CHALMERS | UNIVERSITY OF GOTHENBURG

Guidelines for Multilingual Software Development

A compilation and systematic presentation of guidelines for multilingual software development applicable to software in all stages of the lifecycle with the goal of encouraging and facilitating internationalized and localized software products.

Muhammad Murtaza
Ahmed Shwan

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2009

Guidelines for Multilingual Software Development

Muhammad Murtaza
Ahmed Shwan

Examiner: Agneta Nilsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: the cover page picture is a “word cloud” that shows the most frequently used words in this report.

Department of Computer Science and Engineering
Göteborg, Sweden March 2012

VERSION CONTROL			
VERSION	**CHANGES**	**DATE**	**AUTHOR**
1.0	Template and draft outline added	20110922	MM
2.0	Literature review 4.1 and 4.2 added	20110930	MM & AS
2.1	Literature review 4.3 appended	20111031	MM & AS
2.2	Literature review completed 4.3	20111114	MM & AS
2.3	Literature review reviewed and appended 4.3	20111130	MM
2.4	Preface, Chapters 1 and 3 added	20111213	MM & AS
2.5	Chapter 2 added	20111217	MM & AS
2.6	Section 1.2.1 updated	20111226	AS
2.7	Sections 3.4 and 4.1 updated	20111231	MM & AS
2.8	Chapter 4 updated; up to 4.4.3 finalized	20120102	MM & AS
2.9	Outline for Chapter 5 finalized	20120106	MM
3.0	Sections 4.4.4, 4.4.5 and 4.4.6 finalized	20120108	MM & AS
3.1	Sections 4.5, 4.6 and 4.4 finalized	20120110	MM & AS
3.2	Up to 4.6 completed	20120113	MM & AS
3.3	Chapter 4 completed	20120114	MM
3.4	Section 5.1 completed	20120129	MM & AS
3.5	Chapter 3 updated	20121229	MM
3.6	Appendix added	20121229	AS
3.7	Section 5.2 completed	20120203	MM & AS
3.8	Section 5.3 GNTs done, details to be added.	20120205	MM & AS
3.9	Chapter 6 completed	20120207	MM & AS
4.0	Chapter 5 completed	20120208	MM & AS
4.1	Chapter 7 added	20120209	MM & AS
4.2	Abstract and Conclusion added	20120210	MM & AS
4.3	Overall review done; ready for submission	20120211	AS
4.4	Updated based on Sven's final review	20120224	MM
4.5	Minor updates based on examiner's feedback	20120416	MM & AS

Page left blank intentionally

MASTER'S THESIS IN SOFTWARE ENGINEERING

Guidelines for Multilingual Software Development

Muhammad Murtaza and Ahmed Shwan

Department of Computer Science and Engineering
Chalmers | University of Gothenburg
Gothenburg, Sweden

February 2012

Guidelines for Multilingual Software Development

Muhammad Murtaza and Ahmed Shwan

Master's Thesis 2012

Department of Computer Science and Engineering

Chalmers | University of Gothenburg

SE-412 96 Gothenburg

Sweden

Telephone: +46 (0) 31-772 1000

Printer/Department of Computer Science and Engineering

Gothenburg, Sweden 2012

Guidelines for Multilingual Software Development

Master's Thesis in Software Engineering

Muhammad Murtaza and Ahmed Shwan

Department of Computer Science and Engineering

Chalmers | University of Gothenburg

Abstract

For software products to be effectively usable by an international audience, they must be localized, or translated, to suite the target user group's culture and language. Multilingual software development is a vast topic and it crosscuts all the phases of software development life cycle and it requires consideration of additional factors and activities such as new roles and linguistic translation of content to one or more languages. One of the main reasons why most developers ignore multilingual software development is a lack of awareness about the related activities, tools, technologies, practices and service providers, as well as the associated uncertainty of possible implications.

This thesis project aims to raise awareness about and encourage multilingual software development. This is done by reviewing relevant literature and understanding industry practices, and then presenting a list of guidelines for multilingual software development in an organized manner and providing a means of accessing only the guidelines suitable to and useful for a given project. To be specific, guidelines for multilingual software development are provided for software projects that are under feasibility study, being implemented or in maintenance phase. The relevant guidelines can be retrieved using the Guidelines Navigation Tools (GNT) devised for different software lifecycle phases.

Keywords: multilingual, software, internationalization, localization, development, guidelines, translation, SDLC, guidelines navigation tool, GNT.

Preface

Digital data and information, as opposed to data and information available on paper, can easily be copied, transmitted, shared, reviewed and even converted. We believe conversion of data and information, also known as translation, is being neglected by many and it is about time to bring the attention of software publishers to this issue, and encourage and assist them to make their products international.

It is recognized that unlike copying, transmitting, sharing and reviewing the process of electronic data and information conversion to suit non-native target audiences is complex and multi-disciplinary. It requires numerous technical solutions, administrative approaches and human translators. Also, technological advancements in the field of machine translation or computer-aided translation are far from replacing human translators and interpreters.

Nevertheless, a key step towards making electronic data and information accessible to all humans around the world is to build software with non-native language speakers in mind. This starts by understanding that unlike written text or spoken words, there is a need for more than mere translation. The technical term for software translation is localization, and it is far more than mere translation of the textual content of the software; it includes technical activities, project management, administration and more.

It is not being claimed that all software should be implemented with all human languages in mind, but they should definitely be designed and implemented utilizing the already available technologies and project management practices so that future non-native needs can be cost-effectively satisfied. Just like good design and coding practices are embedded in development technologies and enforced through implementation methodologies, multilingual software requirements should be given a similar attention, if not for immediate requirements, then for the inevitable non-native language needs and potential benefits of multilingual software. After all, project owners often eventually realize the tremendous benefits of potentially accessing a larger audience that is beyond the native speakers.

Designing and developing international software is an important issue, but what about the already developed and deployed software applications? What about the data managed by these applications? If we consider the number of web applications on the internet alone, isn't it important to internationalize them too?

With the presumption that a main problem with the lack of multilingual software today, and as a result globally inaccessible software, lies in the obliviousness of the project owners and developers, and through diligent literature and case studies' reviews, it was concluded that a positive contribution to the main goal (of making software, and thus data, accessible to all) would be the compilation and organization of guidelines for multilingual software development. Guidelines that can be easily navigable and suitable depending on what the software lifecycle stage would be cost-effective and provide encouragement for project owners and developers, and possibly increase the number of multilingual software applications. This includes guidelines for the vast number of software projects already deployed. The industry interviews confirmed this presumption.

Acknowledgements – Muhammad

My work in this project is driven by the dream of making the vast amount of data and information available in digital form, especially on the internet, accessible to people around the world. Electronic data and information can today be cost-effectively delivered to all people including those in remote villages and this can be a catalyst for their change and uplifting. Therefore, regardless of the native language, which could even be a sign language spoken by the deaf and mute communities, using information and communication technologies available today, I believe it is possible to deliver data and information available in any language in digital form to any other community around the world in a language they understand and in a cost-effective way. I have worked on a number of projects that prove this. The approaches differ and often are a combination of different works done by researchers.

Therefore, I thank all the authors and researchers, either in academia and industry, who have contributed, intentionally or as a side effect, in one way or the other, to making data and information accessible to all people around the world. We are what we know and it is the right of all humans to have an opportunity to know.

Also, I take this opportunity to sincerely thank all the interviewees and experts who took the time from their busy schedules to contribute in this work by participating in face to face and online interviews and surveys. Without their contributions, we would not have been able to verify our understanding about their projects and the guidelines for multilingual software developed that was extracted and organized.

I would also like to thank my family for their patience and support and my project supervisor Prof. Sven-Arne Andréasson and project partner Ahmed Shwan for their encouragement and trust.

Acknowledgements – Ahmed

Multilingual software applications present economical benefits and open opportunities in global markets. The importance of the project as I perceived it kept my motivations high throughout. Contributing in such a way that may one day help make all content available in multiple languages is exciting.

This thesis is an intensive, exciting and memorable work experience. Thanks to my thesis partner Muhammad Murtaza, it was an enjoyable endeavor too. It was lively to work with him and a special thanks to my supervisor Sven-Arne Andréasson for his wise advices. Thank you for all the energy, ideas, and inspiration. I wish all master students have the same leadership and interaction that I received.

Words are not enough to thank my family for moral and psychological support, which is necessary to keep going and continue working in spite of all the uncertainties. It kept me going through the vagaries of life, so thank you for believing in my abilities and my intellectual skills.

I am also indebted to the contribution of all those who helped us during this project, including, researchers, developers and project owners.

