSMITH, J., HUNT, A., JEFFRIES, R., KERN, J., MARICK, B., MARTIN, R.C., MELLOR, S., SCHWABER, K., SUTHERLAND, J. y THOMAS, D., 2001b. Principles Behind the Agile Manifesto. *Manif. Agile Softw. Dev.* [en línea]. [consulta: 14 de agosto de 2016]. Disponible en: http://agilemanifesto.org/principles.html.

BEILIS, A., 2012. Boost.Locale. *Boost C Libr.* [en línea]. [consulta: 5 de marzo de 2016]. Disponible en: http://www.boost.org/doc/libs/1_60_0/libs/locale/doc/html/index.html.

BEMER, B., 2000. Colossus - World War II Computer: The First Word Processor. *Thoughts Past Future Comput. Pioneer Bob Bemer* [en línea]. [consulta: 28 de septiembre de 2015]. Disponible en: http://www.bobbemer.com/COLOSSUS.HTM.

BOYER, C., 2004. *The 360 Revolution* [en línea]. Armonk, Nueva York: IBM. [consulta: 18 de octubre de 2015]. Disponible en: http://www-03.ibm.com/systems/tw/resources/system_tw_360revolution_040704.pdf.

BUDD, T., 1993. A Brief Introduction to Smalltalk. *Second ACM SIGPLAN Conf. Hist. Program. Lang.* [en línea]. Nueva York: ACM New York, pp. 367-368. [consulta: 16 de marzo de 2016]. ISBN 0-89791-570-4. DOI 10.1145/154766.155399. Disponible en: http://doi.acm.org/10.1145/154766.155399.

CARAS, J., 2015. The Genie in the Machines. *XRDS*, vol. 22, n.º 2, pp. 32-35. ISSN 1528-4972. DOI 10.1145/2845149.

CARBONELL CORTÉS, O., 1997. Del «conocimiento del mundo» al discurso ideológico: el papel del traductor como un mediador entre culturas. En: E. MORILLAS y J.P. ARIAS (eds.), *El Pap. Trad.* 1.ª ed. Salamanca: Ediciones Colegio de España, Biblioteca de traducción, 2, pp. 59-74. ISBN 978-84-86408-65-7.

CHANDLER, H.M. y DEMING, S.O., 2012. *The Game Localization Handbook*. 2.ª ed. Sudbury, Massachusetts: Jones & Bartlett Learning. ISBN 978-0-7637-9593-1.

CHOW, J., 2015. *The Future of Supercomputers? A Quantum Chip Colder than Outer Space* [en línea]. TED Institute. [consulta: 7 de enero de 2016]. Disponible en: https://youtu.be/VsBuuwGj3zs.

COAD, P. y YOURDON, E., 1991. *Object-Oriented Design*. Nueva Jersey: Prentice Hall.

COLLADO, M., 2013. Lenguajes de guiones (Scripting languages). *Asign. Entornos Program.* [en línea]. [consulta: 10 de agosto de 2016]. Disponible en: http://lml.ls.fi.upm.es/ep/script.pdf.

COMPUTER HISTORY MUSEUM, 2006. Timeline of Computer History. *Comput. Hist. Mus.* [en línea]. [consulta: 25 de enero de 2015]. Disponible en: http://www.computerhistory.org/timeline/?category=cmptr.

COMPUTER HISTORY MUSEUM, 2011a. A SAGE Defense. *Comput. Hist. Mus.* [en línea]. [consulta: 24 de octubre de 2015]. Disponible en: http://www.computerhistory.org/revolution/real-time-computing/6/120.

COMPUTER HISTORY MUSEUM, 2011b. Atanasoff-Berry Computer. *Comput. Hist. Mus.* [en línea]. [consulta: 26 de septiembre de 2015]. Disponible en: http://www.computerhistory.org/revolution/birth-of-the-computer/4/99.

COMPUTER HISTORY MUSEUM, 2011c. DEC's Blockbuster: the PDP-8. *Comput. Hist. Mus.* [en línea]. [consulta: 24 de octubre de 2015]. Disponible en: http://www.computerhistory.org/revolution/minicomputers/11/331.

COMPUTER HISTORY MUSEUM, 2011d. *Fred Brooks, Jr.: Birth of the 360 Project* [en línea]. [consulta: 6 de octubre de 2015]. Disponible en: http://www.computerhistory.org/revolution/mainframe-computers/7/162/2270.

COMPUTER HISTORY MUSEUM, 2011e. Input & Output. *Revolut. First 2000 Years Comput.* [en línea]. [consulta: 25 de enero de 2015]. Disponible en: http://www.computerhistory.org/revolution/input-output/14/347.

COMPUTER HISTORY MUSEUM, 2011f. Intel's Microprocessor. *Comput. Hist. Mus.* [en línea]. [consulta: 4 de enero de 2014]. Disponible en: http://www.computerhistory.org/revolution/digital-logic/12/285.

COMPUTER HISTORY MUSEUM, 2011g. Personal Computers. *Revolut. First 2000 Years Comput.* [en línea]. [consulta: 25 de enero de 2015]. Disponible en: http://www.computerhistory.org/revolution/personal-computers/17/intro.

COMPUTER HISTORY MUSEUM, 2011h. Revolution: The First 2000 Years of Computing. *Comput. Hist. Mus.* [en línea]. [consulta: 12 de octubre de 2014]. Disponible en: http://www.computerhistory.org/revolution.

COMPUTER HISTORY MUSEUM, 2011i. Steve Wozniak: The Homebrew Computer Club and the Apple I. *Comput. Hist. Mus.* [en línea]. [consulta: 5 de enero de 2014]. Disponible en: http://www.computerhistory.org/revolution/personal-computers/17/312/2312.

COMPUTER HISTORY MUSEUM, 2011j. *The Art of Writing Software* [en línea]. [consulta: 12 de octubre de 2014]. Revolution: The First 2000 Years of Computing. Disponible en: http://www.computerhistory.org/revolution/the-art-of-programming/9/357/2216.

COMPUTER HISTORY MUSEUM, 2011k. The First Mainframes. *Comput. Hist. Mus.* [en línea]. [consulta: 2 de octubre de 2015]. Disponible en: http://www.computerhistory.org/revolution/mainframe-computers/7/166.

CONEJERO, J.M., BERG, K.G. van den y HERNÁNDEZ, J., 2007. Un marco conceptual para la formalización de crosscutting en desarrollos orientados a aspectos. En: J.A. JARDINI (ed.), *IEEE Lat. Am. Trans.*, vol. 5, n.º 4, pp. 224-230.

COOPER, A., REIMANN, R. y CRONIN, D., 2007. *About Face 3: The Essentials of Interaction Design*. Indianápolis: Wiley Publishing, Inc.

CRIADO, M.Á., 2014. El ordenador que ayudó a ganar la II Guerra Mundial revive. *El Huffington Post* [en línea]. [consulta: 28 de septiembre de 2015]. Disponible en: http://www.huffingtonpost.es/2014/02/05/colossus-aniversario_n_4728711.html.

DAOUST, N., 2012. *UML Requirements Modeling for Business Analysts: Steps to Modeling Success*. Westfield, Nueva Jersey: Technics Publications, LLC.

DAVIS, G.B., 2006. Information Systems as an Academic Discipline. En: D. AVISON, S. ELLIOT, J. KROGSTIE y J. PRIES-HEJE (eds.), *Past Future Inf. Syst. 1976-2006 Beyond*. Boston: Springer, IFIP International Federation for Information Processing, pp. 11-25.

DENNIS, A., WIXOM, B.H. y TEGARDEN, D., 2009. *Systems Analysis and Design with UML Version 2.0: An Object-Oriented Approach*. 3.ª ed. Hoboken, Nueva Jersey: John Wiley & Sons.

DERESPINIS, F. y INTERNATIONAL BUSINESS MACHINES CORPORATION (eds.), 2012. *The IBM Style Guide: Conventions for Writers and Editors*. Upper Saddle River: IBM Press/Pearson. ISBN 978-0-13-210130-1.

DRAY, S.M. y SIEGEL, D.A., 2006. Melding Paradigms: Meeting the Needs of International Customers Through Localization and User-Centered Design. En: K.J. DUNNE (ed.), *Perspect. Localization*. Ámsterdam; Filadelfia: John Benjamins Publishing Company, American Translators Association Scholarly Monograph Series, pp. 281-307.

DSDM CONSORTIUM, 2014. MoSCoW Prioritisation. [en línea]. [consulta: 27 de noviembre de 2015]. Disponible en: http://www.dsdm.org/dig-deeper/book/10-moscow-prioritisation.

DSDM CONSORTIUM, 2015. What is DSDM. *DSDM Consort.* [en línea]. [consulta: 19 de abril de 2016]. Disponible en: https://www.dsdm.org/what-is-dsdm.

DUNNE, K.J., 2006. A Copernican Revolution: Focusing on the Big Picture of Localization. En: K.J. DUNNE (ed.), *Perspect. Localization*. Ámsterdam; Filadelfia: John Benjamins Publishing Company, American Translators Association Scholarly Monograph Series, pp. 1-11.

ECMA INTERNATIONAL, 2006. *Standard ECMA-334: C# Language Specification* [en línea]. 4.ª ed. Ginebra: s.n. [consulta: 17 de febrero de 2016]. Disponible en: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf.

ECMA INTERNATIONAL, 2016a. History of Ecma. [en línea]. [consulta: 3 de abril de 2016]. Disponible en: http://www.ecma-international.org/memento/history.htm.

ECMA INTERNATIONAL, 2016b. TC49 Programming Languages (Formerly TC39). [en línea]. [consulta: 3 de abril de 2016]. Disponible en: http://www.ecma-international.org/memento/TC49.htm.

ECMA INTERNATIONAL, 2016c. What is Ecma International. [en línea]. [consulta: 3 de abril de 2016]. Disponible en: http://www.ecma-international.org/memento/index.html.

EHN, P., 1988. *Work-Oriented Design of Computer Artifacts*. Tesis doctoral. Estocolmo: Umeå Universitet.

ELRAD, T., FILMAN, R.E. y BADER, A., 2001. Aspect-Oriented Programming: Introduction. *Commun. ACM*, vol. 44, n.º 10, pp. 29-32. ISSN 00010782. DOI 10.1145/383845.383853.

EMBARCADERO TECHNOLOGIES, 2014. Language Overview - RAD Studio. [en línea]. [consulta: 6 de marzo de 2016]. Disponible en: http://docwiki.embarcadero.com/RADStudio/Seattle/en/Language_Overview.

EMBARCADERO TECHNOLOGIES, 2016. RAD Studio 10 Seattle. *Embarcadero Website* [en línea]. [consulta: 18 de abril de 2016]. Disponible en: https://www.embarcadero.com/products/rad-studio.

ESSELINK, B., 2000. *A Practical Guide to Localization*. Ámsterdam: John Benjamins Publishing Company.

ESSELINK, B., 2006. The Evolution of Localization. En: A. PYM, A. PEREKRESTENKO y B. STARINK (eds.), *Transl. Technol. Its Teach. Much Mention Localization*. S.l.: s.n., pp. 21-29.

ETSI, 2013. *Localisation Industry Standards (LIS); Translation Memory eXchange (TMX)* [en línea]. S.l.: European Telecommunications Standards Institute. [consulta: 19 de noviembre de 2015]. Disponible en: http://www.etsi.org/deliver/etsi_gs/LIS/001_099/002/01.04.02_60/gs_LIS002v010402p.pdf.

FAKHROUTDINOV, K., 2016. The Unified Modeling Language (UML). [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: http://www.uml-diagrams.org/.

FITZGERALD, B., 1998. An Empirical Investigation into the Adoption of Systems Development Methodologies. *Inf. Manage.*, vol. 34, n.º 6, pp. 317-328. ISSN 0378-7206. DOI 10.1016/S0378-7206(98)00072-X.

FRANCH, X. y BOTELLA, P., 1998. Putting Non-Functional Requirements into Software Architecture. *Proc. 9th Int. Workshop Softw. Specif. Des.* [en línea]. Washington, DC: IEEE Computer Society, [consulta: 25 de agosto de 2016]. ISBN 0-8186-8439-9. Disponible en: http://dl.acm.org/citation.cfm?id=857205.858291.

FRANKLIN, C., 2015. Fortran: 7 Reasons Why It's Not Dead. *InformationWeek* [en línea], [consulta: 7 de marzo de 2016]. Disponible en: http://www.informationweek.com/software/enterprise-applications/fortran-7-reasons-why-its-not-dead/d/d-id/1321174.

FREE SOFTWARE FOUNDATION, 2015. gettext. *GNU Oper. Syst.* [en línea]. [consulta: 5 de marzo de 2016]. Disponible en: https://www.gnu.org/software/gettext/.

FUENTES LUQUE, A., 2016. *Comunicación personal*. 2016. S.l.: s.n.

GALA, 2016a. What is Globalization? [en línea]. [consulta: 28 de agosto de 2016]. Disponible en: https://www.gala-global.org/language-industry/intro-language-industry/what-globalization.

GALA, 2016b. What is Internationalization? [en línea]. [consulta: 28 de agosto de 2016]. Disponible en: https://www.gala-global.org/language-industry/intro-language-industry/what-internationalization.

GALA, 2016c. What is Localization? [en línea]. [consulta: 28 de agosto de 2016]. Disponible en: https://www.gala-global.org/language-industry/intro-language-industry/what-localization.

GANDOMI, A. y HAIDER, M., 2015. Beyond the Hype: Big Data Concepts, Methods, and Analytics. *Int. J. Inf. Manag.*, vol. 35, n.º 2, pp. 137-144. ISSN 0268-4012. DOI http://dx.doi.org/10.1016/j.ijinfomgt.2014.10.007.

GASSON, S., 1995. The Role of Methodologies in IT-Related Organisational Change. *Proc. BCS Spec. Group Methodol.* Wrexham: s.n.,

GOGUEN, J.A., 1991. The Denial of Error. En: C. FLOYD, H. ZÜLLIGHOVEN, R. BUDDE y R. KEIL-SLAWIK (eds.), *Softw. Dev. Real. Constr.* [en línea]. Secaucus: Springer-Verlag New York, Inc., ISBN 0-387-54349-X. Disponible en: http://swt-www.informatik.uni-hamburg.de/uploads/media/sdrcbook.pdf.