Table of Contents

List of Tables

List of Models

Terms and Definitions

In this section, key terms relevant to the topic of multilingual software are listed. The terms and definitions are divided into two groups. The first contains multilingual software related terms and the other contains general terms related to software, information technology and the research area in general:

Term	Description
Computer Aided Translation (CAT)	It is a form of language translation in which a human translator uses computer software to support and facilitate the translation process.
Cosmetic testing	User interface testing to ensure everything fits in and looks as expected.
Cross-platform	Cross-platform is an attribute in computer software that refers to whether the system can inter-operate on multiple computer platforms.
Document Object Model (DOM)	It is a convention for representing and interacting with objects in HTML, XHTML and XML documents.
Double-byte enablement	Internationalizing a product so it supports the input, processing and display of double-byte characters used in Asian languages.
eXtensible Markup Language (XML)	It is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined and maintained by the W3C.
Full match	Source text segment that matches 100% with a previously stored sentenced in the Translation Memory (see below) tool.
Fuzzy matching	Method used in Translation Memory tools to identify text segments that previously translated segments by 100%, so similar translations is leveraged.
G11N	Globalization addresses the business issues associated with launching a product globally. In the globalization of high-tech products this involves integrating localization throughout a company, after proper internationalization and product design, as marketing, sales, and support in the world market.
Gisting	Using machine translation to convey the approximate meaning.
Globalization	Used in academia and industry to refer to multilingual software and in the context of software, it broadly means the process of developing and marketing software products to a global market. Globalization of software includes both Internationalization and Localization of the software, in addition to other project management and business related activities such as sales and marketing.
GNT	Same as Guidelines Navigation Tool.
Guidelines Navigation Tool	A means to access and retrieve relevant guidelines for multilingual software development.
I18N	Same as Internationalization.
Internationalization	The International Standards Organization (ISO) and the International Electro-technical Committee (IEC) defined Internationalization as: A process of producing an application platform or application which is easily capable of being localized for any cultural environment. In this thesis, the terms internationalization, international or internationalized software has been leniently used to refer to multilingual software or its development.
International Standardization Organization (ISO)	A worldwide federation of national bodies governing standards in approximately 130 countries, one from each country.
Internationalization testing	Testing whether a software product is internationalized properly; testing the localizability of a product.
L10N	Same as Localization.
Layered graphic	An image file in which translatable text is stored on a separate layer than rest of

	the image to enable translation.
Locale	In information technology, locale refers to the supported languages and country and culture specific properties that the user wants to see in their user interface. Basically, it refers to where the user comes from and it is more than merely the spoken and written language, for it includes cultural norms and practices. For example, using SEK instead of $ as currency in Sweden.
Localization	THE ISO and IEC defined Localization as: "A process of adapting an internationalized application platform or application to a specific cultural environment".
Localization glossary	A glossary used in language translation testing, includes verification of context and language suitability of the localized product user interface.
Localization Industry Standards Association (LISA)	An organization which was founded in 1990 and is made up of mostly software publishers and localization service providers. LISA organizes forums, publishes newsletters, conducts surveys, and has initiated several special-interest groups focusing on specific issues in localization.
Localization testing	Combination of linguistic and cosmetic testing to ensure the quality of the user interface in a localized application.
Machine Translation (MT)	A methodology and technology used automate language translations from one human language to another, using terminology glossaries advanced grammatical, syntactic and semantic analysis techniques.
Multi Language Vendors (MLV)	Localization vendor that offers translation services in multiple target languages.
Outsourcing	In the context of localization, contracting certain activities to third parties. Most localization vendors outsource translation work to freelance translators, and publishers often outsource the full localization process to localization vendors, including translation, engineering and testing.
PP	Same as Project Properties.
Pseudo translation	Replace each translatable text string with longer string to spot problems with localized versions.
Project Properties	Characteristics of a software project or product that helps suggest better guidelines when using the GNT.
Quality Assurance (QA)	The steps and processes in place to ensure a quality final product.
SDL Passolo	Commercial tool that allows the user to use the visual localization environment to select the best user interface translation, to edit the translation in runtime, to customize the tool to meet the needs of the user in an integrated development environment, and integrate the environment with other development environment.
Segmentation	It is division of text into translatable units such as sentences or paragraphs, and most TM tools contain segmentation rules.
SimShip	Simultaneous shipment or release of different localized versions.
Single Language Vendor (SLV)	Localization vendor that offers translation services in one target language.
Software localization engineer	A person who is responsible for analysis and preparation of localization files and for localization and testing of GUI, online help, and web sites.
Static web site	A web site that consists mainly of HTML pages with text graphics that are manually updated. The concept of "static" web site contrasts with dynamic web site.
Terminology Management System (TMS)	It is a major organizational asset that contains a list of all terms and their meanings. Efficient management of terminology is a key contributor to quality and consistency of the final multilingual content your company produces.
Terminology-oriented database	It is a conceptual extension of an object-oriented database. It implements concepts defined in a terminology model.
Translation Memory eXchange (TMX)	It is an open XML standard for the exchange of translation memory data created by computer-aided translation and localization tools.

Translation Memory (TM)	It is a database that stores so-called "segments", which can be sentences or sentence-like units (headings, titles or elements in a list) that have previously been translated.
Unicode	A 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard.
UTF-8	An encoding form of Unicode that supports ASCII for backward compatibility and covers the characters for most languages in the world. UTF-8 is short for 8-bit Unicode Transfer Format.

Multilingual Software Terms

Term	Description
3G	Third Generation High Speed Mobile Networks
Android	Is a smart phone operating system created by the Open Handset Alliance, a collaboration of actors in the mobile phone market
API	Application Programming Interface, a well-defined interface that simplifies exchanges between systems
OS	Operating System
RIA	Rich Internet Application
SRS	Software Requirements Specification; document containing all system requirements
UI	User Interface
Wi-Fi	Wireless Network

General Terms

1 Introduction

Software solutions providers and internet companies alike must address the needs of international or non-native customers and users and provide services and contents in different world languages in order to grow and remain competitive (Kane, 2011). This includes both the data and the interface of the software. Large firms utilize or devise different mechanisms to develop and maintain their software in multiple languages, like how Facebook used crowd-sourcing to introduce tens of languages within a couple of years (Malik, 2009). Similarly, all major internet companies such as search engines and social networks provide services in numerous languages (Atkins-Kruger, 2010). However, this does not mean they do not struggle with introducing versions of their software in different languages. In our view, this is especially true when multilingualism of the software isn't part of the requirements and design documents.

While larger organizations are better equipped and have the financial strength to develop and manage multilingual software, even if not pre-planned, the same may not be true for medium and small businesses (Stivala, 2010). Therefore, a majority of such website and other software owners consider only one language. Many that are eventually pulled by market demand or competition to provide their service and content in a second or third language, are taken by surprise and realize that their design or implementation technology do not support extension of the presentation layer; i.e. the user interface, and the data layer; i.e. the database schema, in additional languages. This can lead to makeshift solutions of compromised quality and unexpected costs (Wooten, 2010). It should be noted that medium and small businesses represent a large portion of the economy in most developed and developing countries and their growth and success could positively affect the economy as a whole (Ashrafi & Murtaza, 2010).

Having said that, software applications of all sizes and types have been successfully deployed targeting audiences from different parts of the world, speaking both homogenous and heterogeneous languages. In other words, the technology needed for multilingual software development already exists and success stories can be found around the world of software products developed using different platforms by small and large companies. Before continuing though, first the differences between language types should be explained.

Heterogeneous languages are different in origin and various linguistic properties such as text orientation. Homogeneous languages, on the other hand, come from the same origin and have similar linguistic properties. Technically speaking, it can be observed that software that need to support homogenous languages require a simpler approach than those that must support heterogeneous languages. This may be in terms of the underlying computer character encoding (how the language character is electronically represented in the computer's memory (Wikipedia Character Encoding, 2011)), how the data is stored in the repository or what properties must the user interface (UI) components have (for example, does the user interface support bidirectional languages with right-to-left scripts?). An example of a software developed by a medium-sized company to support Arabic and English (two heterogeneous languages as Arabic is right to left with connected letters and require double-byte encoding while English is left to right and requires single-byte encoding) is iTrust Enterprise Resource Planning (ERP) system. While most software

applications in Eastern countries need to support heterogeneous languages, software applications targeting the Western audience need to support homogenous languages. One example would be Chalmers University of Technology website, which has Swedish and English language versions, two languages that are of Germanic origins with similar linguistic properties.

Now back to the issue of multilingual software. It can be observed that technologies and advancements in the field of software development allow us to successfully develop and even modify multilingual software (by adding new language versions). However, most developers are oblivious to these technologies and approaches, and project owners fail to correctly prioritize multilingual requirements. As a result, once the project owners realize the numerous benefits of international software that support users from different countries around the world, the developers struggle with introducing new language versions.

In this thesis project, numerous guidelines for multilingual software development are compiled and presented it in an easily navigable manner. Some 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, who might need to develop multilingual software or modify existing monolingual software to support new languages.

The guidelines are extracted from two main sources. The first and larger source is the available literature on multilingual software development and the second source is collected data from interviews (interactive at times) with actual software project members.

The following subsections introduce the topic of multilingual software, provide details of this research project and list key terms used in this document.

1.1 Multilingual software development

Multilingual software, also sometimes referred to as International Software, is basically software that is designed and implemented such that it can be used by a wide range of users from around the world in their own languages and cultural norms. Multilingual software exhibits multilingualism in its functionalities and aspects, including input, output (user interface, for instance), and storage (Venkatasamy, 2009). It is important to understand why software publishers should develop multilingual software and when did a large number of specialized service providers emerged.

Prior to 1980, software generally was intertwined with hardware in the form of specialized equipments that were managed using embedded systems. The commercialization of personal computers led to the appearance of more and more software applications aimed at increasing the efficiency of business users or entertaining personal users, for example. This happened in the United States and to some extern in the United Kingdom. In many countries, even today, a personal computer user was expected to be proficient in English language. However, as personal computers were pushed into new markets around the world, soon this language barrier was realized and software applications with interfaces in native languages emerged.

In 1980s, specialist businesses known as Multi Language Vendors (MLV) emerged that offered specialized services such as content translations, software engineering consultation and testing. These firms are known as localization service providers (localization is defined at the end of this chapter). Large software developers had realized that development, sale and maintenance of their solutions in different languages with high quality was a complex process, and this complexity multiplied as different target markets demanded the application in their native languages and software developers started to build multilingual software or localized software, which basically provided the same functionality in whatever language the user selected (Esselink, 2000).

The localization industry grew so much that Localization Industry Standards Association (LISA) was founded in 1990 and in the next few years the industry saw the rise and fall of different corporations, including Lionbridge, ALPNET, SDL and Berlitz. In the 1990s, firms categorized as Single Language Vendors (SLV) were active too and many had strategic partnerships with MLVs (SLV operates only with one country and provides localization services for the language and culture of that country). In fact, such was the growth that Ireland established itself as world leader by providing numerous incentives to software developers and localization firms. Today, numerous firms are completely of partially (with localization teams and divisions) based in Ireland and take care of localization activities for firms such as Oracle, Microsoft and Siebel. Education and training institutes have also stepped in and specialized courses in software localization are offered to software engineering, computer science and translation students.

The 1990s saw the emergence of the internet, which led to a new model of commercial software distribution and sale. With the possibility of selling a successful software application around the world further highlighted the importance of developing multilingual software. Now, without the need to find local partners to train and establish long term relationships with, to name a few commercial challenges of entering new markets, software developers can conveniently target customers around the world using a multilingual website, provided that the application and related documentations support the culture of the target market. After 2000, as the internet replaced all other means of commercial and personal communication and collaboration, additional technologies and approaches emerged, some of which may be incorporated in the process of multilingual software development and making web content available to all in different languages. Machine translation and speech conversion technologies are two examples.

In his book, Bert Esselink states that French, Italian, German and Spanish are the first locales that software developers tend to localize their products into. However, Middle East and North Africa region is considered to be one of the fastest growing regions (Ashrafi & Murtaza, 2010) and compared to Europe and Americas, the languages spoken there are fundamentally different and require special technical solution as outlined by Sameer Abufardeh in his doctoral research project (Abufardeh S. O., 2009).

Multilingual software development is a complex and expensive endeavor due to the high costs of content translation, increased man hours in design and coding, necessity of additional tools and skills, and effects on project management and architecture. The sooner the project owners realize the multilingual requirements,

the sooner the project manager and architect can provision for international software. Software engineers know today that the earlier the requirements are understood and defects are detected, the cheaper and easier the implementation and solution. In our view, in case of multilingual software, this is even more true and important.

Today, technologies and techniques needed for multilingual software development are available. For example, the introduction of Unicode, an encoding system that supports all scripts used in the world by using 2 bytes or 16 bits to represent each character digitally, made it possible for operating systems and platforms to easily handle all world languages. Similarly, different programming languages or development technologies provide specialized localization frameworks, libraries and properties for the user interfaces components. This might not be perfect, but surely development of international software has come a long way, and unfortunately, in spite of the well documented advantages of developing international software (Wooten, 2010), the vast majority of software is monolingual and even public websites are targeted at a small number of locales. Cities around the world are increasingly becoming multicultural and number of languages native to users online is increasing (Internet World Stats, 2010). Companies and economies can grow if new markets are effectively entered by building international software or modifying current software to support new languages. What is it that needs to be done to encourage and facilitate multilingual software development?

1.2 About the research

The goal of this research during the early stages was to find ways to make data and information available online and in electronic form available to all around the world. Since all electronic data are delivered through software, naturally the electronic data and information is more than mere linguistic translation but could include automated translation. Since this goal is quite ambitious and long-term in nature, the focus of the research is more specific hopefully the findings of this project will be a stepping stone towards this long-term goal. The focus of this project was refined multiple times as discussed in Chapter 4 titled Literature Review.

In this section an overview of this thesis project's topic and its importance are presented, as well as its key attributes such as the research questions, the methodology, and the main contributions.

1.2.1 Focus of the research

Numerous and diverse solutions for multilingual software development are available for different kinds of software developers. Yet the majority of commercial software developed around the world are designed and implemented without considering the likely future requirements to target the software application at new locales around the world due to the tremendous commercial benefits. Similarly, free software and public websites fail to consider the majority of users online who speak different world languages. It is understandable for certain software products and services to be monolingual or support a limited number of

languages, but sadly it is easy to find commercial web-based service providers ignoring potential customers that are non-native.

Although English is the contemporary language of commerce, being able to market in additional languages can provide tremendous competitive advantage (Wooten, 2010). The following table provides descriptive examples of software applications and websites from Sweden that are monolingual (only Swedish interfaces):

Organization	Limitation
Blocket.se	This classified advertisements website has only one Swedish version. It is not wrong to believe that Blocket should at least provide an English version of their website, especially since their content is user generated. Swedish attracts thousands of foreign students every year and such websites can prove to be invaluable.
IKEA Sweden	It is fair to expect IKEA addressing the needs of temporary foreign professionals or students who move to Sweden temporarily by having an English version for their website in Sweden at least, if not the online store. IKEA's international branches, such as IKEA Kuwait, support both English and Arabic.
Nordea Internet Banking	Although security is crucial, thousands of international students have to manage their bank accounts in Swedish language as the online banking application is monolingual. This is strange, as the website is accessible in numerous languages, including English.

Table 1-1: Monolingual Software in Sweden

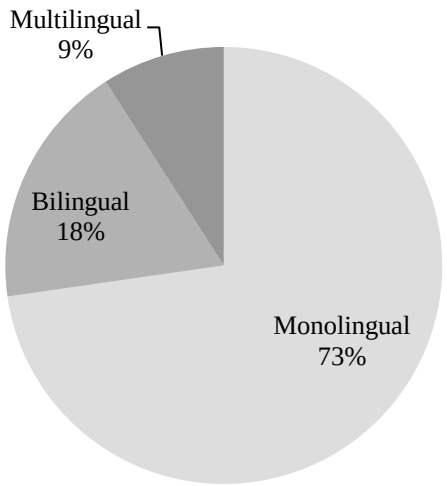
It is not implied that the entities in the table wish to introduce their content or service in additional languages or that they are unhappy with their current software. However, the examples highlight the main problem that must be addressed; i.e. the lack of awareness by many developers about non-native users, the ad-hoc approach to developing international software, limitations in introducing additional languages post-implementation due to poor design or implementation practices, and a lack of proper utilization of tools and service providers specialized in software localization. Also, these examples are of web-based applications but the same applies to mobile applications and platform-dependent software. Similarly, numerous examples can be found in Sweden and elsewhere around the world.

In this thesis project numerous guidelines for multilingual software development are presented in an organized manner. These guidelines are extracted from the literature and from analysis of sample projects, mostly from Sweden. In addition to raising awareness among developers and project owners about international software, these guidelines would facilitate the process by decreasing complexity and costs and increasing confidence and certainty.

With effects on virtually all organizations around the world, commercial and governmental, the possible economical losses and dissatisfaction to a large number of customers, the importance of this topic is clear.

Additionally, to further justify the focus of this study, let us consider the most visited websites in a country where English is not the native language (a website is a web-based software application after all and although the context might be different than other software, because the users are generally diverse, we feel it is a good example to point out how developers ignore internationalization even when the website is literally accessible by all). In Sweden, the top fifty websites, or websites most visited by internet users from inside Sweden, includes internet giants like Google, Facebook, YouTube, Windows Live, MSN and Yahoo. This is in line with the international trend and all are available in multiple languages. The list also includes Swedish newspapers and magazines and due to the nature of business, it is normal for such websites to be monolingual to reflect the language of the newspaper. Nevertheless, the list also includes numerous web-based services and businesses founded in Sweden, and sadly upon analyzing them, it is found that most of them are available only in Swedish. This includes banks, online trading and classifieds websites. This could mean a loss of international users and hence limited growth, or inside Sweden, it could mean poor services and lack of consideration for non-Swedish speakers, which could lead to loss of business (Alexa.com, 2011).