GOOGLE, 2016a. Bidirectionality - Usability. *Google Des. Guidel.* [en línea]. [consulta: 4 de abril de 2016]. Disponible en: https://www.google.com/design/spec/usability/bidirectionality.html#.

GOOGLE, 2016b. Material Design. *Google Des. Guidel.* [en línea]. [consulta: 4 de abril de 2016]. Disponible en: https://www.google.com/design/spec/material-design/introduction.html.

GOOGLE, 2016c. Writing - Style. *Google Des. Guidel.* [en línea]. [consulta: 4 de abril de 2016]. Disponible en: https://www.google.com/design/spec/style/writing.html.

GOSLING, J., JOY, B., STEELE, G.L., BRACHA, G. y BUCKLEY, A., 2015. *The Java Language Specification* [en línea]. S.l.: s.n. [consulta: 8 de marzo de 2016]. ISBN 978-0-13-390069-9. Disponible en: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf.

GOYVAERTS, J., 2015. Welcome. *Regul.-Expressionsinfo* [en línea]. Disponible en: http://www.regular-expressions.info/index.html.

GRAD, B., 2015. Software Industry. *Eng. Technol. Hist. Wiki* [en línea]. [consulta: 7 de septiembre de 2015]. Disponible en: http://ethw.org/index.php?title=Software_Industry&oldid=112940.

GRIES, D., 1978. ALGOL 60 Language Summary. *ACM SIGPLAN Not.* [en línea], vol. 13, n.º 8. [consulta: 26 de enero de 2016]. ISSN 0362-1340. DOI 10.1145/960118.808368. Disponible en: http://doi.acm.org/10.1145/960118.808368.

HASKELL WIKI, 2013. Introduction. *Haskell Program. Lang.* [en línea]. [consulta: 7 de marzo de 2016]. Disponible en: https://wiki.haskell.org/Introduction#What_is_Haskell.3F.

HASKELL WIKI, 2014. Functional Programming. *Haskell Program. Lang.* [en línea]. [consulta: 8 de marzo de 2016]. Disponible en: https://wiki.haskell.org/Functional_programming.

HEATH, S., 2003. *Embedded Systems Design* [en línea]. 2.ª ed. Oxford: Newness. [consulta: 24 de enero de 2016]. ISBN 978-0-08-047756-5. Disponible en: https://books.google.com.pr/books?id=BjNZXwH7HlkC.

HENN, S., 2012. The Night A Computer Predicted The Next President. *NPR.org* [en línea]. [consulta: 2 de octubre de 2015]. Disponible en: http://www.npr.org/sections/alltechconsidered/2012/10/31/163951263/the-night-a-computer-predicted-the-next-president.

HILLARD, N., 2015. *What's New on Internationalization* [en línea]. 2015. S.l.: Apple. [consulta: 1 de agosto de 2015]. Disponible en: http://devstreaming.apple.com/videos/wwdc/2015/227s0ti458qgg/227/227_whats_new_in_internationalization.pdf?dl=1.

HITMILL, 2006. What is Visual Basic.NET. *hitmill.com* [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: http://www.hitmill.com/programming/vbnet/about_vbnet.html.

HOLM, J.A., 2015. 3 Open Source Projects for Modern COBOL Development. *Opensource.com* [en línea]. [consulta: 23 de febrero de 2016]. Disponible en: https://opensource.com/life/15/10/open-source-cobol-development.

HOPPER, G.M., 1981. Keynote Address. En: R.L. WEXELBLAT (ed.), *Hist. Program. Lang. I*. Nueva York: ACM New York, pp. 7-20.

How to Use Tcl 8.1 Internationalization Features. *Tcl Dev. Xchange* [en línea], [sin fecha]. [consulta: 20 de marzo de 2016]. Disponible en: http://www.tcl.tk/doc/howto/i18n.html.

HUDAK, P., HUGHES, J., PEYTON JONES, S. y WADLER, P., 2007. A History of Haskell: Being Lazy with Class. *Proc. Third ACM SIGPLAN Conf. Hist. Program. Lang.* [en línea]. Nueva York: ACM New York, pp. 12-1–12-55. ISBN 978-1-59593-766-7. DOI 10.1145/1238844.1238856. Disponible en: http://doi.acm.org/10.1145/1238844.1238856.

HURLBURT, G., 2015. The Internet of Things… of All Things. *XRDS*, vol. 22, n.º 2, pp. 22-26. ISSN 1528-4972. DOI 10.1145/2845143.

INTEL CORPORATION, 2012. How Intel Makes Chips: Transistors to Transformations. *Intel* [en línea]. [consulta: 2 de noviembre de 2015]. Disponible en: http://www.intel.com/content/www/us/en/history/museum-transistors-to-transformations-brochure.html.

INTERNATIONAL BUSINESS MACHINES, 2003a. 1130 Facts Folder: Programming Systems. [en línea]. [consulta: 15 de marzo de 2016]. Disponible en: https://www-03.ibm.com/ibm/history/exhibits/1130/1130_facts4.html.

INTERNATIONAL BUSINESS MACHINES, 2003b. IBM Archives: IBM System/3. [en línea]. [consulta: 25 de octubre de 2015]. Disponible en: https://www-03.ibm.com/ibm/history/exhibits/rochester/rochester_4008.html.

INTERNATIONAL BUSINESS MACHINES, 2003c. IBM Archives: IBM System/36. [en línea]. [consulta: 25 de octubre de 2015]. Disponible en: https://www-03.ibm.com/ibm/history/exhibits/rochester/rochester_4018.html.

INTERNATIONAL BUSINESS MACHINES, 2003d. IBM Archives: IBM System/38. [en línea]. [consulta: 25 de octubre de 2015]. Disponible en: https://www-03.ibm.com/ibm/history/exhibits/rochester/rochester_4009.html.

INTERNATIONAL BUSINESS MACHINES, 2003e. IBM Archives: SAGE console. [en línea]. [consulta: 24 de octubre de 2015]. Disponible en: http://www-03.ibm.com/ibm/history/exhibits/vintage/vintage_4506VV2216.html.

INTERNATIONAL BUSINESS MACHINES, 2008. z/VSE Operating System - History. [en línea]. [consulta: 21 de octubre de 2015]. Disponible en: http://www-03.ibm.com/systems/z/os/zvse/about/history.html.

INTERNATIONAL BUSINESS MACHINES, 2013a. Character Data Representation Architecture. *IBM Glob.* [en línea]. [consulta: 29 de febrero de 2016]. Disponible en: http://www-01.ibm.com/software/globalization/cdra/index.html.

INTERNATIONAL BUSINESS MACHINES, 2013b. Coded Character Sets: An Overview. *IBM Glob.* [en línea]. [consulta: 29 de febrero de 2016]. Disponible en: http://www-01.ibm.com/software/globalization/topics/charsets/index.html.

INTERNATIONAL BUSINESS MACHINES, 2013c. New Software for Mainframe Apps. [en línea]. [consulta: 23 de febrero de 2016]. Disponible en: http://www-03.ibm.com/press/us/en/pressrelease/41095.wss.

INTERNATIONAL BUSINESS MACHINES, 2013d. Processing Unicode Data in COBOL Applications. *IBM Knowl. Cent.* [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: https://www.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc.char/src/tpc/db2z_processunidatacobol.dita?lang=en.

INTERNATIONAL BUSINESS MACHINES, 2013e. Processing Unicode Data in PL/I Applications. *IBM Knowl. Cent.* [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: https://www.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc.char/src/tpc/db2z_processunidatapli.dita?lang=en.

INTERNATIONAL BUSINESS MACHINES, 2013f. Who Uses Mainframes and Why Do They Do It? [en línea]. [consulta: 14 de julio de 2016]. Disponible en: http://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zconc_whousesmf.htm.

INTERNATIONAL BUSINESS MACHINES, 2014a. IBM Archives: Valuable Resources on IBM's History. *IBM* [en línea]. [consulta: 24 de septiembre de 2014]. Disponible en: http://www-03.ibm.com/ibm/history/index.html.

INTERNATIONAL BUSINESS MACHINES, 2014b. *IBM Highlights 1970-1984* [en línea]. 2014. S.l.: IBM. [consulta: 25 de enero de 2015]. Disponible en: http://www-03.ibm.com/ibm/history/documents/pdf/1970-1984.pdf.

INTERNATIONAL BUSINESS MACHINES, 2015. *IBM z13: The New Possible* [en línea]. [consulta: 14 de septiembre de 2015]. Disponible en: https://www.youtube.com/watch?v=NAFWySXY5L0.

INTERNATIONAL BUSINESS MACHINES, 2016. Rational Developer family. [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: http://www-03.ibm.com/software/products/en/developer.

INTERNATIONAL STANDARDS ORGANIZATION, 2015. Search - ISO. *ISO* [en línea]. [consulta: 19 de noviembre de 2015]. Disponible en: http://www.iso.org/iso/home/search.htm.

IOT COUNCIL, 2015. The Internet of Things. *Internet Things* [en línea]. [consulta: 10 de enero de 2016]. Disponible en: http://www.theinternetofthings.eu/what-is-the-internet-of-things.

IRANI, L. y BRENNAN, M.A., 2001. A-0, FORTRAN, COBOL. *Hist. Program. Lang.* [en línea]. [consulta: 12 de octubre de 2014]. Disponible en: http://www.inf.fu-berlin.de/lehre/SS01/hc/pl/.

ISHIDA, R., 2010. Character Encodings: Essential Concepts. *W3C Int.* [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: https://www.w3.org/International/articles/definitions-characters/.

JANSSEN, C., 2016. *Techopedia.com* [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: https://www.techopedia.com.

JIMÉNEZ-CRESPO, M.Á., 2013. *Translation and Web Localization*. Londres; Nueva York: Routledge. ISBN 978-0-415-64316-0.

JTC 1, 2010. *WD 1539-1 Fortran 2008 (last draft)* [en línea]. 27 abril 2010. S.l.: ISO/IEC. [consulta: 7 de marzo de 2016]. Disponible en: http://www.j3-fortran.org/doc/year/10/10-007.pdf.

JTC 1, 2012. *ISO/IEC TS 29113:2012 Information Technology-Further Interoperability of Fortran with C* [en línea]. 1 diciembre 2012. S.l.: ISO/IEC. [consulta: 7 de marzo de 2016]. Disponible en: https://www.iso.org/obp/ui/#iso:std:iso-iec:ts:29113:ed-1:v1:en.

JVA INITIATIVE COMMITTEE y IOWA STATE UNIVERSITY, 2011. Patent Court Case: Honeywell vs. Sperry Rand. *John Vincent Atanasoff Birth Digit. Comput.* [en línea]. [consulta: 26 de septiembre de 2015]. Disponible en: http://jva.cs.iastate.edu/courtcase.php.

KAY, A.C., 1993. The Early History of Smalltalk. *SIGPLAN Not.*, vol. 28, n.º 3, pp. 69-95. ISSN 0362-1340. DOI 10.1145/155360.155364.

KAZMAN, R. y BASS, L., 1994. Toward Deriving Software Architectures From Quality Attributes. [en línea]. Technical Report. Pittsburg, Pensilvania: Software Engineering Institute, Carnegie Mellon University. [consulta: 5 de julio de 2015]. CMU/SEI-94-TR-10. Disponible en: https://resources.sei.cmu.edu/asset_files/TechnicalReport/1994_005_001_16301.pdf.

KAZMAN, R. y REDDY, K., 1996. An Empirical Study of Architectural Design Operations. [en línea]. S.l.: [consulta: 22 de diciembre de 2015]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.3729&rep=rep1&type=pdf.

KENT, W., 1983. A Simple Guide to Five Normal Forms in Relational Database Theory. *Commun. ACM*, vol. 26, n.º 2, pp. 120-125. ISSN 0001-0782. DOI 10.1145/358024.358054.

KICZALES, GREGOR J., LAMPING, JOHN O., LOPES, CRISTINA V., HUGUNIN, JAMES J., HILSDALE, ERIK A. y BOYAPATI, CHANDRASEKHAR, 2002. Aspect-Oriented Programming [en línea]. 6,467,086. [consulta: 22 de junio de 2015]. 6,467,086. Disponible en: http://www.pmg.lcs.mit.edu/~chandra/publications/aop.html.

KINNERSLEY, B., [sin fecha]. The Language List. [en línea]. [consulta: 26 de enero de 2016]. Disponible en: http://people.ku.edu/~nkinners/LangList/Extras/search.htm.

KOBIE, N., 2015. What is the Internet of Things? *The Guardian* [en línea]. 6 mayo 2015. [consulta: 9 de enero de 2016]. ISSN 0261-3077. Disponible en: http://www.theguardian.com/technology/2015/may/06/what-is-the-internet-of-things-google.

KOZIRIS, N., 2015. Fifty Years of Evolution in Virtualization Technologies: From the First IBM Machines to Modern Hyperconverged Infrastructures. *Proc. 19th Panhellenic Conf. Inform.* [en línea]. Nueva York: ACM New York, pp. 3-4. ISBN 978-1-4503-3551-5. DOI 10.1145/2801948.2802039. Disponible en: http://doi.acm.org/10.1145/2801948.2802039.

KURTZ, T.E., 1981. BASIC Session. *Hist. Program. Lang. I* [en línea]. Nueva York: ACM New York, pp. 515-537. ISBN 0-12-745040-8. Disponible en: http://doi.acm.org/10.1145/800025.1198404.

LANDER, R., 2015. Announcing .NET Core and ASP.NET 5 RC. *NET Blog* [en línea]. [consulta: 18 de enero de 2016]. Disponible en: http://blogs.msdn.com/b/dotnet/archive/2015/11/18/announcing-net-core-and-asp-net-5-rc.aspx.

LANE, T., 1990. Studying Software Architecture Through Design Spaces and Rules. [en línea]. Pittsburg, Pensilvania: Software Engineering Institute, Carnegie Mellon University. [consulta: 23 de junio de 2015]. CMU/SEI-90-TR-018. Disponible en: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11217.

LÉVÉNEZ, É., 2015. Computer Languages History. [en línea]. [consulta: 26 de enero de 2016]. Disponible en: http://www.levenez.com/lang/.

LEWIS, T.G., 1978. BASIC Language Summary. *SIGPLAN Not.*, vol. 13, n.º 8, pp. 101-102. ISSN 0362-1340. DOI 10.1145/960118.808375.

LINDHOLM, T., YELLIN, F., BRACHA, G. y BUCKLEY, A., 2015. *The Java Virtual Machine Specification* [en línea]. S.l.: s.n. [consulta: 8 de marzo de 2016]. Java. ISBN 978-0-13-326044-1. Disponible en: https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf.

LOMMEL, A., 2007. *The Globalization Industry Primer: An Introduction to Preparing your Business and Products for Success in International Markets* [en línea]. S.l.: The Localization Industry Standards Association (LISA). [consulta: 5 de abril de 2013]. Disponible en: http://www.acclaro.com/assets/files/downloads/whitepapers/lisa_globalization_primer.pdf.

MAIL ONLINE, 2007. Beaten by the Germans: Colossus Loses Code-Cracking Race. [en línea]. [consulta: 28 de septiembre de 2015]. Disponible en: http://www.dailymail.co.uk/sciencetech/article-494225/Beaten-Germans-Colossus-loses-code-cracking-race.html.

MATA PASTOR, M., 2016. *Una propuesta dinámica para la integración de la localización en la formación de traductores*. Tesis doctoral. Granada: Universidad de Granada.

MAX VISUAL BASIC, 2010. History of Visual Basic. *Max Vis. Basic Resour. Vis. Basic Program.* [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: http://www.max-visual-basic.com/history-of-visual-basic.html.

MCCARTHY, J., 1981. History of LISP. En: R.L. WEXELBLAT (ed.), *Hist. Program. Lang. I* [en línea]. Nueva York: ACM New York, pp. 173-185. ISBN 0-12-745040-8. Disponible en: http://doi.acm.org/10.1145/800025.1198360.

MCJONES, P., 2014. Xerox Alto Source Code. *Softw. Gems Comput. Hist. Mus. Hist. Source Code Ser.* [en línea]. [consulta: 25 de enero de 2015]. Disponible en: http://www.computerhistory.org/atchm/xerox-alto-source-code/.

MICROSOFT, 2009. The History of Microsoft. *Channel 9* [en línea]. [consulta: 26 de enero de 2015]. Disponible en: http://channel9.msdn.com/Series/History.

MICROSOFT, 2016a. Cross Platform Mobile Development. *Vis. Studio* [en línea]. [consulta: 3 de abril de 2016]. Disponible en: https://www.visualstudio.com/features/mobile-app-development-vs.

MICROSOFT, 2016b. Tools for Windows Apps and Games. *Vis. Studio* [en línea]. [consulta: 3 de abril de 2016]. Disponible en: https://www.visualstudio.com/features/windows-apps-games-vs.

MICROSOFT, 2016c. Visual C++. *Vis. Studio* [en línea]. [consulta: 3 de abril de 2016]. Disponible en: https://www.visualstudio.com/features/cplusplus.

MICROSOFT CORPORATION (ed.), 2012. *Microsoft Manual of Style*. 4.ª ed. Redmond, Washington: Microsoft Press. ISBN 978-0-7356-4871-5.

MICROSOFT DEVELOPER NETWORK, 2009. *Microsoft Application Architecture Guide* [en línea]. 2.ª ed. S.l.: s.n. [consulta: 21 de junio de 2015]. Microsoft Patterns & Practices. Disponible en: https://msdn.microsoft.com/en-us/library/ff650706.aspx.

MICROSOFT DEVELOPER NETWORK, 2014a. Code Pages. *Microsoft Dev. Netw.* [en línea]. [consulta: 4 de junio de 2014]. Disponible en: http://msdn.microsoft.com/en-us/library/dd317752(v=vs.85).aspx.

MICROSOFT DEVELOPER NETWORK, 2014b. Colección de terminología de Microsoft. *Microsoft Portal Lingüíst.* [en línea]. [consulta: 4 de junio de 2014]. Disponible en: http://www.microsoft.com/Language/es-es/Terminology.aspx.

MICROSOFT DEVELOPER NETWORK, 2014c. Locales and Language. *Microsoft Dev. Netw.* [en línea]. [consulta: 4 de junio de 2014]. Disponible en: http://msdn.microsoft.com/en-us/library/dd318716(v=vs.85).aspx.

MICROSOFT DEVELOPER NETWORK, 2016. Overview of the .Net Framework. [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx.

MILINKOVICH, M., 2005. About the Eclipse Foundation. [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: http://www.eclipse.org/org/.

MILLIGAN, K., 2014a. Enhancing IBM i Applications to Reach Global Markets. [en línea]. S.l. [consulta: 15 de marzo de 2016]. Disponible en: https://public.dhe.ibm.com/partnerworld/pub/pdf/courses/c596.pdf.

MILLIGAN, K., 2014b. Making Enterprise Applications Globally Available. [en línea]. S.l. [consulta: 15 de marzo de 2016]. Disponible en: http://public.dhe.ibm.com/partnerworld/pub/pdf/courses/9dce.pdf.

MONO PROJECT, 2016a. Home. *Mono Proj.* [en línea]. [consulta: 30 de marzo de 2016]. Disponible en: http://www.mono-project.com/.

MONO PROJECT, 2016b. The Mono Runtime. *Mono Proj.* [en línea]. [consulta: 3 de abril de 2016]. Disponible en: http://www.mono-project.com/docs/advanced/runtime/.

MONODEVELOP PROJECT, 2016. MonoDevelop. [en línea]. [consulta: 18 de enero de 2016]. Disponible en: http://www.monodevelop.com.

MUNDARGI, K., 2014. History of C Programming language. *PEOI* [en línea]. [consulta: 15 de febrero de 2016]. Disponible en: http://www.peoi.org/Courses/Coursesen/cprog/fram1.html.

MURTAZA, M. y SHWAN, A., 2012. *Guidelines for Multilingual Software Development* [en línea]. Tesis de máster. Gotemburgo: Universidad de Gotemburgo. [consulta: 5 de abril de 2013]. Disponible en: http://hdl.handle.net/2077/30066.

NIGRO, M., 2014. Language. *Brain Games*. Vídeo. National Geographic. Temporada 4, Episodio 3.

NORMAN, D.A., 2013. *The Design of Everyday Things*. Nueva York: Basic Books.

OASIS, 2008. What is XLIFF? *OASIS XML Localis. Interchange File Format TC* [en línea]. [consulta: 19 de noviembre de 2015]. Disponible en: https://www.oasis-open.org/committees/xliff/faq.php#WhatIsXLIFF.

OBJECT MANAGEMENT GROUP, INC., 2016. Home Page. *Welcome UML Web Site* [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: www.uml.org.

O'HAGAN, M. y ASHWORTH, D., 2002. *Translation-Mediated Communication in a Digital World: Facing the Challenges of Globalization and Localization*. Cleveland: Multilingual Matters. Topics in Translation, 23. ISBN 1-85359-580-2.

O'HAGAN, M. y MANGIRON, C. (eds.), 2013. *Game Localization: Translating for the Global Digital Entertainment Industry*. Ámsterdam; Filadelfia: John Benjamins Publishing Company. Benjamins Translation Library, 106. ISBN 978-90-272-2457-6.

OLIF CONSORTIUM, 2008. The Open XML Language Standard Data. [en línea]. [consulta: 19 de noviembre de 2015]. Disponible en: http://www.olif.net/.

ORACLE CORPORATION, 2016. An Introduction to NetBeans. [en línea]. [consulta: 3 de abril de 2016]. Disponible en: https://netbeans.org/about/index.html.

ORACLE TECHNOLOGY NETWORK, 2015a. Java EE at a Glance. [en línea]. [consulta: 8 de marzo de 2016]. Disponible en: http://www.oracle.com/technetwork/java/javaee/overview/index.html.

ORACLE TECHNOLOGY NETWORK, 2015b. Java Platform, Micro Edition (Java ME). [en línea]. [consulta: 8 de marzo de 2016]. Disponible en: http://www.oracle.com/technetwork/java/embedded/javame/index.html.

ORACLE TECHNOLOGY NETWORK, 2016. Java SE at a Glance. [en línea]. [consulta: 8 de marzo de 2016]. Disponible en: http://www.oracle.com/technetwork/java/javase/overview/index.html.

OUSTERHOUT, J., [sin fecha]. History of Tcl. *Tcl Dev. Xchange* [en línea]. [consulta: 19 de marzo de 2016]. Disponible en: http://www.tcl-lang.org/about/history.html.

PARIS, J. y GANTNER, S., 2013. Happy Free RPG Day! *IBM Syst. Mag.* [en línea], Disponible en: http://www.ibmsystemsmag.com/ibmi/developer/rpg/free_rpg/?page=1.

PARKER, J.C. y DUFFIN, J.M., 1990. ENIAC Patent Trial. *Penn Univ. Arch. Rec. Cent.* [en línea]. [consulta: 26 de septiembre de 2015]. Disponible en: http://www.archives.upenn.edu/faids/upd/eniactrial/upd8_10.html.

PATCH, N., 2014. Unicode Beyond Just Characters: Localization with the CLDR. [en línea]. S.l. [consulta: 2 de febrero de 2016]. Disponible en: https://youtu.be/DcPpUnlENAs.

PHILLIPS, A. y DAVIS, M. (eds.), 2009. *BCP 47: Tags for Identifying Languages* [en línea]. septiembre 2009. S.l.: IETF. [consulta: 31 de enero de 2016]. Disponible en: http://tools.ietf.org/html/bcp47.

POSTEL, J., REKHTER, Y. y LI, T., 1995. *BCP 1: Best Current Practices* [en línea]. agosto 1995. S.l.: IETF. [consulta: 31 de enero de 2016]. Disponible en: https://tools.ietf.org/html/rfc1818.

PROFFITT, B., 2013. What APIs Are and Why They're Important. *ReadWrite* [en línea]. [consulta: 13 de febrero de 2016]. Disponible en: http://readwrite.com/2013/09/19/api-defined.

PUGH, E.W., 1995. *Building IBM: Shaping an Industry and its Technology*. Cambridge: MIT Press. ISBN 0-262-16147-8.

PUGH, E.W., 2015. IBM System/360. *Eng. Technol. Hist. Wiki* [en línea]. [consulta: 7 de septiembre de 2015]. Disponible en: http://ethw.org/index.php?title=IBM_System/360&oldid=112932.

RADIN, G., 1978. The Early History and Characteristics of PL/I. *SIGPLAN Not.*, vol. 13, n.º 8, pp. 227-241. ISSN 0362-1340. DOI 10.1145/960118.808389.

REAL ACADEMIA ESPAÑOLA, 2014. *Diccionario de la lengua española* [en línea]. 23.ª ed. Madrid: Espasa Libros, S.L.U. [consulta: 25 de agosto de 2016]. ISBN 978-84-670-4189-7. Disponible en: http://dle.rae.es/.

RENKEMA, T.J.W. y BERGHOUT, E.W., 1997. Methodologies for Information Systems Investment Evaluation at the Proposal Stage: A Comparative Review. *Inf. Softw. Technol.*, vol. 39, n.º 1, pp. 1-13. ISSN 09505849. DOI 10.1016/0950-5849(96)85006-3.

RITCHIE, D.M., 1996. The Development of the C Programming Language. En: T.J. BERGIN Jr. y R.G. GIBSON Jr. (eds.), *Hist. Program. Lang. II* [en línea]. Nueva York: ACM New York, pp. 671-698. [consulta: 15 de febrero de 2016]. ISBN 0-201-89502-1. Disponible en: http://doi.acm.org/10.1145/234286.1057834.

ROTURIER, J., 2015. *Localizing Apps*. Abingdon; Nueva York: Routledge. Translation Practices Explained. ISBN 978-1-138-80358-9.

ROY, D., VEYS, N. y LAFFRA, C., 2007. FAQ What is Eclipse? [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: http://wiki.eclipse.org/FAQ_What_is_Eclipse%3F.

RUIZ, F. y GONZÁLEZ HARBOUR, M., 2009. *Diseño de software* [en línea]. 2 noviembre 2009. S.l.: s.n. [consulta: 21 de junio de 2015]. Disponible en: http://www.ctr.unican.es/asignaturas/is1/is1-t04-trans.pdf.

SALOMON, D., 1993. *Assemblers and Loaders* [en línea]. West Sussex: Ellis Horwood Ltd. [consulta: 14 de febrero de 2016]. Ellis Horwood Series in Computers and Their Applications. Disponible en: http://www.davidsalomon.name/assem.advertis/AssemAd.html.

SAMMET, J.E., 1981. The Early History of COBOL. En: R.L. WEXELBLAT (ed.), *Hist. Program. Lang. I*. Nueva York: ACM New York, pp. 199-243.

SAXENIAN, A., 1996. *Regional Advantage: Culture and Competition in Silicon Valley and Route 128*. S.l.: Harvard University Press.

SCHERER, M., 2000. A Brief Introduction to Code Pages and Unicode: How Unicode Relates to Code Pages, Character Sets, and Encoding. *IBM Dev.* [en línea]. [consulta: 16 de febrero de 2016]. Disponible en: http://www.ibm.com/developerworks/library/ws-codepages/.

SEARLE, S.J., 2004. A Brief History of Character Codes in North America, Europe, and East Asia. *TRON Web* [en línea]. [consulta: 14 de noviembre de 2015]. Disponible en: http://tronweb.super-nova.co.jp/characcodehist.html.

SEREBRENIK, A., 2014. Software Architecture: Domain-Specific Software Architecture and Architectural Patterns. [en línea]. [consulta: 17 de junio de 2015]. Disponible en: http://www.win.tue.nl/~aserebre/2IW80/2013-2014/A2%20-%20DSSA%20Arch%20patterns.pdf.

SHIELDS, A., 2014. Why IBM's Mainframe Solutions are the Company's Moat. *Mark. Realist* [en línea]. [consulta: 23 de octubre de 2015]. Disponible en: http://marketrealist.com/2014/07/imbs-mainframe-solutions-companys-moat/.

SIMS, G., 2015. Google Play Store vs the Apple App Store: By the Numbers (2015). *Android Auth.* [en línea]. [consulta: 9 de febrero de 2016]. Disponible en: http://www.androidauthority.com/google-play-store-vs-the-apple-app-store-601836/.

SJ, 2015. The 7 Ways of Counting Characters. *LINE Eng. Blog* [en línea]. [consulta: 22 de enero de 2016]. Disponible en: http://developers.linecorp.com/blog/?p=3473.