Excluding international websites, local newspapers and media organizations, the top fifty websites in Sweden include: blocket.se, swedbank.se, hitta.se, eniro.se, tradera.se, prisjakt.nu, nordea.se, handelsbanken.se, hemnet.se, ams.se and seb.se. Out of these, swedbank.se and seb.se are available in English and Swedish. Nordea.se is only in Swedish but an alternative international website (nordea.com) is available with support for numerous languages, however, all the local banking services are available native languages only. Website for the Swedish employment service, ams.se, is available in more than 35 world languages, however, different versions have different content and all other languages have less content than that available in the original Swedish version. The following graph summarizes the websites in terms of the supported languages:



Model 1-1: Swedish Websites by Supported Languages

As shown, 73% of the top fifty websites in Sweden, excluding media and international websites are monolingual. A similar pattern can be found in almost all countries where English isn't the native language. The purpose of this analysis is to highlight this pattern and the phenomena of majority of local software applications being monolingual, both developed by private businesses and government agencies.

1.2.2 Why is it urgent now?

It is not a matter of urgency as much as of paying attention to a crucial aspect of software that would improve both customer satisfaction and organizational competitiveness. We believe a majority of software developers overlook multilingualism during the early stages of software development and consider it a low priority. Also, like post-release changes in general, the costs of introducing the complete system in an additional language can be very high (SimulTrans, 2011). If small and medium software publishers around the world and in countries like Sweden start to internationalize their products and services, the potential for growth is great, and many of the larger software publishers have already proved this.

Furthermore, with emergence and spread of convergent handheld devices, developers today often must create different versions of their software for different platforms. For example, the group scheduling software Doodle is now available on iOS and Android, in addition to the cross-platform web-based version. In such cases, if multilingualism is ignored during the early stages, the consequences could be undesirable as introducing the software in a new language for a single platform is challenging enough, doing the same for multiple platforms can be agonizing.

1.2.3 Why can it be approached now?

Today, more multilingual software exist, compared to 1990s and before, and one reason for this is the growth of Information and Communications Technology (ICT) in other parts of the world and the emergence and growth of the internet as mentioned before. For example, this can be specifically observed in Arab Gulf countries, where the oil boom has led to the development of the ICT industry. This in turn has led to the introduction of numerous governmental and commercial electronic services, most of which are bilingual; in Arabic language for the natives and English for the large expat community working in the region.

Google Translate is widely used and the service now supports 58 world languages (Google, 2011). This service clearly does not replace professional translation services, but it surely is a big step towards making the web available to a much larger audience. While many use it to translate texts manually, the service is also used by many to translate websites. This is done either when the visitor of the website manually translates it by entering the web address (URL) or when software developers use the Google Translate API to provide a feature within the software to translate the software into additional languages. Today, Google Translate API is unavailable (as of end of 2011) for free and a paid version of the service has been offered as a replacement to developers (Feldman, 2011). To address the high demand for this service, Microsoft has launched Microsoft Translator Tools and Bing Translator, and Yahoo has been offering its Babel Fish

translation service. These services and other alternatives show the importance of the issue and the high demand for a multi language internet.

Having said this, none of these services empower the developer to have control over the translated content, perhaps intentionally. But imagine a governmental service or some large corporate website using one of the auto-translation services to introduce new languages. The quality of the translated text from such services is known to be low (SEO Translator, 2011) and even smaller and medium service providers seldom use it.

Considering such competition among Internet giants and demand in general for multilingual software, it seems like the right time to finally provide easily navigable guidelines that can be used to transform any software application into an international software application, regardless of the project lifecycle phase (whether the project is being study, is under development or already released).

1.2.4 Research questions and methodology

In this section, the key research questions that guide us throughout the project are listed and the methodology is briefly described. The scope for this research project is defined by the following research questions:

- What are the available tools, techniques, approaches and technologies for multilingual software development and for modification of existing software to support additional languages and cultures?
- What software development practices inhibit future modification of software applications to support additional languages (future localization of existing software)?
- Many software applications and websites today are monolingual, ignoring the needs of many users and possibly ignoring large sources of revenue. What are the main reasons for this and how can this change?
- What considerations can be made and practices introduced in the Software Development Life Cycle (SDLC) to facilitate multilingual software development?

The answers may clarify practices and attitudes in the industry and encourage and facilitate the development of international software, potentially resulting in growth for businesses and economies. In order to pursue these answers systematically, a research methodology was diligently followed. The methodology is discussed in detail in Chapter 3. Here a brief description of the activities and tasks is provided:

- Literature review of publications on multilingual software to, in addition to understand the topic better, extract effective guidelines for multilingual software development.
- Selection of suitable sample software applications for study and analysis.
- Interview selected applications' project team members to confirm analysis and extract additional guidelines for multilingual software development.

- Compilation of the extracted multilingual software development guidelines and its presentation in an organized manner. Also, the development of a navigation tool to ease the selection of the appropriate guidelines based on numerous project properties and factors, such as lifecycle phase.
- Verification of the results (the guidelines for multilingual software development) through a focus group survey.

1.2.5 Research contribution and limitations

The software industry as well as the emergence of electronic businesses providing products and services over the internet has changed the way we in general perceive and interact with software. The fastest growing markets are in Middle East and other parts of Asia, where English is not native. Statistics indicate that the number of non-native English speakers is increasing and with cities becoming ever more multi cultural, it is safe to expect a future with software, websites, electronic data and web-based services supporting multiple languages (Internet World Stats, 2010). We believe and hope that this research project contributes to the transformation of all electronic data and the software that manages the data and related services into a form that is accessible by all people regardless of the languages they understand and speak and the culture and locale they belong to.

To be more specific, this thesis contributes by:

1. Compiling a large number of international software development guidelines from literature and industry and presented by software project lifecycle phase, technology and other categories.
2. Developing a navigation tool to help project managers and other team members quickly and easily navigate through a large number of guidelines to view only the relevant ones.
3. Increasing the number of international software applications and websites by reducing uncertainty and costs and by increasing the confidence of project members.
4. Providing a high level introduction about all tools, techniques, technologies and practices related to multilingual software in one document.

We also hope that this project contributes towards our long term goals and hopes of making all information on the internet accessible to people around the world. In terms of limitations, like all work, we have identified certain shortcomings.

This is discussed in detail at the end of this document in Chapter 7, nevertheless, the main limitations are the lack of coverage of all kinds of software in detail (software are different and embedded systems have different characteristics than websites and standalone mission critical software, for example), a small number of sample size mostly from Sweden, and limited validation of results.

1.3 Document Structure

Chapter 1 introduced the subject of this research project by concisely discussing the field of multilingual software and by describing the research topic (Guidelines for Multilingual Software Development) and the

project's goals. The remainder of this document provides more detail about this qualitative research project and all the works done during the project.

Chapter 2 provides a background for the research topic and discusses the main motivation to work on this project. Chapter 3 outlines and explains the research methodology followed during this project.

A large part of the project required the compilation of sound guidelines for multilingual software development and so Chapter 4 includes the literature reviewed. In addition to presenting literature on relevant topics in an organized manner, from which most of the guidelines were extracted, how the review of literature affected the research topic's focus is also discussed.

Chapter 5 is the main chapter and provides all the results of this work. It includes transcripts of the interviews with software project team members, the guidelines for multilingual software development, as extracted from literature and from analysis of sample projects, and provides a means for navigating through the vast number of guidelines.

Chapter 6 provides the results of the verification tasks done to verify the key findings and results of the research project, as presented in Chapter 5. Chapter 7 provides a discussion on the overall paper and discusses the limitations of this research project and outlines possible future uses and projects as a consequence of the results and findings of this research paper.

Finally, Chapter 8 summarizes and concludes everything discussed and presented in this paper.

2 Background and Motivation

In this chapter, a detailed background about multilingual software development is presented. Additionally, the motivation is briefly discussed. This chapter provides more contextual data about the research topic and it is intended for readers interested in knowing more about the history and contemporary state of international software.

2.1 Background on Multilingual Software

The information presented here provides a background to the literature review in Chapter 4, which focuses on the available material that can be used as guidelines today for multilingual software. From different literature various aspects of multilingual software are reviewed, including: the industry, internationalization, localization, software engineering practices, software project management, software quality assurance, software translation and content translation technology.

2.1.1 The industry

In Chapter 1, in the section titled "multilingual software development", the domain was introduced and it was discussed how the software industry witnessed the development of a new and fast growing industry specializing in software localization. This new industry seems like a natural specialization of the software and information technology industries that met the growing demand for international software by merging software developers with translators and language specialists. As a continuation of section 1.1, briefly the current state of the industry is described.

The localization industry today in 2011-2012 is quite stable and mature. It has struck a balance between the needs of large software developers, localization tools developers and freelance and commercial translators. A number of online communities sprung up offering the services of translators and well-defined administrative and business processes have been developed to efficiently international software. The emergence of single and multiple language vendors (also known as SLV and MLV) have further strengthened the industry. Nevertheless, observing of small and medium sized software published around the world, it seems like the industry is satisfied by focusing on larger developers. Perhaps the reason might be the tight budgets of such small firms and their focus on meeting short term goals. Such software publishers, who after all develop and maintain the vast majority of software applications, mobile apps and even websites, hopefully would benefit from this research the most through an improved awareness about the benefits and importance of international software. Also, the guidelines compiled and presented in this paper should enable them to confidently take the decision of internationalizing their software.

In addition to the observation about the obliviousness of small and medium sized software developers towards multilingualism, another observation can be made in terms of the affects latest technologies and practices in the software industry at large might have had at the localization industry and its practices. To be specific, it is intriguing how the rise of mobile applications development, availability of machine translation

services such as Google Translate and Microsoft Translator. Also, the successful emergence of agile practices for software development that focus on iterative releases and closer collaboration among all stakeholders is an important development, thus requiring consideration from a multilingual software development perspective. Since no direct literature was found on these topics, we use the best of our knowledge to discuss these issues, extending to some extent the ideas of Abufardeh (Abufardeh S. O., 2009) and Esselink (Esselink, 2000).

2.1.1.1 Multilingual software development and mobile platforms

Mobile application development is influenced by the success and failure of mobile platforms. 2010 and 2011 saw the fall of leading platforms such as Nokia Symbian and Blackberry RIM. Nokia adapted Microsoft Windows Phone for all its smart phones and it is too soon to tell how this decision turns out. Apple iOS and Android are market leaders and majority of mobile applications developers focus on these two platforms. Additionally, the history is repeating itself with the emergence of web technologies for mobile application development that are fortified by bridging technologies (Hall, 2008) (Bai, 2011). In our view, the localization industry and its practices need not change in light of the emergence of mobile platforms. This is said because the development technologies used for mobile development simply extend the ones that were used for desktop and web development, perhaps with the exception of bridging technologies (these are mobile application development frameworks that allow web developers to write software for different platforms using cross platform web technologies such as HTML and with the power of native technologies through access to native device resources such as GPS and phone). For example, Android developers using Java programming language and iOS for iPhone uses Conditional C, two mature development technologies used for many years. The developers would have, like they would in case of any target platform, must check the languages supported by the mobile platforms on the operating system level. For example, Android does not fully support a number of bidirectional languages such as Arabic and Urdu.

2.1.1.2 Multilingual software development and web based translation services

As far as machine translation is concerned, the only effect the emergence of free services have had on the localization industry is the availability of built-in translators in some web-based applications. It is not uncommon to find websites developed around the world utilizing tools such as Google Translate Element to offer the users an option to translate the content in their own languages. However, this is only the case of free and non-critical websites such as blogs and nonprofit organizations as the quality of the translation is low in most cases (Beninatto, 2011). Additionally, although there is no data to support this, based on logic and observations, we believe that the number of freelance translators (at least in some parts of the world) has increased and translators today can work faster because of machine translation services available for free. The number of freelance translators has somewhat increased as many multilingual persons around the world utilize free services from Google, Microsoft and Yahoo to translate content and then they improve the quality based on their native skills to finesse the final output before submission.

2.1.1.3 Multilingual software development and agile software methodologies

Of all the changes in software industry, in our view agile development practices have affected multilingual software projects the most. In particular, practices such as having all team members in one room and releasing software frequently demand consideration for translators and other localization experts and practices. Guidelines for this from literature and industry are provided in Chapter 5. Nevertheless, in brief, a closer collaboration would be required with localization firms or freelance translators (Acclaro TM, 2011).

2.1.2 Internationalization

Internationalization is defined by Localization Industry Standards Associations (LISA) as the systematic generalization of a software product such that it can handle multiple locales. In other words, the product is designed in such a way that say if tomorrow a fourth or fifth target locale needs to be added, it can be done so without changing the product architecture and code.

So why are products internationalized? Esselink (Esselink, 2000) identifies two key reasons why software publishers and developers internationalize their products. First is to ensure that the application can be sold internationally, which is key for company growth as mentioned before. Second is so the application can be localized to a new target market without the need for design or code changes, which present tremendous commercial leverage over competitors by minimizing localization costs. The strategy behind technically internationalizing a software product, regardless of development technology or target market, is to identify and externalize all software components (user interface controls, user messages, user input methods, user interface styling and so on) so that they are separate from the product source code and can easily be translated by non-technical personnel. For example, this can be achieved in Android by utilizing the frameworks recommended external XML files to store user interface strings such that each target locale has a separate XML file, which is selected at runtime according to the users' desired user interface language. Another example of a product internationalized for all worlds markets is one where the developers ensure built-in support for all world characters, for example by using Unicode encoding, which is double byte encoding with support for all world languages, instead of ASCII, which only supports Latin characters. Other strategies for internationalization exist, which is presented in Chapter 5.

2.1.3 Localization

The ISO and IEC defined Localization as "a process of adapting an internationalized application platform or application to a specific cultural environment". In other words, as explained before, localization in practical terms means translating the externalized software components to suit a new locale. Examples of localization activities include: translating all the user prompt messages and user interface components such as labels, replacing icons to suit the target culture, and changing the currency symbol.

Traditionally, localization was done once an application was completed and even deployed. In reality, this may be the case for all products that were developed for one target culture without consideration for future growth beyond local boundaries. Nevertheless, it is recommended to internationalize and localize all

software products that are still under development. Not only that, if possible, all software products already developed (and being used) should be internationalized, as opposed to modifying the product on ad-hoc basis to support new languages. When software is localized during the development phase, notwithstanding the additional costs, the software publisher can benefit significantly by simultaneously releasing multiple localized versions of the product in different markets targeting a far larger audience, an achievement known as Simultaneous Shipment, or SimShip (Esselink, 2000) (Hogan, Ho-Stuart, & Pham, 2004).

2.1.4 Software engineering practices

Among the other aspects of software development, the requirement of developing internationalized software has a tremendous effect on software engineering activities such as design and coding. Naturally, the development and target environments influence these activities. For example, developing application for Windows, Mac OS, Linux, or mobile platforms will affect the design and coding activities needed to meet the internationalization and localization requirements. Different platforms and software engineering is discussed in terms of multilingual software development more specifically in Chapter 4, here the focus is on more general issues. In particular, issues and topics discussed include the role of the Localization Engineer, the software engineering skills required to develop multilingual software and elements of the software product that are affected by localization (Esselink, 2000).

To start with, teams 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. In addition to general software engineering competencies, a localization engineer should be competent with:

- Different operating systems in terms of support of world locales
- Internationalization and localization techniques and tools
- The available character set issues such as ASCII and Unicode
- Usage and utilization of computer aided translation tools
- Different spoken languages, especially foreign to the development team

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.

2.1.5 Software project management

In terms of project management for multilingual software projects, the number and diversity of stakeholders increases tremendously. The project manager traditionally plays a central role and is responsible of carrying out of management related tasks. In agile projects, since the role of project manager is dissolved among team members, the corresponding roles are assigned to different team members or to the whole team. In multilingual software projects, additional stakeholders are introduced and collaboration and communication with additional them may complicate things further. So what project management related skills and experience is required in multilingual software projects (Esselink, 2000) (Schwaber, 2004)?

In addition to the general project management skills, the involvement of one or more team members with experience in multilingual software development is important. In traditional sequential projects, the project manager is expected to have such previous experience and availability of localization engineers can be beneficial. In agile projects such as Scrum, either the scrum master or other another team members should have some experience and should help others in internationalization or localization related issues. These individuals should have experience in dealing with third party localization firms, if relevant, and should be familiar with localization techniques and tools in terms of their limitations and benefits.

2.1.6 Software quality assurance

Quality assurance and pre-release testing are crucial activities for successful project. Not only they improve software quality, but also save tremendous amount of money for project developers as the cost of post-release bug fixing is exponentially higher (Galin, 2003). Multilingual software required additional testing and quality assurance activities that are not traditionally documented or taught in quality assurance lectures. Cosmetic and linguistic testing are two examples of such kinds of testing (Esselink, 2000). Cosmetic testing focuses on the quality of the localized versions of the application, in particular, the user interface elements. Linguistic testing refers to verification of the translated content. This is discussed further in Chapter 5.

2.1.7 Documentation translation

Multilingual software is obviously targeted at users who speak different languages and come from different parts of the world. Any software project would produce some documentation, so how are such documents affected by the international nature of the final product? Development related documents need not be available in multiple languages. However, documents that are used by end users must be also localized. For example, the user manual, contact form, online help and even customer support are components that should surely be translated (Esselink, 2000).