SMITHSONIAN INSTITUTE, 2014. COBOL. *Natl. Mus. Am. Hist.* [en línea]. Disponible en: http://americanhistory.si.edu/cobol/introduction.

SPOLSKY, J., 2003. The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!). *Joel Softw.* [en línea]. [consulta: 14 de noviembre de 2015]. Disponible en: http://www.joelonsoftware.com/articles/Unicode.html.

STROUSTRUP, B., 1993. A History of C++: 1979-1991. *SIGPLAN Not.*, vol. 28, n.º 3, pp. 271-297. ISSN 0362-1340. DOI 10.1145/155360.155375.

SWARTZLANDER, E.E., 2015. Inventing the Computer. *Eng. Technol. Hist. Wiki* [en línea]. [consulta: 7 de septiembre de 2015]. Disponible en: http://ethw.org/index.php?title=Inventing_the_Computer&oldid=112927.

THAKUR, D., 2015. History of C? Why We Use C Programming Language. *Comput. Notes* [en línea]. [consulta: 15 de febrero de 2016]. Disponible en: http://ecomputernotes.com/what-is-c/basic-of-c-programming/a-brief-history-of-c-why-we-use-c-programming-language.

THE LOCALIZATION INSTITUTE, 2012. Terminology. *Localization Inst. Inc* [en línea]. [consulta: 4 de junio de 2014]. Disponible en: http://www.localizationinstitute.com/switchboard.cfm?page=terminology.

THE NATIONAL MUSEUM OF COMPUTING, 2013. *Uncovering Colossus* [en línea]. Bletchley Park: The National Museum of Computing. [consulta: 27 de septiembre de 2015]. Disponible en: https://youtu.be/Yl6pK1Z7B5Q.

THE NATIONAL MUSEUM OF COMPUTING, 2015. The Colossus Gallery. [en línea]. [consulta: 28 de septiembre de 2015]. Disponible en: http://www.tnmoc.org/explore/colossus-gallery.

THEJENDRA, B.., 2014. *Disaster Recovery and Business Continuity: A Quick Guide for Organisations and Business Managers* [en línea]. 3.ª ed. Ely, Cambridgeshire: IT Governance Publishing. [consulta: 24 de octubre de 2015]. ISBN 978-1-84928-539-1. Disponible en: http://www.books24x7.com/marc.asp?bookid=62285.

TIOBE SOFTWARE, 2016. TIOBE Index for April 2016. [en línea]. [consulta: 12 de abril de 2016]. Disponible en: http://www.tiobe.com/tiobe_index.

TORRES DEL REY, J., 2016. *Comunicación personal*. 2016. S.l.: s.n.

UNICODE CONSORTIUM, 2005. Common Locale Data Repository (CLDR) Project. *CLDR - Unicode Common Locale Data Repos.* [en línea]. [consulta: 30 de enero de 2016]. Disponible en: http://www.unicode.org/L2/L2005/05177-cldr-sched.html.

UNICODE CONSORTIUM, 2009. Index Characters. *CLDR - Unicode Common Locale Data Repos.* [en línea]. [consulta: 31 de enero de 2016]. Disponible en: http://cldr.unicode.org/development/development-process/design-proposals/index-characters.

UNICODE CONSORTIUM, 2013. About the Unicode Standard. *Unicode Consort.* [en línea]. [consulta: 4 de junio de 2014]. Disponible en: http://www.unicode.org/standard/standard.html.

UNICODE CONSORTIUM, 2015a. Acknowledgments. *CLDR - Unicode Common Locale Data Repos.* [en línea]. [consulta: 30 de enero de 2016]. Disponible en: http://cldr.unicode.org/index/acknowledgments.

UNICODE CONSORTIUM, 2015b. CLDR Features. *CLDR - Unicode Common Locale Data Repos.* [en línea]. [consulta: 4 de febrero de 2016]. Disponible en: http://cldr.unicode.org/cldr-features.

UNICODE CONSORTIUM, 2015c. Unicode CLDR Project. *CLDR - Unicode Common Locale Data Repos.* [en línea]. [consulta: 30 de enero de 2016]. Disponible en: http://cldr.unicode.org/.

UNICODE CONSORTIUM, 2016. ICU Home Page. *ICU - Int. Compon. Unicode* [en línea]. [consulta: 16 de febrero de 2016]. Disponible en: http://site.icu-project.org/.

UNISYS, 2008. Internationalization. *Rep. Program Gener. RPG Program. Ref. Man.* [en línea]. S.l.: s.n., pp. 5-1-106. [consulta: 16 de marzo de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-15.0/pdf/86000544-103.pdf.

UNISYS, 2009. Internationalization. *Pascal Program. Ref. Man.* [en línea]. S.l.: s.n., pp. 15-1-138. [consulta: 11 de septiembre de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-14.0/pdf/86000080-103.pdf.

UNISYS, 2011. Internationalization. *C Program. Ref. Man.* [en línea]. S.l.: s.n., pp. E-1 a E-92. [consulta: 11 de septiembre de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-16.0/pdf/86002268-205.pdf.

UNISYS, 2012a. Internationalization. *COBOL ANSI-74 Program. Ref. Man.* [en línea]. S.l.: s.n., pp. 16-1-138. [consulta: 20 de marzo de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-14.0/pdf/86000296-207.pdf.

UNISYS, 2012b. Internationalization. *COBOL ANSI-85 Program. Ref. Man.* [en línea]. S.l.: s.n., pp. 16-1-174. [consulta: 20 de marzo de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-14.0/pdf/86001518-312.pdf.

UNISYS, 2013a. Internationalization. *ALGOL Program. Ref. Man.* [en línea]. S.l.: s.n., pp. 10-1-118. [consulta: 26 de enero de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-15.0/pdf/86000098-511.pdf.

UNISYS, 2013b. Internationalization Service Library. *Clear. OS 2200 Softw. Prod. Cat.* [en línea]. S.l.: s.n., pp. 11-14. [consulta: 20 de marzo de 2016]. Disponible en: https://public.support.unisys.com/2200/docs/cp14.0/pdf/78505252-017.pdf.

UNISYS, 2013c. *MultiLingual System Administration, Operations, and Programming Guide* [en línea]. S.l.: s.n. [consulta: 26 de enero de 2016]. Disponible en: https://public.support.unisys.com/aseries/docs/clearpath-mcp-15.0/pdf/86000098-511.pdf.

UNIVERSITY OF PENNSYLVANIA SEAS, 2013. ENIAC: Celebrating Penn Engineering History. [en línea]. [consulta: 29 de septiembre de 2015]. Disponible en: https://www.seas.upenn.edu/about-seas/eniac/index.php.

VENKATESAN, V.P., 2008. *ARMMS an Architectural Reference Model for Multilingual Software* [en línea]. Tesis doctoral. Pondicherry, India: Pondicherry University. [consulta: 5 de abril de 2013]. Disponible en: http://dspace.pondiuni.edu.in/xmlui/handle/pdy/441.

VEYS, N. y LAFFRA, C., 2006. FAQ Where did Eclipse come from? [en línea]. [consulta: 29 de marzo de 2016]. Disponible en: https://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F.

VISUAL PARADIGM, 2015. Data Flow Diagram with Examples - Food Ordering System. [en línea]. [consulta: 22 de marzo de 2016]. Disponible en: https://www.visual-paradigm.com/tutorials/data-flow-diagram-example-food-ordering-system.jsp.

VOWELS, R., 2010. Frequently Asked Questions (and their answers) about PL/I. [en línea]. [consulta: 15 de marzo de 2016]. Disponible en: http://members.dodo.com.au/~robin51/pli_faq.htm.

WACHENCHAUZER, R., MANTEROLA, M., CURIA, M., MEDRANO, M. y PAEZ, N., 2014. Persistencia de datos. *Algoritm. Program. Con Python* [en línea]. S.l.: s.n., [consulta: 30 de noviembre de 2015]. Disponible en: http://librosweb.es/libro/algoritmos_python/capitulo_11/persistencia_de_datos.html.

What is OCaml. [en línea], 2014. [consulta: 11 de marzo de 2016]. Disponible en: http://ocaml.org/learn/description.html.

WIDEBURG, L., 1998. Early Years of Unicode. *Unicode Consort.* [en línea]. [consulta: 14 de noviembre de 2015]. Disponible en: http://www.unicode.org/history/earlyyears.html.

WIKIPEDIA, 2016. List of Programming Languages. *Wikipedia* [en línea]. [consulta: 20 de enero de 2016]. Disponible en: https://en.wikipedia.org/w/index.php?title=List_of_programming_languages&oldid=700728967.

WINOGRAD, T., 1995. Heidegger and the Design of Computer Systems. En: A. FEENBERG y A. HANNAY (eds.), *Technol. Polit. Knowl.* Bloomington: Indiana University Press, The Indiana series in the philosophy of technology, pp. 108-127. ISBN 0-253-32154-9.

YOUNG, E.L., 2001. *A Framework for the Integration of Internationalization into the Software Development Process* [en línea]. Tesis de máster. Dakota del Sur: Universidad de Dakota del Sur. [consulta: 9 de marzo de 2015]. Disponible en: https://web.archive.org/web/20090310130414/https://www.usd.edu/csci/research/theses/graduate/sp2001/eyoung.pdf.

YUE, T., 2011. History of Pascal. [en línea]. [consulta: 11 de marzo de 2016]. Disponible en: http://www.taoyue.com/tutorials/pascal/history.html.

ZENTGRAF, D.C., 2015. What Every Programmer Absolutely, Positively Needs To Know About Encodings And Character Sets To Work With Text. *Kunststube* [en línea]. [consulta: 14 de noviembre de 2015]. Disponible en: http://kunststube.net/encoding/.

Referencias

ABUFARDEH, S.O. y MAGEL, K., 2008. Culturalization of Software Architecture: Issues and Challenges. Proceedings of the 2008 International Conference on Computer Science and Software Engineering [en línea]. Washington, DC: IEEE Computer Society, pp. 436-439. ISBN 978-0-7695-3336-0. DOI 10.1109/CSSE.2008.1414. Disponible en: http://dx.doi.org/10.1109/CSSE.2008.1414.

ABUFARDEH, S.O. y MAGEL, K., 2010. The Impact of Global Software Cultural and Linguistic Aspects on Global Software Development process (GSD): Issues and

challenges. 4th International Conference on New Trends in Information Science and Service Science (NISS). S.l.: s.n., pp. 133-138.

ALBA, D., 2015. Why on Earth Is IBM Still Making Mainframes? Wired [en línea]. [consulta: 23 de octubre de 2015]. Disponible en: http://www.wired.com/2015/01/z13-mainframe/.

ALEMÁN FLORES, M., 2005. El proceso de desarrollo de software. En: D. REINEKE (ed.), Traducción y localización: mercado, gestión y tecnologías. Las Palmas de Gran Canaria: Anroart Ediciones, pp. 21-44.

ALLEN, J.D., ANDERSON, D., BECKER, J., COOK, R., DAVIS, M., EDBERG, P., EVERSON, M., FREYTAG, A., IANCU, L., ISHIDA, R., JENKINS, J.H., LUNDE, K., MCGOWAN, R., MOORE, L., MULLER, E., PHILLIPS, A., POURNADER, R., SUIGNARD, M. y WHISTLER, K. (eds.), 2014. *The Unicode Standard Version 7.0 - Core Specification* [en línea]. Mountain View, California: Unicode Consortium. [consulta: 5 de abril de 2015]. ISBN 978-1-936213-09-2. Disponible en: http://www.unicode.org/versions/Unicode7.0.0/UnicodeStandard-7.0.pdf.

ALOSTATH, J.M., ALMOUMEN, S. y ALOSTATH, A.B., 2009. Identifying and Measuring Cultural Differences in Cross-Cultural User-Interface Design. En: N. AYKIN (ed.), Internationalization, Design and Global Development. Berlín: Springer, pp. 3-12.

ALSHAIKH, Z., MOSTAFA, S., WANG, X. y HE, S., 2015. An Empirical Study on the Status of Software Localization in Open Source Projects. *International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Pittsburg, Pensilvania: s.n.,

ÁLVAREZ MARTÍN-NIETO, T., 2015. La importancia del idioma y la traducción en la internacionalización de la web. PuroMarketing - Marketing, Publicidad, Negocios y Social Media en Español [en línea]. [consulta: 28 de mayo de 2016]. Disponible en: http://www.puromarketing.com/14/22927/importancia-idioma-traduccion-internacionalizacion-web.html.

ANDERSEN, J.G., AMIR, M., SCHMALZ, W., URSULIS, A. y AHN, J., 1997. *Speak the Right Language with Your AS/400 System* [en línea]. Rochester, Nueva York: IBM Redbooks. Disponible en: http://www.redbooks.ibm.com/abstracts/sg242154.html?Open.

ANDROID DEVELOPERS, 2015. Localization Checklist. [en línea]. [consulta: 1 de agosto de 2015]. Disponible en: https://developer.android.com/distribute/tools/localization-checklist.html.

ANDROID DEVELOPERS, [sin fecha]. Building a Flexible UI. [en línea]. [consulta: 4 de abril de 2016 a]. Disponible en: http://developer.android.com/training/basics/fragments/fragment-ui.html.

ANDROID DEVELOPERS, [sin fecha]. Localization Checklist. [en línea]. [consulta: 4 de abril de 2016 b]. Disponible en: http://developer.android.com/distribute/tools/localization-checklist.html.

ANDROID DEVELOPERS, [sin fecha]. String Resources | Android Developers. [en línea]. [consulta: 4 de abril de 2016 c]. Disponible en: http://developer.android.com/guide/topics/resources/string-resource.html#Plurals.

APPLE INC., 2015. Internationalizing Your Code. iOS Developer Library [en línea]. [consulta: 4 de febrero de 2016]. Disponible en: https://developer.apple.com/library/ios/documentation/MacOSX/Conceptual/BPInternational/InternationalizingYourCode/InternationalizingYourCode.html#//apple_ref/doc/uid/10000171i-CH4-SW1.

AREVALILLO DOVAL, J.J., 2013. La traducción automática en las empresas de traducción. Tradumàtica: tecnologies de la traducció, n.º 10, pp. 179-184. ISSN 1578-7559.

AUSTERMÜHL, F., 2001. *Electronic Tools for Translators*. Mánchester: St. Jerome Publishing. Translation Practices Explained. ISBN 1-900650-34-7.