2.1.8 Graphics translation

The use of graphics inhibits localization of software applications and therefore graphics should be used very carefully. Often developers embed text in graphical images, which makes it impossible to translate without replacing the whole image. Also, sometimes the navigational menus are implemented using graphical

elements such as arrows, which cause problems if the application is localized into a language that is right to left, for example. Nevertheless, sometimes the project is such that graphics must be used, and some relevant guidelines are presented in Chapters 4 and 5.

2.1.9 Translation technology

The term computer aided translations, also known as CAT, has evolved since early 1990s and today numerous tools exist to help translate content from one language to another electronically. While most users today are familiar with web based services like Google Translate, Microsoft Translator and Babel Fish, localization firms utilize additional diverse tools. Firstly, it is important to understand that these services are categorized as machine translation, or MT, services with the main goal of translating content. In spite of years of investment and diverse approaches in this field, the quality of machine translation is low. Professional software publishers still rely on human translators to ensure high quality in localized versions of their applications. On the other hand, CAT tools are used by translators to support them carry out their work. It makes them better and faster translators.

Computer aided translation tools are further categorized as Translation Memory (TM), Terminology or Software Localization tools.

2.2 Motivation

To summarize the motivation to compile, organize and methodologically present guidelines for multilingual software development are as follows:

- Multilingual software applications tremendously increase the potential target market or audience and as a result provide the opportunity for great commercial and economical gains.
- A large number of monolingual software applications, for example developed in Sweden and for Swedish speakers, can be as successful in other countries around the world. If such applications can be cost effectively and with good quality localized for new markets, great opportunities exist.
- Large software publishers often target a wider audience and have almost finessed the localized versions of their products as well as the localization process. As a result, a large industry has emerged and numerous practices and tools have been devised that facilitate with the process.
- Small and medium sized publishers ignore internationalization and localization of their products and are often oblivious to the potential benefits of entering markets beyond their boundaries.
- All software publishers, especially small and medium sized ones, can greatly benefit with the availability guidelines for multilingual software development and maintenance in one place. The benefits are multiplied if only the relevant guidelines can be easily retrieved.
- Electronic data and information, especially that available on the internet, can be made accessible to all people around the world regardless of their language using the available technology above. If all software publishers internationalize their software, without necessarily localizing it to too many

target locales, this can be a fundamental stepping stone towards a future with electronic information accessible by all.

3 Methodology

This chapter explains in detail the selected research methodology for this project. First the background of the selected research approach and several of its activities are discussed and described, and then the specific research methodology followed in this project is outlined and explained, including how the literature has been reviewed.

3.1 Study approach

The study is primarily qualitative in nature but a quantitative method is used to validate and get feedback on the results. The aim of this study is the compilation of multilingual software development practices and presenting them in form of guidelines for project team members who wish to internationalize the software applications they are working on. It is also to present the guidelines in an organized manner with a suitable method that they can easily and quickly be retrieved. Since the research method is influenced by the nature and aims of the research, a qualitative method is best suited for this study. The popularity of qualitative research has increased in over the last couple of decades, especially in the field of social sciences. Nevertheless, in order to validate the results, a quantitative approach was utilized (Hennink, Hutter, & Bailey, 2010).

The aim of a qualitative study is to gather an in-depth understanding of human behavior and try to understand the why, how, who, when, where and what related to the behavior. Unlike, the quantitative approach, this approach attempts to understand the why and how, not just the what, when and where. This is the reason a small sample, even a single case study, may suffice to complete the research. In other words, a large sample size is not needed to generalize the findings (Denzin & Lincoln, 2005). In the context of this research project, the purpose is to understand why multilingual software are and are not developed in terms of motivations, technical complexities and so on. While the emergence of qualitative methods created a new dimension to research and academic studies, there are key disadvantages that must be tackled. For example, determining validity or reliability of the results can be subjective; based on the researchers' experiences, predispositions and knowledge. Due to the subjectivity involved in data analysis, the influence of the researcher is inevitable. In this study, quantitative methods are used to verify and validate our results and findings. Chapter 6 presents this in detail. As far as the subjectivity of the researcher is concerned, the suggestions provided by Monique Hennink et al (Hennink, Hutter, & Bailey, 2010) is followed to try to understand and outline our influences on the research and its various activities. In fact, one of the authors participated in the interviews.

The interview in qualitative studies is defined as an interview with the goal of extracting descriptions on the topic of interest from the interviewee. There are numerous ways of carrying out interviews and the most famous and effective way being face to face interview. In this research, mostly online interviews were carried out and just three were face to face interviews. This was due to logistical reasons and the unavailability of interviewees (Hennink, Hutter, & Bailey, 2010). Structured interviews with open ended

questions yielded a large amount of information about the sample projects from the perspective of multilingual software development.

The expert opinions from a focus group are utilized to verify and validate our finding objectively. Focus group is a form of qualitative research in which a group of people are asked about their perceptions, opinions, beliefs and attitudes towards a product, service, concept, advertisement, idea, or packaging (Henderson, 2009). Krueger (Krueger, 2000) defined a focus group as *"A carefully planned discussion designed to obtain perceptions on a defined area of interest in a permissive, nonthreatening environment"*. A focus group basically attempts trigger points of views through a discussion among experts, but one of the disadvantages of this method is the difficulty to bring together experts together, and unfortunately this was encountered in this project. Therefore, instead of a focus group, a survey with case scenarios replicating the possible discussions that could have triggered in the focus group was created and quantitative feedbacks from numerous experts were collected. The themes prepared for focus groups were developed into Likert-scale online surveys. This type of scale asks participants to respond to the statements by ranking it to the degree which they agree or disagree (strongly disagree, disagree, neutral, agree, and strongly agree). A number of diverse scenarios representing software applications were prepared and the experts were asked to rank the corresponding guidelines (for the multilingual software development). This provided valuable feedback on the guidelines as well as the general grouping of guidelines for unique scenarios.

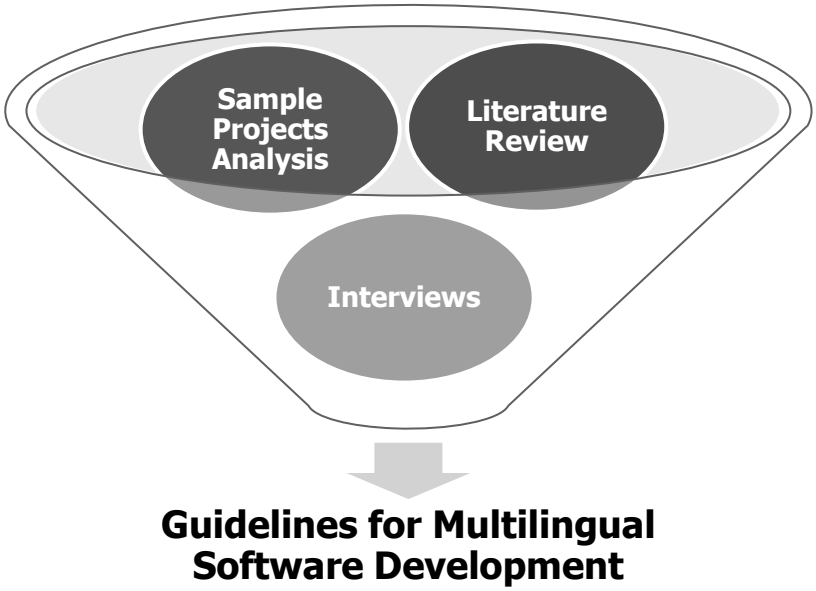
The advantages of using this method include: cost effectiveness, quick to respond to, easy to participate, convenient as the respondents can fill it at any time or place, it is easy to statistically analyze close-ended questions, reduced interviewer bias. The main disadvantages include: inability to gauge the focus and seriousness of the respondent and inability to trigger discussions among experts, unlike the originally planned focus group.

3.2 Methodology followed

The methodology followed to carry out this research consists of five main steps:

1. Literature review and extraction of guidelines for multilingual software development.
2. Selection of suitable sample software applications for study and analysis.
3. Interview selected applications' project team members.
4. Compilation of the extracted multilingual software development guidelines and its presentation in an organized manner.
5. Validation of the results.

This is illustrated in the following model:



Model 3-1: Research Methodology

Early in this study, a comprehensive and systematic review of the literature on multilingual software was carried out. Details of the reviewed literature are available in Chapters 2 and 4. The focus of the study had to be refined as the project progressed with the literature review. In other words, the literature review itself has affected the research methodology and results. For example, 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. This prompted us to change the focus of the research and accordingly the research methodology. The affects of the literature review on the research project is discussed in detail in Chapter 4.