AUSTERMÜHL, F., 2013. Future (and Not-So-Future) Trends in the Teaching of Translation Technology. Tradumàtica: tecnologies de la traducció, n.º 11, pp. 326-337. ISSN 1578-7559.

AYKIN, N. (ed.), 2009. *Internationalization, Design and Global Development*. Berlín: Springer. ISBN 978-3-642-02766-6.

BACHMANN, F. y CLEMENTS, P.C., 2005. Variability in Software Product Lines. [en línea]. Technical Report. Pittsburg, Pensilvania: Software Engineering Institute, Carnegie Mellon University. CMU/SEI-2005-TR-012. Disponible en: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7675.

BARRAZA, F., 2012. *Modelado y Diseño de Arquitectura de Software: Atributos de Calidad y Arquitectura de Software* [en línea]. 13 abril 2012. S.l.: s.n. [consulta: 22 de junio de 2015]. Disponible en: http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:s3_atributoscalidad.pdf.

BASKERVILLE, R., PRIES-HEJE, J. y MADSEN, S., 2011. Post-Agility: What Follows a Decade of Agility? Information and Software Technology, vol. 53, n.º 5, pp. 543-555. DOI http://dx.doi.org/10.1016/j.infsof.2010.10.010.

BBC, 2008. Celebrating the UK's Computer Pioneers. BBC [en línea]. 24 julio 2008. [consulta: 27 de septiembre de 2015]. Disponible en: http://news.bbc.co.uk/2/hi/technology/7521868.stm.

BENINATTO, R., 2002. The Translation and Localization Market. En: D. REINEKE (ed.), Traducción y localización: mercado, gestión y tecnologías. S.l.: s.n., pp. 11-19.

BENNETT, S., MCROBB, S. y FARMER, R., 2007. *Análisis y diseño orientado a objetos de sistemas usando UML*. 3.ª ed. Aravaca: McGraw-Hill/Interamericana de España.

BERNAL MERINO, M., 2006. On the Translation of Video Games. *JoSTrans*, n.º 6, pp. 22-36. ISSN 1740-357X.

BETTELS, J. y BISHOP, F.A., 1993. Unicode: A Universal Character Code. En: J.C. BLAKE (ed.), Digital Technical Journal [en línea], vol. 5, n.º 3. [consulta: 23 de enero de 2015]. Disponible en: http://www.hpl.hp.com/hpjournal/dtj/vol5num3/vol5num3art2.pdf.

BIAU GIL, J.R. y PYM, A., 2006. Technology and Translation (a Pedagogical Overview). En: A. PYM, A. PEREKRESTENKO y B. STARINK (eds.), Translation Technology and its Teaching (with Much Mention of Localization) [en línea]. Tarragona: Intercultural Studies Group, [consulta: 20 de febrero de 2016]. ISBN 978-84-611-1131-2. Disponible en: http://www.intercultural.urv.cat/media/upload/domain_317/arxius/Technology/BiauPym_Technology.pdf.

BOWKER, L., 2002. *Computer-Aided Translation Technology: A Practical Introduction*. Ottawa: University of Ottawa Press. Didactic of Translation Series. ISBN 0-7766-0538-0.

BROOKS, D., 2000. What Price Globalization? Managing Costs at Microsoft. En: R.C. SPRUNG (ed.), Translating into Success: Cutting-edge Strategies for Going Multilingual in a Global Age. Ámsterdam; Filadelfia: John Benjamins Publishing Company, American Translators Association Scholarly Monograph Series, 11, pp. 43-57. ISBN 978-90-272-3186-4.

BYRNE, J., 2009. Localisation - When Language, Culture and Technology Join Forces. [en línea], Disponible en: http://www.languageatwork.eu.

BYRNE, J., 2012. *Scientific and Technical Translation Explained: A Nuts and Bolts Guide for Beginners*. Nueva York: Routledge.

CHANDLER, H.M., 2005. *The Game Localization Handbook*. Hingham: Charles River Media. ISBN 1-58450-343-2.

CHEN, Q. y SHARMA, V., 2002. Human Factors in Interface Design: An Analytical Survey and Perspective. En: E. SZEWCZAK y C. SNODGRASS (eds.), Human Factors in Information Systems. Hershey, Pensilvania: IRM Press, pp. 45-54.

CHRISTENSSON, P., 2006. TechTerms.com [en línea]. [consulta: 28 de junio de 2015]. Disponible en: http://techterms.com/.

CLEMENTS, P.C., BACHMANN, F., BASS, L., GARLAN, D., IVERS, J., LITTLE, R., MERSON, P., NORD, R. y STAFFORD, J., 2011. *Documenting Software Architectures: Views and Beyond*. 2.ª ed. Upper Saddle River: Addison-Wesley. SEI series in software engineering. ISBN 978-0-321-55268-6.

CLEMMENSEN, T., 2010. Regional Styles of Human-Computer Interaction. Proceedings of the 3rd international conference on Intercultural collaboration [en línea]. Nueva York: ACM New York, pp. 219-222. ISBN 978-1-4503-0108-4. DOI 10.1145/1841853.1841891. Disponible en: http://doi.acm.org/10.1145/1841853.1841891.

COMPUTER HISTORY MUSEUM, 2011. Punched Cards. Revolution: The First 2000 Years of Computing [en línea]. [consulta: 12 de octubre de 2014]. Disponible en: http://www.computerhistory.org/revolution/punched-cards/2.

CORPAS PASTOR, G. y VARELA SALINAS, M.J., 2003. *Entornos informáticos de la traducción profesional: las memorias de traducción*. Granada: Átrio. ISBN 84-96101-15-0.

CORTADA, J.W., 1990. *A Bibliographic Guide to the History of Computing, Computers, and the Information Processing Industry*. Nueva York: Greenwood Press. Bibliographies and indexes in science and technology, no. 6. ISBN 978-0-313-26810-6.

CORTADA, J.W., 1996. *Second Bibliographic Guide to the History of Computing, Computers, and the Information Processing Industry*. Westport, Connecticut: Greenwood Press. Bibliographies and indexes in science and technology, no. 9. ISBN 978-0-313-29542-3.

CRONIN, M., 2003. *Translation and Globalization*. Oxon: Routledge. ISBN 0-415-27065-2.

CRONIN, M., 2013. *Translation in the Digital Age*. Oxon; Nueva York: Routledge. New perspectives in translation studies. ISBN 978-0-415-60859-6.

DAVIES, I., GREEN, P., ROSEMANN, M., INDULSKA, M. y GALLO, S., 2006. How Do Practitioners Use Conceptual Modeling in Practice? Data & Knowledge Engineering, vol. 58, n.º 3, pp. 358-380. ISSN 0169023X. DOI 10.1016/j.datak.2005.07.007.

DE PEDRO RICOY, R., 2007. Internationalization vs. Localization: The Translation of Videogame Advertising. Meta: Journal des traducteurs, vol. 52, n.º 2, pp. 260. ISSN 0026-0452, 1492-1421. DOI 10.7202/016069ar.

DEL GALDO, E.M. y NIELSEN, J., 1996. *International User Interfaces*. Nueva York: John Wiley & Sons. ISBN 0-471-14965-9.

DENNING, P.J. y DARGAN, 1996. Action-Centered Design. En: T. WINOGRAD (ed.), Bringing Design to Software. Nueva York; Reading, Massachusetts: ACM New York; Addison-Wesley, pp. 105-127. ISBN 978-0-201-85491-6.

DENNIS, A., WIXOM, B.H. y ROTH, R.M., 2006. *Systems Analysis and Design*. 3.ª ed. S.l.: John Wiley & Sons.

DENNIS, A., WIXOM, B.H. y ROTH, R.M., 2009. *Systems Analysis and Design*. 4.ª ed. Hoboken, Nueva Jersey: John Wiley & Sons. ISBN 978-0-470-22854-8.

DENNIS, A., WIXOM, B.H. y TEGARDEN, D., 2005. *Systems Analysis and Design with UML Version 2.0: An Object-Oriented Approach*. 2.ª ed. S.l.: John Wiley & Sons.

DINGSØYR, T., NERUR, S., BALIJEPALLY, V. y MOE, N.B., 2012. A Decade of Agile Methodologies: Towards Explaining Agile Software Development. The Journal of Systems and Software, vol. 85, n.º 6, pp. 1213-1221. ISSN 0164-1212. DOI 10.1016/j.jss.2012.02.033.

DOHERTY, S., 2016. The Impact of Translation Technologies on the Process and Product of Translation. International Journal of Communication, vol. 10, pp. 947-969. ISSN 1932-8036.

DOI, M. y LEI, H., 2015. Word Processing for the Japanese Language. Engineering and Technology History Wiki [en línea]. [consulta: 17 de octubre de 2015]. Disponible en: http://ethw.org/Word_Processing_for_the_Japanese_Language.

DUMAS, M., VAN DER AALST, W. y TER HOFSTEDE, A.H.M. (eds.), 2005. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Hoboken, Nueva Jersey: John Wiley & Sons.

DUMAS, M., VAN DER AALST, W. y TER HOFSTEDE, A.H.M., [sin fecha]. Introduction. En: M. DUMAS, W. VAN DER AALST y A.H.M. TER HOFSTEDE (eds.), Process-Aware Information Systems. S.l.: s.n.,

DUNNE, K.J. (ed.), 2006. *Perspectives on Localization*. Ámsterdam; Filadelfia: John Benjamins Publishing Company. American Translators Association Scholarly Monograph Series.

DUNNE, K.J., 2011. From Vicious to Virtuous Cycle: Customer-focused Translation Quality Management Using ISO 9001 Principles and Agile Methodologies. En: K.J. DUNNE y E.S. DUNNE (eds.), Translation and Localization Project Management: The Art of the Possible [en línea]. Ámsterdam; Filadelfia: John Benjamins Publishing Company, American Translators Association Scholarly Monograph Series, pp. 153-187. [consulta: 26 de julio de 2016]. ISBN 978-90-272-3192-5. Disponible en: https://benjamins.com/catalog/ata.xvi.09dun.

DUNNE, K.J. y DUNNE, E.S. (eds.), 2011. *Translation and Localization Project Management: The Art of the Possible*. Ámsterdam; Filadelfia: John Benjamins Publishing Company. American Translators Association Scholarly Monograph Series. ISBN 978-90-272-3192-5.

DVK, 2009. Was ALGOL Ever Used for «Mainstream» Programming? StackOverflow [en línea]. [consulta: 26 de enero de 2016]. Disponible en: http://stackoverflow.com/questions/1463321/was-algol-ever-used-for-mainstream-programming.

ENGELS, G., FÖRSTER, A., HECKEL, R. y THÖNE, S., [sin fecha]. Process Modeling Using UML. En: M. DUMAS, W. VAN DER AALST y A.H.M. TER HOFSTEDE (eds.), Process-Aware Information Systems. S.l.: s.n.,

ENRÍQUEZ RAÍDO, V., 2016. Translators as Adaptive Experts in a Flat World: From Globalization 1.0 to Globalization 4.0? International Journal of Communication, vol. 10, pp. 970-988. ISSN 1932-8036.

FEENBERG, A., 1999. *Questioning Technology*. Londres: Routledge.

FITZGERALD, B., 1997. The Use of Systems Development Methodologies in Practice: A Field Study. Information Systems Journal, vol. 7, n.º 3, pp. 201-212. ISSN 1350-1917, 1365-2575. DOI 10.1046/j.1365-2575.1997.d01-18.x.

FLORES, F., GRAVES, M., HARTFIELD, B. y WINOGRAD, T., 1988. Computer Systems and the Design of Organizational Interaction. ACM Trans. Inf. Syst., vol. 6, n.º 2, pp. 153-172. ISSN 1046-8188. DOI 10.1145/45941.45943.

FLOYD, C., ZÜLLIGHOVEN, H., BUDDE, R. y KEIL-SLAWIK, R. (eds.), 1991. *Software Development and Reality Construction* [en línea]. Secaucus: Springer-Verlag New York, Inc. ISBN 0-387-54349-X. Disponible en: http://swt-www.informatik.uni-hamburg.de/uploads/media/sdrcbook.pdf.

FOLARON, D., 2006. A Discipline Coming of Age in the Digital Age. En: K.J. DUNNE (ed.), Perspectives on Localization. Ámsterdam; Filadelfia: John Benjamins Publishing Company, American Translators Association Scholarly Monograph Series,

FOLEY-FISHER, Z. y HANSON, C., 2014. *Localizing with XCode 6: Best practices and new workflows* [en línea]. 2014. S.l.: Apple. [consulta: 8 de enero de 2015]. Disponible en:

http://devstreaming.apple.com/videos/wwdc/2014/412xx80au1lrfcn/412/412_localizing_with_xcode_6.pdf?dl=1.

FORTRANFAN, 2014. Support for Internationalization in 2015 Beta. Intel Developer Zone [en línea]. [consulta: 11 de enero de 2016]. Disponible en: https://software.intel.com/en-us/forums/intel-visual-fortran-compiler-for-windows/topic/509837.

FOWLER, M. y PARSONS, R., 2011. *Domain-Specific Languages*. Upper Saddle River: Addison-Wesley. ISBN 0-321-71294-3.

FREIGANG, K.-H., 2005. Sistemas de memorias de traducción. En: D. REINEKE (ed.), Traducción y localización: mercado, gestión y tecnologías. Las Palmas de Gran Canaria: Anroart Ediciones, pp. 95-122.

FREIVALDS, J., 2005. The Changing Face of Translation Project Management. MultiLingual, vol. 16, n.º 1, pp. 17.

FRIEDMAN, T.L., 2005. It's a Flat World, After All. The New York Times Magazine [en línea]. 3 marzo 2005. [consulta: 23 de febrero de 2016]. Disponible en: http://nyti.ms/1eAIBJL.

GADDIS, T. y IRVINE, K., 2011. *Starting out with Visual Basic 2010*. 5.ª ed. Boston: Addison-Wesley.

GALA, 2016. Industry Standards. [en línea]. [consulta: 11 de marzo de 2016]. Disponible en: http://www.gala-global.org/lisa-oscar-standards.

GALITZ, W.O., 2002. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. 2.ª ed. Nueva York: John Wiley & Sons. ISBN 978-0-471-08464-8.