The purpose of the literature review, as always, was to understand and document other studies on the topic of multilingual software. This activity also enhanced and deepened our knowledge of the topic, which helped us realize that what software developers truly need is awareness and straightforward and relevant guidelines in order to increase the number of international software. Additionally, guidelines relating to multilingual software development were extracted from the reviewed literature and assigned to one of the specified categories, such are feasibility study, development lifecycle phases, post release, budget, benefits, and so on. In essence, this was also a data collection activity and over thirty publications were reviewed on the topic. In this sense, the literature review is also a research method; i.e. it enabled the survey, categorization of guidelines for multilingual software development and later conceptualization of ways to make the guidelines accessible. This is in line with what Torraco states about integrative literature review (Torraco, 2005) in that it: "is a form of research that reviews, critiques, and synthesizes representative

literature on a topic in an integrated way such that new frameworks and perspectives on the topic are generated".

While the literature review phase continues from the start of the research project to about half way, it was also decided, early in the project, on a number of sample projects to analyze and study. Initially, there were problems finding cooperative respondents but with persistence, such as emailing over 200 software project owners in Sweden and Oman, and improved communications, team members of eleven industry projects, mostly from Sweden, were interviewed. As soon as a positive response was received, which was not very often, a preliminary study of the application or website was conducted to extract guidelines and understand the project more in order to decide on what sets of interview questions were relevant to the project.

The initial plan was to interview at least ten sample projects, including both the multilingual and monolingual projects. Eventually 11 interviews were carried out, out of which 3 were monolingual projects and 8 multilingual. This was done to confirm our analysis and extracted guidelines as well are extract additional guidelines and identify problems in the industry. This included three face to face interviews and the rest were online interviews, all of which were structured and consisted of mostly open ended questions. The industry sample projects eventually studied included: bilingual (in Swedish and English) web portals of three Swedish universalities, the websites of six Swedish government agencies (two monolingual and four multilingual), one open source software, and one bespoke distributed software project (Muhammad Murtaza, also author of this paper, was the project manager and lead programmer in this project. This project was included to increase the number of sample projects, especially since most of the other projects were CMS implementation in Sweden, and because the interviewees supplement the guidelines for multilingual software development extracted from literature. This project was implemented in Oman and in Arabic, so it provided unique inputs). These interviews and analysis of these projects are available in Chapter 5, and Chapter 7 discusses the limitations and possible implications of these sample projects. Sample of the questions asked during interviews is provided in the Appendices.

After analyzing the data it was found that some interviewees did not give the appropriate answers to some questions. The reason could be that the interviewee did not have enough knowledge in the topic to answer the technical questions. Another reason could be that the interviewees misunderstood the questions. On the other hand, some answers were very detailed and technical and this was generally when the interviewee was software developer. Many respondents left the questions unanswered.

Once the interviews were completed and a large number of international software development guidelines had been extracted from the literature, the categorization of these guidelines started in a logical manner. For example, if a given guideline recommended requirements specification formats for multilingual software, it was documented under the category of Requirements Engineering, which in turn was under the category of multilingual software development and software lifecycle. Next, it was realized that it is not feasible for project managers and other team members to read the whole study and the derived guidelines. After all, one of our key goals was to encourage the development of international software and a main research question

was about what could be done to change the status quo of developing monolingual software or localize the application on an ad-hoc basis, without considering industry practices. Therefore, after a significant brainstorming effort, the "Guideline Navigation Tool" was devised, which basically linked guidelines for multilingual software development to real-life project statuses, requirements and other conditions. After using the Guideline Navigation Tool for a particular project, the project owner is given a handful of relevant guidelines to help internationalize and localize their software projects.

Finally, in order to verify our guidelines and the results derived from the Guideline Map, carried out a survey with experts to gather valuable feedback on our findings. A likert-scale online survey was used where a number of diverse scenarios were listed representing software applications. The experts were asked to rank the corresponding guidelines. These case scenarios are available in Appendix B.

The guidelines for multilingual software development are extracted from published literature for the most part. Some guidelines and contextual understandings are based on the selected industry sample project. For this reason, and due to the fact that it was logistically very difficult to group experts on multilingual software in the same place and time, it was decided that general feedback on the guidelines would provide valuable feedback, if not validate the results. Additionally, the respondents were academicians and industry practitioners in the field of software development, but not experts in internationalization and localization (in fact, many responders left the survey uncompleted). This further weakens the validity and the feedbacks.

3.3 Literature Review Description

In this section, the effect the literature review had on this project is discussed. Also, the specific methodology followed to carry out the literature review is described.

3.3.1 Effects on the thesis project

Based on our past work experience in multilingual software development, where we often found ourselves experimenting with different techniques, we felt there was a need to find a general and technology independent design and solution to multilingual software development, like an architecture or design pattern. It seemed strange that in spite of the numerous benefits (Esselink, 2000) (Wooten, 2010), so many software projects fail to consider internationalization. Therefore, early in the project, it was decided that this project must contribute towards facilitating internationalization and localization of software applications.

As a preliminary literature review before writing the thesis proposal was carried out, it became clear that numerous and diverse solutions were used in different projects. At this point of our project, it was felt that instead of finding a generalized design solution, it was important to first identify and categorically list the different methods as well as their advantages and disadvantages. Thus, the thesis proposal mainly focused on the identification and listing of the existing methods used in industry to internationalize and localize software as well as the design and implementation practices that inhibit the localization of software. It was

believed that such a work would contribute by helping designers select the most suitable and cost effective solution to internationalize their software by comparing the different practices.

The next and final major change in the focus of the study occurred during the detailed literature review and it was because of two main reasons. Firstly, a large number of technology dependent and technology independent design and architecture solution already existed. Secondly, internationalization of software is not a problem that could be simply solved using only the right architecture or design; it also depends on the underlying operating system, the language support provided by the database, and the implementation technology (Abufardeh & Magel, Software Internationalization: Crosscutting Concerns across the Development Lifecycle, 2009). In other words, the operating system, database system, the hardware and the implementation technology have to provide support for the characters of different languages as well as the direction. Considering this, it seemed the problem was not a lack of design solution or implementation technology. In fact, even published frameworks and guidelines exist that help incorporate software internationalization practices into different implementation methodologies (Young, 2001) (Abufardeh S. O., 2009) (literature related to implementation methodology was reviewed because one of our original research goals was to recommend practices for different development lifecycle phases to encourage consideration of software internationalization and localization). Similarly, a published architecture reference model and architecture pattern exists (Venkatasamy, 2009), which originally was a planned contribution of this work. 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? The answers to these questions became the new focus of our study.

Each project is different when it comes to internationalization and localization (Esselink, 2000), and if the goal is to facilitate the internationalization of software, this project must consider as many cases as possible and provide relevant guidelines for internationalization and localization. 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.

It was assumed that most local software developers ignore internationalization and localization because of their lack of awareness about existing technical solutions and potential commercial benefits of software that can be used by an international audience. In order to confirm this, the selected sample projects were analyzed and it was attempted to understand the motivations of project members through interviews. After this activity, there was a conviction that 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. Therefore, a comprehensive list of

guidelines for multilingual software development, presented in a suitable manner with associated implications, can indeed both raise awareness and encourage software developers to internationalize and localize their software by reducing uncertainties.

These guidelines must take into consideration factors and questions such as: is a project still under study (feasibility study)? Is a project still under development (pre-release)? Has the software application already released (the project is in maintenance phase)? Are the project team and source code accessible? What development technology was the project implemented in? Does the architecture of the product take into consideration future localizations (was the product internationalized)? Also, other important factors such as the available budget, number of target translated languages and so on.

Eventually it was felt that concise and pragmatic guidelines that are relevant to specific project and are presented in an easily accessible manner facilitates software internationalization, regardless of what stage the software is in its lifecycle.

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 an bird-eye view of what needs to be considered and what solutions are available for their projects. Therefore, it was decided that this would be the new focus of this project and the methodology was revised accordingly.

3.3.2 Method used for literature review

In this section, the method used when reviewing the published literature is explained; this was divided into three phases:

1. Preliminary review: this was done when writing the thesis proposal and during the early stages of the project. In this phase, a random search took place for published and unpublished (mostly from technical tutorials and commercial internationalization services) information about multilingual software development.
2. Explorative review: This was done at the start of literature review and the main goal was to search and categorize published literature on multilingual software development. Google Scholar, IEEE Xplore and ACM Digital Library were search using keywords mentioned in section 3.5.2.2. The abstract of each paper was read and the paper was put it in one of the categories mentioned in section 3.5.2.1.
3. In-depth review: this took place during the literature review and in this phase the selected papers were read in more detail, category by category. The parts read included the introduction, the conclusion and relevant sections (to each category) of each document. Papers that seemed very relevant and useful, their literature review sections were also read. Some papers provided references to other relevant published literature, which were added to the reading list. While reading, possible guidelines for multilingual software development were highlighted, extracted and written down.