GARCÍA, I., 2006. Translators on Translation Memories: A Blessing or a Curse? En: A. PYM, A. PEREKRESTENKO y B. STARINK (eds.), Translation Technology and its Teaching (with Much Mention of Localization) [en línea]. Tarragona: Intercultural

Studies Group, [consulta: 20 de febrero de 2016]. ISBN 978-84-611-1131-2. Disponible en: http://www.intercultural.urv.cat/media/upload/domain_317/arxius/Technology/Garcia_Translators.pdf.

GARCÍA PEÑALVO, J., CONDE GONZÁLEZ, M.Á. y BRAVO MARTÍN, S., 2008. *Introducción a la ingeniería del software* [en línea]. 24 septiembre 2008. S.l.: Universidad de Salamanca. [consulta: 12 de junio de 2014]. Disponible en: http://ocw.usal.es/ensenanzas-tecnicas/ingenieria-del-software/contenidos/Tema1-IntroduccionalaIS-1pp.pdf.

GARNER, R., 2015. Early Popular Computers, 1950 - 1970. Engineering and Technology History Wiki [en línea]. [consulta: 7 de septiembre de 2015]. Disponible en: http://ethw.org/index.php?title=Early_Popular_Computers,_1950_-_1970&oldid=112928.

GASPARINI, I., PIMENTA, M.S. y PALAZZO M. DE OLIVEIRA, J., 2011. Vive la différence!: A Survey of Cultural-Aware Issues in HCI. En: C.S. DE SOUZA, A. SA NCHEZ y A.S. GOMES (eds.), Proceedings of the 10th Brazilian Symposium on Human Factors in Computing Systems and the 5th Latin American Conference on Human-Computer Interaction (IHC+CLIHC 2011) Porto de Galinhas, PE, October 25th to 28th, 2011. Porto Alegre, Brazil: Brazilian Computer Society, pp. 13-22. ISBN 978-85-7669-257-7.

GHAI, R., 2014. Automation Journey in the World of L10n!! En: e-mail autora: reetika.ghai@gmail.com, Adobe Globalization: Globalization, Internationalization, Localization, Translation [en línea]. [consulta: 18 de junio de 2015]. Disponible en: http://blogs.adobe.com/globalization/2014/02/06/automation-journey-in-the-world-of-l10n/.

GIBSON, M.L. y HUGHES, C.T., 1994. *Systems Analysis and Design: A Comprehensive Methodology with CASE*. Danvers, Massachusetts: Boyd & Fraser Publishing Company. ISBN 0-87709-247-8.

Globalization and Localization Association (GALA). [en línea], [sin fecha]. [consulta: 5 de abril de 2013]. Disponible en: http://www.gala-global.org/.

Glosario de siglas en los RFC. Grupo de traducción al castellano de RFC [en línea], 2001. [consulta: 31 de enero de 2016]. Disponible en: http://www.rfc-es.org/guia/glosario-siglas.txt.

GODUNKO, V., BASOV, A. y REZNIK, M., 2013. League. Matreshka Ada Framework [en línea]. [consulta: 22 de enero de 2016]. Disponible en: http://forge.ada-ru.org/matreshka/wiki/League.

GOETHALS, G., HODGSON, R., PRONI, G., ROBINSON, D. y STECCONI, U., 2003. Semiotranslation: Peircean Approaches to Translation. En: S. PETRILLI (ed.), Translation Translation. Ámsterdam: Rodopi, pp. 253-270.

GORTON, I., 2011. *Essential Software Architecture* [en línea]. Berlín; Heidelberg; Nueva York: Springer. [consulta: 9 de marzo de 2015]. ISBN 978-3-642-19176-3. Disponible en: http://site.ebrary.com/id/10466518.

GREENWOOD, T.G., 1993. International Cultural Differences in Software. En: J.C. BLAKE (ed.), Digital Technical Journal [en línea], vol. 5, n.º 3. [consulta: 23 de enero de 2015]. Disponible en: http://www.hpl.hp.com/hpjournal/dtj/vol5num3/vol5num3art1.pdf.

GROSS, S., 2006. *Internationalization and Localization of Software* [en línea]. Tesis de máster. Ypsilanti, Michigan: Eastern Michigan University. [consulta: 15 de marzo de 2009]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.175.2883&rep=rep1&type=pdf.

HAENTJENS, R., 1993. The Ordering of Universal Character Strings. En: J.C. BLAKE (ed.), Digital Technical Journal [en línea], vol. 5, n.º 3. [consulta: 23 de enero de 2015]. Disponible en: http://www.hpl.hp.com/hpjournal/dtj/vol5num3/vol5num3art4.pdf.

HALL, K., 2015. Migraine in the Mainframe: Unisys Reports More Losses. The Register [en línea]. [consulta: 23 de octubre de 2015]. Disponible en: http://www.theregister.co.uk/2015/10/22/unisys_reports_more_disappointing_results/.

HARRIS, J. y MCCORMACK, R., 2000. Translation is not Enough. Considerations for Global Internet Development. [en línea]. S.l.: Sapient Corp. [consulta: 31 de diciembre de 2015]. Disponible en: https://web.archive.org/web/20050515063219/http://www.sapient.com/pdfs/strategic_viewpoints/sapient_globalization.pdf.

HERRMANN, A. y SACHSE, F., [sin fecha]. Internacionalización de aplicaciones de software. . S.l.: s.n., pp. 45-70.

HIRSCHHEIM, R. y KLEIN, H.K., 2011. Tracing the History of the Information Systems Field. En: R.D. GALLIERS y W.L. CURRIE (eds.), The Oxford Handbook of Management Information Systems: Critical Perspectives and New Directions [en línea]. S.l.: Oxford University Press, pp. 16-61. [consulta: 10 de junio de 2014]. Disponible en: http://books.google.com.pr/books?id=aekv5WECPu8C&lpg=PA16&ots=LgyNxuAt-V&dq=info%3AKyNjAefe0LUJ%3Ascholar.google.com&lr&pg=PA16#v=twopage&q&f=true.

HIRSCHHEIM, R. y NEWMAN, M., 1991. Symbolism and Information Systems Development: Myth, Metaphor and Magic. Information Systems Research, vol. 2, n.º 1, pp. 29-62. ISSN 1047-7047.

HO, G., 2004. *Globalisation and Translation: Towards a Paradigm Shift in Translation Studies* [en línea]. Tesis doctoral. S.l.: University of Auckland. [consulta: 30 de diciembre de 2015]. Disponible en: http://search.proquest.com/docview/305111592.

HOFSTEDE, G., 1991. *Cultures and Organizations: Software of the Mind*. Berkshire: McGraw-Hill Book Company (UK) Limited. ISBN 0-07-707474-2.

HOGAN, J.M., HO-STUART, C. y PHAM, B., 2004. Key Challenges in Software Internationalisation. En: J. HOGAN (ed.), Australasian Workshop on Software Internationalisation (AWSI2004) [en línea]. Dunedin, Nueva Zelanda: ACS, pp. 187-194. [consulta: 30 de marzo de 2015]. Disponible en: http://crpit.com/abstracts/CRPITV32Hogan.html.

HURTADO ALBIR, A., 2011. *Traducción y traductología: Introducción a la traductología*. 5.ª ed. Madrid: Ediciones Cátedra. ISBN 978-84-376-2758-8.

HUSTED, B.W., 2003. Globalization and Cultural Change in International Business Research. En: *Globalization and the Role of the Global Corporation*, Journal of International Management, vol. 9, n.º 4, pp. 427-433. ISSN 1075-4253. DOI 10.1016/j.intman.2003.08.006.

Information Technology - Framework for Internationalization, 1998. Technical Report. S.l.: ISO/IEC. ISO/IEC TR 11017.

INTEL CORPORATION, 2003. *Intel: Thirty-five Years of Innovations (1968-2003)* [en línea]. S.l.: s.n. [consulta: 2 de noviembre de 2015]. Disponible en: http://www.intel.com/Assets/PDF/General/35yrs.pdf.

INTERNATIONAL BUSINESS MACHINES, 2003. IBM Archives: IBM Mainframes. [en línea]. [consulta: 27 de septiembre de 2015]. Disponible en: https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_intro.html.

INTERNATIONAL BUSINESS MACHINES, 2008. *Mainframe Concepts* [en línea]. S.l.: s.n. [consulta: 26 de febrero de 2016]. Disponible en: https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/zosbasics/com.ibm.zos.zmainframe/zmainframe_book.pdf.

INTERNATIONAL BUSINESS MACHINES, 2012a. Globalize Your Business. IBM Globalization [en línea]. [consulta: 2 de febrero de 2016]. Disponible en: http://www-01.ibm.com/software/globalization/index.html.

INTERNATIONAL BUSINESS MACHINES, 2012b. IBM z Systems - Operating Systems. [en línea]. [consulta: 26 de octubre de 2015]. Disponible en: http://www-03.ibm.com/systems/z/os/#zvm.

INTERNATIONAL BUSINESS MACHINES, 2012c. Unicode. IBM Globalization [en línea]. [consulta: 29 de febrero de 2016]. Disponible en: http://www-01.ibm.com/software/globalization/topics/unicode/index.html.

INTERNATIONAL BUSINESS MACHINES, 2013. Globalized Applications and Unicode. IBM Globalization [en línea]. [consulta: 29 de febrero de 2016]. Disponible en: http://www-01.ibm.com/software/globalization/topics/migratingdata/index.html.

INTERNATIONAL BUSINESS MACHINES, 2015. IBM Launches z13: Most Powerful & Secure System Ever Built. [en línea]. [consulta: 23 de octubre de 2015]. Disponible en: http://www-03.ibm.com/press/us/en/pressrelease/45808.wss.

ISHIDA, R., 2005. Hints for Designing International Web Pages. W3C [en línea]. [consulta: 19 de junio de 2015]. Disponible en: http://www.w3.org/2007/Talks/1015-ishida-iuc31/Overview.html.

ISHIDA, R., 2011a. Re-using Strings in Scripted Content. W3C Internationalization [en línea]. [consulta: 19 de junio de 2015]. Disponible en: http://www.w3.org/International/articles/text-reuse/.

ISHIDA, R., 2011b. Working with Composite Messages. W3C Internationalization [en línea]. [consulta: 19 de junio de 2015]. Disponible en: http://www.w3.org/International/articles/composite-messages/.

JIMÉNEZ-CRESPO, M.Á. y TERCEDOR, M., 2011. Applying Corpus Data to Define Needs in Web Localization Training. Meta: Journal des traducteurs, vol. 56, n.º 4, pp. 998. ISSN 0026-0452, 1492-1421. DOI 10.7202/1011264ar.

JOHN, B.E. y BASS, L., 2001. Usability and Software Architecture. Behaviour & Information Technology, vol. 20, n.º 5, pp. 329-338. ISSN 0144-929X. DOI 10.1080/01449290110081686.

JONES, P.H., 2010. The Language/Action Model of Conversation: Can Conversation Perform Acts of Design? interactions, vol. 17, n.º 1, pp. 70-75. ISSN 10725520. DOI 10.1145/1649475.1649493.

KAZMAN, R. y BASS, L., 1996. Software Architectures for Human-Computer Interaction: Analysis and Construction. [en línea]. S.l.: [consulta: 22 de diciembre de 2015]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.9516&rep=rep1&type=pdf.

KENDALL, K.E. y KENDALL, J.E., 2005. *Análisis y diseño de sistemas*. 6.ª ed. México: Pearson Educación.

KENDALL, K.E. y KENDALL, J.E., 2011. *Systems Analysis and Design*. 8.ª ed. Upper Saddle River: Pearson Prentice Hall. ISBN 978-0-13-608916-2.

KERSTEN, G.E., KERSTEN, M.A. y RAKOWSKI, W.M., 2001. Application Software and Culture: Beyond the Surface of Software Interface. InterNeg Reports INR1/01 [en línea], [consulta: 10 de abril de 2013]. Disponible en: http://www.researchgate.net/publication/228514009_Application_software_and_culture_Beyond_the_surface_of_software_interface/file/32bfe5102f6704210f.pdf.

KERSTEN, G.E., MATWIN, S., NORONHA, S.J. y KERSTEN, M.A., 2000. The Software for Cultures and the Cultures in Software. Proceedings of the 8th European Conference on Information Systems. S.l.: s.n., pp. 509-514.

KOSKINEN, K., 2010. Agency and Causality: Towards Explaining by Mechanisms in Translation Studies. En: T. KINNUNEN y K. KOSKINEN (eds.), Translator's Agency [en línea]. Tampere: Tampere University Press, Tampere Studies in Language,

Translation and Culture, B 4, pp. 165-187. [consulta: 24 de octubre de 2014]. Disponible en: https://tampub.uta.fi/bitstream/handle/10024/65639/978-951-44-8082-9.pdf?sequence=1.

KROENKE, D.M., 2013. *Using MIS 2013*. 5.ª ed. Harlow: Pearson Prentice Hall. ISBN 0-273-76648-1.

LACHAT-LEAL, C., 2015. Localización de aplicaciones: Estrategias de cooperación con los desarrolladores. *AIETI 7: Nuevos horizontes en los Estudios de Traducción e Interpretación*. Málaga: s.n.,

LANE, T., 1990. A Design Space and Design Rules for User Interface Software Architecture. [en línea]. Pittsburg, Pensilvania: Software Engineering Institute, Carnegie Mellon University. [consulta: 24 de junio de 2015]. CMU/SEI-90-TR-022. Disponible en: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11243.

LEE, A.S., 2004. Thinking About Social Theory and Philosophy for Information Systems. En: J. MINGERS y L.P. WILLCOCKS (eds.), Social Theory and Philosophy for Information Systems. Chíchester: John Wiley & Sons, John Wiley Series in Information Systems, pp. 1-26. ISBN 978-0-470-01121-8.

LEIDNER, D.E., 2010. Globalization, Culture, and Information: Towards Global Knowledge Transparency. The Journal of Strategic Information Systems, vol. 19, n.º 2, pp. 69-77. ISSN 0963-8687. DOI 10.1016/j.jsis.2010.02.006.

LIN, R., LIN, P.-H., SHIAO, W.-S. y LIN, S.-H., 2009. Cultural Aspect of Interaction Design Beyond Human-Computer Interaction. En: N. AYKIN (ed.), Internationalization, Design and Global Development. Berlín: Springer, pp. 49-58.

LINCOLN, Y.S., LYNHAM, S.A. y GUBA, E.G., 2011. Paradigmatic Controversies, Contradictions, and Emerging Confluences, Revisited. En: N.K. DENZIN y Y.S. LINCOLN (eds.), The Sage Handbook of Qualitative Research. 4.ª ed. Thousand Oaks, California: Sage Publications, pp. 97-128.

LITTAU, K., 2016. Translation's Histories and Digital Futures. International Journal of Communication, vol. 10, pp. 907-928. ISSN 1932-8036.

LIU, Z. y ZHANG, G., 2010. Exploring Culture Factors in HCI Design: A Perspective from SEUC. 1st International Workshop on Comparative Informatics (IWCI 2010): Understanding Cultural Influences in Usability, Gaming, Collaboration, Government and Design [en línea]. Copenhague: s.n., [consulta: 1 de mayo de 2013]. Disponible en: http://www.itu.dk/people/rkva/IWCI-2010/docs/100727-position%20paper-final.doc.

LÓPEZ SÁNCHEZ, R., 2012. *Guía básica de software para traductores*. S.l.: s.n.

MALVEAU, R.C. y MOWBRAY, T.J., 2001. *Software Architect Bootcamp*. Upper Saddle River: Prentice Hall. Software architecture series. ISBN 0-13-027407-0.

MAROTO ORTIZ-SOTOMAYOR, J., 2004. *Localisation, Anti-Essentialism, Ethics and the Online Advertiser: The Impact of Contemporary Translation Studies Theories in Cross-Cultural Online Advertising*. julio 2004. S.l.: s.n.

MAROTO ORTIZ-SOTOMAYOR, J. y DE BORTOLI, M., [sin fecha]. Creativity Across Borders, a New Approach. Global Propaganda [en línea]. [consulta: 23 de julio de 2016]. Disponible en: http://www.globalpropaganda.com/articles/CreativityAcrossBorders.pdf.

MATA PASTOR, M., 2005. Localización y traducción de contenido web. En: D. REINEKE (ed.), Traducción y localización: mercado, gestión y tecnologías. Las Palmas de Gran Canaria: Anroart Ediciones, pp. 187-252.

MAZUR, I., 2007. The Metalanguage of Localization: Theory and Practice. En: Y. GAMBIER y L. VAN DOORSLAER (eds.), International Journal of Translation Studies, vol. 19, n.º 2, pp. 337-357. ISSN 0924-1884.

MEI, Q. y BOYLE, T., 2010. Dimensions of Culturally Sensitive Factors in the Design and Development of Learning Objects. Journal of Interactive Media in Education, vol. 2010, n.º 1, pp. 6. ISSN 1365-893X. DOI 10.5334/2010-6.

MICROSOFT DEVELOPER NETWORK, 2016. Globalizing and Localizing .NET Framework Applications. [en línea]. [consulta: 20 de marzo de 2016]. Disponible en: https://msdn.microsoft.com/en-us/library/h6270d0z(v=vs.110).aspx.

MINGERS, J. y WILLCOCKS, L. (eds.), 2004. *Social Theory and Philosophy for Information Systems*. Chíchester: John Wiley & Sons. John Wiley series in information systems. ISBN 0-470-85117-1.

MISRA, T., 2014. Can Google Build A Typeface To Support Every Written Language? Code Switch: Frontiers of Race, Culture and Ethniticiy [en línea]. [consulta: 6 de agosto de 2014]. Disponible en: http://www.npr.org/blogs/codeswitch/2014/08/03/337168933/-no-tofu-doesn-t-equate-to-no-problem-for-google-universal-typeface.

MOL, M., 2015. Unicode Strings. Rosetta Code [en línea]. [consulta: 26 de enero de 2016]. Disponible en: http://rosettacode.org/wiki/Unicode_strings.

MOLICH, R. y NIELSEN, J., 1990. Improving a Human-Computer Dialogue. Communications of the ACM, vol. 33, n.º 3, pp. 338-348. ISSN 00010782. DOI 10.1145/77481.77486.

MONTALT I RESURRECCIÓ, V., 2010. La traducción de géneros electrónicos: el caso de la localización. En: E. ORTEGA ARJONILLA, A.B. MARTÍNEZ LÓPEZ y E. ECHEVERRÍA PEREDA (eds.), Panorama actual de investigación en traducción e interpretación. Granada: Átrio, pp. 313-328. ISBN 978-84-96101-78-4.

MORADO VÁZQUEZ, L. y TORRES DEL REY, J., 2015. Teaching XLIFF to Translators and Localisers. Localisation Focus. The International Journal of Localisation, vol. 14, n.º 1, pp. 4-13. ISSN 1649-2358.

MORRIS, S.B., 2006. Aspect Oriented Programming and Internationalization. Java.net [en línea]. [consulta: 5 de abril de 2013]. Disponible en: http://today.java.net/pub/a/today/2006/07/25/aop-and-i18n.html.

MOYA, V., 2010. *La selva de la traducción: teorías traductológicas contemporáneas*. 3.ª ed. Madrid: Ediciones Cátedra. ISBN 978-84-376-2118-0.

MUMFORD, E., 2006. The Story of Socio-Technical Design: Reflections on its Successes, Failures and Potential. Information Systems Journal, vol. 16, n.º 4, pp. 317-342. ISSN 13501917.

MUSTONEN-OLLILA, E. y LYYTINEN, K., 2003. Why Organizations Adopt Information System Process Innovations: A Longitudinal Study Using Diffusion of Innovation Theory. Information Systems Journal, vol. 13, n.º 3, pp. 275-297. ISSN 1350-1917, 1365-2575. DOI 10.1046/j.1365-2575.2003.00141.x.

MUZII, L., 2006. Building a Localization Kit. [en línea]. [consulta: 30 de junio de 2016]. Disponible en: http://www.translationdirectory.com/article1151.htm.

NASS, C.I., MATTHEW, L., HENRIKSEN, L. y STEUER, J., 1995. Anthropocentrism and Computers. Behaviour & Information Technology, vol. 14, n.º 4, pp. 229-238.

NASS, C.I. y STEUER, J., 1993. Voices, Boxes, and Sources of Messages. Human Communication Research, vol. 19, n.º 4, pp. 504-527. ISSN 1468-2958. DOI 10.1111/j.1468-2958.1993.tb00311.x.

.NET FOUNDATION, 2015. .NET Core: A General Purpose Managed Framework. .NET Core [en línea]. [consulta: 18 de enero de 2016]. Disponible en: https://dotnet.github.io/.

NICCOLAI, J., 2015. IBM's New z13 Mainframe Eats Mobile App Data for Lunch. Computerworld [en línea]. [consulta: 23 de octubre de 2015]. Disponible en: http://www.computerworld.com/article/2868457/ibms-new-z13-mainframe-eats-mobile-app-data-for-lunch.html.

NIELSEN, J. y MOLICH, R., 1990. Heuristic Evaluation of User Interfaces. [en línea]. Nueva York: ACM New York, pp. 249-256. [consulta: 22 de marzo de 2015]. ISBN 0-201-50932-6. DOI 10.1145/97243.97281. Disponible en: http://portal.acm.org/citation.cfm?doid=97243.97281.

NORMAN, D., 2002. *The Design of Everyday Things*. Nueva York: Basic Books.

OATES, B.J., 2006. *Researching Information Systems and Computing*. Londres: Sage Publications.

OBJECT MANAGEMENT GROUP, INC., 2015. *OMG Unified Modeling Language Version 2.5* [en línea]. S.l.: s.n. [consulta: 30 de noviembre de 2015]. Disponible en: http://www.omg.org/spec/UML/2.5/PDF.

O'CONNER, J., 2013. Internationalization as an Architecture. Adobe Globalization: Globalization, Internationalization, Localization, Translation [en línea]. [consulta: 18 de junio de 2015]. Disponible en: http://blogs.adobe.com/globalization/2013/03/07/internationalization-as-an-architecture/.

O'HAGAN, M., 1996. *The Coming Industry of Teletranslation: Overcoming Communication Barriers through Telecommunication*. Clevedon: Multilingual Matters. Topics in translation, 4. ISBN 1-85359-326-5.

O'HAGAN, M., 2016. Massively Open Translation: Unpacking the Relationship Between Technology and Translation in the 21st Century. International Journal of Communication, vol. 10, pp. 929-946. ISSN 1932-8036.

OLIVER, A. y MORÉ, J., 2008. *Traducción y tecnologías*. Barcelona: Editorial UOC.

OLLE, T.W., 2006. IFIP TC8 Information Systems: Conception, Birth and Early Years. En: D. AVISON, S. ELLIOT, J. KROGSTIE y J. PRIES-HEJE (eds.), The Past and Future of Information Systems: 1976-2006 and Beyond. Boston: Springer, IFIP International Federation for Information Processing, pp. 1-10.

PACEY, A., 2001. *Meaning in Technology*. Cambridge: MIT Press. ISBN 0-262-66120-9.

PATCH, N., 2015. Unicode Beyond Just Characters: Localization with the CLDR. [en línea]. [consulta: 2 de febrero de 2016]. Disponible en: http://patch.codes/talks/localization-with-the-unicode-cldr/.

PÉREZ-QUIÑONES, M.A., PADILLA-FALTO, O.I. y MCDEVITT, K., 2005. Automatic Language Translation for User Interfaces. Proceedings of the 2005 Conference on Diversity in Computing [en línea]. Nueva York: ACM New York, pp. 60-63. [consulta: 30 de marzo de 2015]. ISBN 1-59593-257-7. DOI 10.1145/1095242.1095268. Disponible en: http://doi.acm.org/10.1145/1095242.1095268.

PERRY, D.E. y WOLF, A.L., 1992. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, vol. 17, n.º 4, pp. 40-52. ISSN 01635948. DOI 10.1145/141874.141884.

PESQUET, C.H., 1993. Foreword. En: J.C. BLAKE (ed.), Digital Technical Journal [en línea], vol. 5, n.º 3. [consulta: 23 de enero de 2015]. Disponible en: http://www.hpl.hp.com/hpjournal/dtj/vol5num3/foreword.htm.

POULSEN, L., 2001. Computer History. Lars Poulsen's Professional WWW Pages [en línea]. [consulta: 24 de septiembre de 2014]. Disponible en: http://www.beagle-ears.com/lars/engineer/comphist/.

PROUDFOOT, D., 2014. Alan Turing: Codebreaker and Computer Pioneer. History Today [en línea], vol. 54, n.º 7. [consulta: 26 de septiembre de 2015]. Disponible en: http://www.historytoday.com/diane-proudfoot/alan-turing-codebreaker-and-computer-pioneer.

PYM, A., 2006. What Localization Models Can Learn from Translation Theory. [en línea], [consulta: 6 de agosto de 2015]. Disponible en: http://usuaris.tinet.cat/apym/on-line/translation/localization_translation_theory.pdf.

PYM, A., 2010. *Exploring Translation Theories*. Oxon; New York: Routledge. ISBN 978-0-415-55362-9.

PYM, A., 2011. Website Localization. [en línea], vol. 10. Disponible en: http://usuaris. tinet. cat/apym/online/translation/2009_website_localization_feb. pdf.

PYM, A., 2013. Translation Skill-Sets in a Machine-Translation Age. Meta: Journal des traducteurs, vol. 58, n.º 3, pp. 487. ISSN 0026-0452, 1492-1421. DOI 10.7202/1025047ar.

PYM, A., PEREKRESTENKO, A. y STARINK, B. (eds.), 2006. *Translation Technology and its Teaching (with Much Mention of Localization)* [en línea]. Tarragona: Intercultural Studies Group. [consulta: 20 de febrero de 2016]. ISBN 978-84-611-1131-2. Disponible en: http://www.intercultural.urv.cat/media/upload/domain_317/arxius/Technology/translationtechnology.pdf.

Quality Attributes. SoftwareArchitectures.com [en línea], 2013. [consulta: 21 de junio de 2015]. Disponible en: http://www.softwarearchitectures.com/qa.html.

RADI, S., AHMAD, A. y BOROKHOV, P., 2015. *New UIKit Support for International User Interfaces* [en línea]. 2015. S.l.: Apple. [consulta: 1 de agosto de 2015]. Disponible en: http://devstreaming.apple.com/videos/wwdc/2015/222ngkqh58b52/222/222_new_uikit_support_for_international_user_interfaces.pdf?dl=1.

RAU, P.-L.P., GAO, Q. y LIANG, S.-F.M., 2008. Good Computing Systems for Everyone–How on earth? Cultural Aspects. Behaviour & Information Technology, vol. 27, n.º 4, pp. 287-292. ISSN 0144-929X. DOI 10.1080/01449290701761250.

RAYMOND, E.S., 2003. Origins and History of Unix, 1969-1995. The Art of Unix Programming [en línea]. S.l.: s.n., [consulta: 15 de febrero de 2016]. Disponible en: http://www.catb.org/esr/writings/taoup/html/ch02s01.html.

REAL ACADEMIA ESPAÑOLA, 2005. *Diccionario panhispánico de dudas*. Madrid: Santillana. ISBN 958-704-368-5.

REAL ACADEMIA ESPAÑOLA, 2010. *Ortografía de la lengua española*. Madrid: Espasa.

REINECKE, K. y BERNSTEIN, A., 2007. Culturally Adaptive Software: Moving Beyond Internationalization. En: N. AYKIN (ed.), Usability and Internationalization. Global and Local User Interfaces [en línea]. S.l.: Springer Berlin Heidelberg, Lecture Notes in Computer Science, pp. 201-210. ISBN 978-3-540-73288-4. Disponible en: http://dx.doi.org/10.1007/978-3-540-73289-1_25.

REINEKE, D. (ed.), 2005. *Traducción y localización: mercado, gestión y tecnologías*. Las Palmas de Gran Canaria: Anroart Ediciones.

RESSIN, M. y ABDELNOUR-NOCERA, J., 2011. Comparing Development Roles in Software Localisation. 2nd Comparative Informatics Workshop (IWCI 2011). Copenhague: s.n.,

RISKU, H., PEIN-WEBER, C. y MILOSEVIC, J., 2016. «The Task of the Translator»: Comparing the Views of the Client and the Translator. International Journal of Communication, vol. 10, pp. 989-1008. ISSN 1932-8036.

RODRÍGUEZ V. DE ALDANA, E. y TORRES DEL REY, J., 2013. Localisation Standards for Joomla! Translator-Oriented Localisation of CMS-Based Websites. Localisation Focus. The International Journal of Localisation, vol. 12, n.º 1, pp. 4-14. ISSN 1649-2358.

RODRÍGUEZ VÁZQUEZ, S., 2013a. Making Localised Web Content Accessible: A Collaborative Task Between the Developer and the Localiser. En: P. SÁNCHEZ-GIJÓN, O. TORRES-HOSTENCH y B. MESA-LAO (eds.), Conducting Research in Translation Technologies. Berna: Peter Lang, New trends in translation studies, pp. 93-115. ISBN 978-3-0343-0994-3.

RODRÍGUEZ VÁZQUEZ, S., 2013b. Towards Defining the Role of Localisation Professionals in the Achievement of Multilingual Web Accessibility. Tradumàtica: tecnologies de la traducció, n.º 11, pp. 383-388. ISSN 1578-7559.

RODRÍGUEZ VÁZQUEZ, S. y TORRES DEL REY, J., 2014. Fostering Accessibility Through Web Localization. MultiLingual, vol. 25, n.º 7, pp. 32-37. ISSN 15230309.

ROMERO FRESCO, P., 2013. Accessible Filmmaking: Joining the Dots Between Audiovisual Translation, Accessibility and Filmmaking. *JoSTrans*, n.º 20, pp. 201-223. ISSN 1740-357X.

ROWLEY, J., 1990. *The Basics of Systems Analysis and Design for Information Managers*. Londres: Clive Bingley Limited.

RUSSO, P. y BOOR, S., 1993. How Fluent is your Interface?: Designing for International Users. [en línea]. Nueva York: ACM New York, pp. 342-347. [consulta: 19 de enero de 2015]. ISBN 0-89791-575-5. DOI 10.1145/169059.169274. Disponible en: http://portal.acm.org/citation.cfm?doid=169059.169274.

SAFAR, L. y MACHALA, J., 2010. Best Practices in Localization Testing. MultiLingual, vol. 21, n.º 1, pp. 28-32. ISSN 1523-0309.

SAHA, G.K., 2008. On Localization of Enterprise Information Systems. En: L.D. XU, A.M. TJOA y S.S. CHAUDHRY (eds.), Research and Practical Issues of Enterprise Information Systems II [en línea]. S.l.: Springer, IFIP — The International Federation for Information Processing, pp. 545-551. ISBN 978-1-4757-0563-8. Disponible en: http://dx.doi.org/10.1007/978-0-387-75902-9_60.

SALDANHA, G. y O'BRIEN, S., 2013. *Research Methodologies in Translation Studies*. Mánchester: St. Jerome Publishing. ISBN 978-1-909485-00-6.

SÁNCHEZ-GIJÓN, P., TORRES-HOSTENCH, O. y MESA-LAO, B. (eds.), 2013. *Conducting Research in Translation Technologies*. Berna: Peter Lang. New trends in translation studies. ISBN 978-3-0343-0994-3.

SATZINGER, J.W., JACKSON, R.B. y BURD, S.D., 2009. *Systems Analysis and Design in a Changing World*. 5.ª ed. Boston: Course Technology, Cengage Learning. ISBN 978-1-111-53415-8.

SAVOUREL, Y., 2001a. Localization in XML. Markup Languages: Theory & Practice, vol. 3, n.º 4, pp. 387-393. ISSN 10996621.

SAVOUREL, Y., 2001b. *XML Internationalization and Localization*. Indianápolis: Sams. ISBN 978-0-672-32096-5.

SCHEER, A.-W., THOMAS, O. y ADAM, O., [sin fecha]. Process Modeling Using Event-Driven Process Chains. En: M. DUMAS, W. VAN DER AALST y A.H.M. TER HOFSTEDE (eds.), Process-Aware Information Systems. S.l.: s.n.,

SCHMITZ, K.-D., 2005. Gestión terminológica en la localización de software. En: D. REINEKE (ed.), Traducción y localización: mercado, gestión y tecnologías. Las Palmas de Gran Canaria: Anroart Ediciones, pp. 123-141.

SCHWANNINGER, C., WUCHNER, E. y KIRCHER, M., 2004. Encapsulating Crosscutting Concerns in System Software. *AOSD'04 International Conference on Aspect-Oriented Software Development* [en línea]. Láncaster: s.n., [consulta: 21 de enero de 2015]. Disponible en: http://www.kircher-schwanninger.de/michael/publications/CC_Patterns.pdf.

SHAPIRO, R.J., 2014. The U.S. Software Industry: An Engine for Economic Growth and Employment. S.l.: Software and Information Industry Association.

SHELLY, G.B. y ROSENBLATT, H.J., 2012. *Systems Analysis and Design*. Boston: Course Technology, Cengage Learning. ISBN 978-0-538-48161-8.

SOFTWARE ENGINEERING INSTITUTE AT CARNEGIE MELLON UNIVERSITY, 2015. Software Architecture Glossary. [en línea]. [consulta: 5 de junio de 2015]. Disponible en: http://www.sei.cmu.edu/architecture/start/glossary/index.cfm.

SOWA, J.F. y ZACHMAN, J.A., 1992. Extending and Formalizing the Framework for Information Systems Architecture. IBM Systems Journal, vol. 31, n.º 3, pp. 590-616. ISSN 0018-8670. DOI 10.1147/sj.313.0590.

SPRUNG, R.C. (ed.), 2000. *Translating into Success: Cutting-Edge Strategies for Going Multilingual in a Global Age*. Ámsterdam; Filadelfia: John Benjamins Publishing Company. American Translators Association Scholarly Monograph Series, 11. ISBN 978-90-272-3186-4.

STEELE, G.L. y GABRIEL, R.P., 1996. The Evolution of Lisp. En: T.J. BERGIN Jr. y R.G. GIBSON Jr. (eds.), History of Programming Languages II [en línea]. Nueva York: ACM New York, pp. 233-330. ISBN 0-201-89502-1. Disponible en: http://doi.acm.org/10.1145/234286.1057818.

STENNING, K. y OBERLANDER, J., 1995. A Cognitive Theory of Graphical and Linguistic Reasoning: Logic and Implementation. Cognitive Science, vol. 19, n.º 1, pp. 97-140.

STROUSTRUP, B., 1991. What is «Object-Oriented Programming»? (1991 revised version). *1st European Software Festival* [en línea]. Múnich: s.n., [consulta: 4 de marzo de 2016]. Disponible en: http://www.stroustrup.com/whatis.pdf.

SUMNER, J., 2010. History of Computing in the UK: A Resource Guide. Special Interest Group: Computers, Information and Society [en línea]. Disponible en: http://www.sigcis.org/britain.

SUNDAR, S.S. y NASS, C.I., 2000. Source Orientation in Human-Computer Interaction. En: Sage Publications, Inc., Communication Research, vol. 27, n.º 6, pp. 683-703.

TAKEUCHI, H. y NONAKA, I., 1986. The New New Product Development Game. Harvard Business Review, pp. 137-146. ISSN 0017-8012.

TEXIN, T. y CRAFT, X., 2011. Script Direction and Languages. W3C Internationalization [en línea]. [consulta: 8 de agosto de 2015]. Disponible en: http://www.w3.org/International/questions/qa-scripts.

The Missing Apple iOS Localization Term Glossary. [en línea], [sin fecha]. [consulta: 11 de marzo de 2016]. Disponible en: https://www.ibabbleon.com/apple-ios-localization-term-glossary.html.

THE NATIONAL MUSEUM OF COMPUTING, 2014. *Colossus at 70: Colossus and the Breaking of the Lorenz/Tunny* [en línea]. Bletchley Park: The National Museum of Computing. [consulta: 27 de septiembre de 2015]. Disponible en: https://youtu.be/QcaHpvznC7g.

TORRES DEL REY, J., 2005. *La interfaz de la traducción: formación de traductores y nuevas tecnologías*. Granada: Editorial Comares. Interlingua, 47. ISBN 84-8444-937-8.

TORRES DEL REY, J., 2014. Book review: Translation and Web Localization. Parallèles, n.º 26, pp. 133-135.

TORRES DEL REY, J., [sin fecha]. The Proper Place of Localisation in Translation Curricula: An Inclusive Communicative, Objectual and Social Approach. . S.l.: s.n.,

TORRES DEL REY, J. y MORADO VÁZQUEZ, L., 2015. XLIFF and the Translator: Why Does it Matter? Tradumàtica: tecnologies de la traducció, n.º 13, pp. 561-570. ISSN 1578-7559.

TORRES DEL REY, J. y RODRÍGUEZ V. DE ALDANA, E., 2014. La localización de webs dinámicas: objetos, métodos, presente y futuro. *JoSTrans* [en línea], n.º 21. [consulta: 19 de julio de 2016]. ISSN 1740-357X. Disponible en: http://www.jostrans.org/issue21/art_torres_rodrigue.php.

TORRES DEL REY, J. y RODRÍGUEZ VÁZQUEZ, S., 2012. A Communicative Approach to Evaluate Web Accessibility Localisation Using a Controlled Language Checker: the Case of Text Alternatives for Images. The International Journal of Localisation, vol. 11, n.º 1, pp. 27-39. ISSN 1649-2358.

TORRESI, I., 2010. *Translating Promotional and Advertising Texts*. Mánchester; Kinderhook: St. Jerome Publishing. Translation Practices Explained. ISBN 978-1-905763-20-7.

TRACTINSKY, N., 2000. A Theoretical Framework and Empirical Examination of the Effects of Foreign and Translated Interface Language. Behaviour & Information

Technology, vol. 19, n.º 1, pp. 2-13. ISSN 0144-929X. DOI 10.1080/01449290050086345a.

UNICODE CONSORTIUM, 2015. Unicode CLDR Project. CLDR - Unicode Common Locale Data Repository [en línea]. [consulta: 30 de enero de 2016]. Disponible en: http://cldr.unicode.org/.

UNISYS, 2015. Unisys Debuts Most Powerful ClearPath Dorado Systems Ever, Completing Family's Transition to Intel-Based Fabric Architecture. Unisys [en línea]. [consulta: 23 de octubre de 2015]. Disponible en: http://www.unisys.com/offerings/high-end-servers/clearpath-systems/clearpath-dorado-systems/News%20Release/Unisys-Debuts-Most-Powerful-ClearPath-Dorado-Systems-Ever.

UREN, E., 1997. Annotated Bibliography on Internationalization and Localization. SIGDOC Asterisk J. Comput. Doc., vol. 21, n.º 1, pp. 26-33. ISSN 0731-1001. DOI 10.1145/250982.250991.

VAN DER VEER, M., 2015. A Brief History of Programming Languages. Learning Algol 68 Genie [en línea]. S.l.: s.n., pp. 3-5. [consulta: 17 de enero de 2016]. Disponible en: http://jmvdveer.home.xs4all.nl/learning-algol-68-genie.pdf.

VENKATESAN, V.P. y KUPPUSWAMI, S., 2008. ARMMS - Architecture Reference Model for Multilingual Software. Journal of Convergence Information Technology, vol. 3, n.º 2, pp. 74-81.

VENTERS, W., CONFORD, T. y LANCASTER, M., [sin fecha]. Information Systems Development Methodologies: Literature Review. Pegasus: Particle-physics Engagement with the Grid: A Socio-technical Usability Study [en línea]. [consulta: 11 de marzo de 2016]. Disponible en: http://www.lse.ac.uk/management/research/pegasus/Outputs/ISDMLitReview.htm.

Visual Paradigm Community Edition [en línea], 2016. S.l.: s.n. [consulta: 11 de marzo de 2016]. Disponible en: http://www.visual-paradigm.com/editions/community.jsp.

VISURI, S. y SWEEZEY, M.R., 2014. Localization Testing in an Agile Environment. MultiLingual, vol. 25, pp. 45.

VU, J.H., 2010. *Software Internationalization: A Framework Validated Against Industry Requirements for Computer Science and Software Engineering Programs* [en línea]. Tesis de máster. San Luis Obispo, California: California Polytechnic State University. [consulta: 9 de marzo de 2015]. Disponible en: http://digitalcommons.calpoly.edu/theses/248.

What´s New in Internationalization [en línea], 2015. San Francisco: [consulta: 14 de agosto de 2016]. WWDC 2015 Session 227. Disponible en: https://developer.apple.com/videos/play/wwdc2015/227//.

WICKENS, C.D., GORDON, S.E. y LIU, Y., 1998. *An Introduction to Human Factors Engineering*. Nueva York: Addison-Wesley Educational Publishers Inc.

WILLIAMS, J. y CHESTERMAN, A., 2002. *The Map: A Beginner's Guide to Doing Research in Translation Studies*. Mánchester: St. Jerome Publishing.

WINOGRAD, T. y FLORES, F., 1987. *Understanding Computers and Cognition: A New Foundation for Design*. Boston: Addison-Wesley Longman Publishing Co., Inc. ISBN 0-201-11297-3.

WINTERS, G.B., 1993. International Distributed Systems: Architectural and Practical Issues. En: J.C. BLAKE (ed.), Digital Technical Journal [en línea], vol. 5, n.º 3. [consulta: 23 de enero de 2015]. Disponible en: http://www.hpl.hp.com/hpjournal/dtj/vol5num3/vol5num3art5.pdf.

WISE, T.A., 1966. I.B.M.'s $ 5,000,000,000 Gamble. Fortune, vol. 74, n.º 4, pp. 118-123. ISSN 0015-8259.

YEO, A., 1996. Cultural User Interfaces: A Silver Lining in Cultural Diversity. SIGCHI Bull., vol. 28, n.º 3, pp. 4-7. ISSN 0736-6906. DOI 10.1145/231132.231133.

YEO, A., 2001. Global-Software Development Lifecycle: An Exploratory Study. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems [en línea]. Nueva York: ACM New York, pp. 104-111. ISBN 1-58113-327-8. DOI 10.1145/365024.365060. Disponible en: http://doi.acm.org/10.1145/365024.365060.

ZACHMAN, J.A., 1987. A Framework for Information Systems Architecture. IBM Systems Journal, vol. 26, n.º 3, pp. 276-292. ISSN 0018-8670. DOI 10.1147/sj.263.0276.

ZIKOPOULOS, P., DEROOS, D. y BIENKO, C., 2015. *Big Data Beyond the Hype: A Guide to Conversations for Today's Data Center*. Nueva York: McGraw Hill Education. ISBN 978-0-07-184465-9.