3.3.2.1 Categorization of related literature

As mentioned, the literatures were divided into a number of categories. Categorizing the extracted guidelines from the literature as discussed below later helped map each guideline to one or more software project property. Project Properties (PP) reflect possible attributes of different software projects and examples include: number of desired target language, project budget, content size, availability of source code and so on. These guidelines are presented in Chapter 5 along with "Guidelines Navigation Tool (GNT)" (a term used to refer to the table-like model devised in this paper in order to facilitate the process of retrieving only the relevant guidelines and each GNT consist of project properties to help link the guidelines to the project at hand). Please refer to Chapter 5 for details of GNT and PP.

The categories for the extracted literature are as follows:

- Economical and financial: this category included literature about the financial, technical and other benefits of multilingual software. It covers publications, reports and articles about multilingual software in terms of costs, benefits and perceived motivations. The aim of reviewing such literature was to confirm the importance of the topic as well to understand the reasons for and related costs of internationalizing software. The extracted guidelines are especially relevant to pre-project decision making; help stakeholders during feasibility study whether internationalization of the project is feasible, and if so, what are the main cost components?
- Administrative and managerial: in this category the literature about multilingual software development in terms of administrative and managerial implications is covered. The goal was to extract guidelines for project teams and companies that would help understand the best ways to structure teams, utilize services of third-party service providers, such as translation companies and localization firms, as well as free lance translators. Depending on numerous Project Properties (PP), such as the static content size (content that needs translation) and desirable target languages, the extracted guidelines differ.
- Multilingual software development and SDLC: in this category, the added papers discuss the implications of multilingual software development to any of the Software Development Life-Cycle (SDLC) phases, including software development methodologies. This category is the most important one and majority of guidelines are extracted from literature in this category. Reviewing literature on this topic helped us understand the overall effects of internationalization on the implementation process and deduce relevant guidelines for internationalization for different implementation phases. Literatures in this category are sub-categorized into the SDLC phases: Feasibility Study, Requirements Engineering, Architecture and Design, Software Programming, Testing, and Maintenance (or Post-release).
- Technology or vendor dependent: many guidelines are unique to specific implementation technologies such as J2SE, Flash, .Net, Objective-C, PHP, or Android. Other solutions are unique to specific frameworks or Commercially Off The Shelf (COTS) products. This category includes

literature relevant to such technologies or products. Naturally, not all technologies and platforms are covered, but the main ones are.

- Miscellaneous: this category includes ad-hoc yet relevant and useful papers and case studies.

The guidelines from literature in one category often cross-cut numerous Project Properties, as show in Chapter 5, and as a result, often a project with a small budget (an economical and financial project property) may result in retrieval of guidelines extracted from literature categorized as administrative financial. This means that the literature categories influence the eventual categories of the guidelines for multilingual software, but do not affect the project relevant guidelines that are retrieved from the Guidelines Navigation Tree (GNT) based on the selected Project Properties (PP).

3.3.2.2 Electronic database search keyword

Google Scholar, IEEE Xplore and ACM Digital Library were searched with the following keywords, which are also briefly described:

- Software internationalization: this is a term commonly used in technical contexts when referring to software that are specifically designed to support more than one language user interface (Esselink, 2000).
- Software localization: this refers to the actual activity of customizing a software application for a particular locale, culture and language. It is often said that internationalized software can easily be localized (Esselink, 2000). This does not only refer to the words but also to numbers, currencies, working days and other issues that may be different from one country to another.
- Multilingual software: This is a general term used in both technical and non technical domains such as sales and marketing to refer to software that support more than one language and can be adapted for numerous locales (Venkatasamy, 2009) (Esselink, 2000).
- Software globalization: This process to combine all the other three terms. It also considers the globalized nature of many software development projects, where team members may come from different parts of the world. This aspect might not be particularly relevant to software internationalization, so related papers were filtered out. Also, it is commonly said that software globalization focuses on the commercial and administrative sides of multilingual software development (Esselink, 2000) (Abufardeh & Magel, 2010).

Additionally, it was tried to search for "multi language software" keywords but this failed because it resulted in papers on the topic of developing software using different computer languages in the same project, like Java, VB.Net and so on.

These three resources were searched after a discussion with the project supervisor. In particular, IEEE Xplore is a rich source of technology related literatures. ACM Digital Library returned a good number of papers, compared to other general sources, when searched with the relevant keywords, thus it was used as one of the primary sources. Finally, Google Scholar has emerged as a tool that retrieves numerous

literatures from diverse sources, and given the time constraints, it is important to use such a tool. Google Scholar returned a large number of literatures that were earlier retrieved from the previous two sources, which confirmed their selection, but it also returned a number of relevant books and papers published in other sources and this helped us collect a wide range of literatures. To extract multilingual software development guidelines related to different technologies and systems, the internet was searched using Google with keywords “XYZ Internationalization”, “XYZ Localization” and “XYZ Multilingual Software”, where XYZ is the programming language or system name, like Java, C# or InfoGlue CMS.

3.3.3 Scope of the literature review

The main concern of this paper is not the specific techniques, platforms or solutions available for multilingual software development and this is reflected in the reviewed literature. For example, while papers and case studies where machine translation was used to translate the content of the software are relevant and interesting, papers on the topic of machine translation and its different approaches are out of scope.

3.4 Ethical consideration

No ethical approval was required for this study.

4 Literature Review

Development of multilingual software is not one dimensional; it requires support and consideration for world languages on multiple layers. Taking into consideration a contemporary example, if we were to develop a trivial multilingual application for Android operating system using Java programming language, it should be quite straightforward due to Java's support for multiple languages and Android SDK's suggested practice of maintaining all the interface resources in a XML file, which can be translated into different languages. Similarly, we may develop a website or application using web-based technologies and optimize it for mobile phones such as those that run on Android. Nevertheless, in spite of doing all that is necessary to develop the application to support multiple languages, it could still not run on specific Android-based mobile devices. In certain cases, maybe some of the supported languages would run as expected on Android while others would not. There could be a number of reasons for this.

For example, Android 2.2 (codenamed Froyo), which was the latest release of the operating system in 2010-2011 did not support Arabic characters and replaced them with mysterious square symbols. Android 2.3 (code named Gingerbread), which was the latest release at the end of 2011, supported Arabic characters but did not display them properly and each character in a word was disconnected (characters in Arabic, Farsi and Urdu are joined to form words, somewhat like a running and connected handwriting in English).

However, the same versions of mobile devices sold in the Middle East supported Arabic perfectly due to the mobile vendors like Samsung customizing the platform before selling them in the Arabic speaking region and due to the availability of numerous third-party installations for Android (Alsanad, 2011). Naturally, eventually the default version of Android will support Arabic and numerous other languages, but this is one example of how software internationalization is not one dimensional (it is not enough to worry about internationalization during the implementation phase, one must also consider the target platform). It is an example of how internationalization can demand consideration of numerous issues within a specific phase or dimension. To clarify, in this case, it wasn't sufficient the developers designed and implemented the software application to support multiple languages. They had to also consider the support provided by the target platform.

A large number of literatures exist on the topic of multilingual software development from different perspectives. Some of the publications focus on specific technical solutions and implementation methodologies, while others are more general in scope. Reviewing and cataloging such publications helps one understand how software internationalization and localization should be done and what its implication are in general and specific circumstance. This is what was done, but additionally, while carrying out the literature review, specific, relevant and proven guidelines were derived for software internationalization, which together formed the main contribution of this thesis project.

The following subsections present the reviewed literature:

4.1 Introductory literature

Quite a bit of introductory information from literature was presented in Chapter 2. Additional introductory information, with relevance to software development and implementation, is presented here.

Three common areas of software are identified (Abufardeh S. O., 2009) where internationalization must be considered. These areas are:

1. Locale and culture, which refers to the software elements affected by the norms in a country or geographical region. This includes the calendar, date, time, currency, phone number, address, number formats, measurement system, and the spoken language (At times the Input Method Editor (IME) may be needed when the keyboard does not support language characters used in a locale). For example, in Middle East, not only the language of the calendar is different, but also the complete dating system. Similarly, the weekends are generally on Thursdays and Fridays, or Fridays and Saturdays in many financial institutions. In Sweden dates are generally noted as 8 digits, where the first four (or two if shortened) digits represent the year and the next four represent the month and day. In the same manner, the address and phone number formats differ from country to country. Internationalized software must take into consideration the norms of different target markets in order to compete internationally.
2. UI and documentation, which refers to the visual components of the graphical user interface – also known as GUI or UI – that the user interacts with and sees. This includes windows, forms, menus, toolbars, error messages, status bars, tool tips, online help, images, icons, colors, sounds, text directionality and layouts. For instance, the directionality of software localized for Bidi speaking countries is right to left, and the software layout must reflect this.
3. Data storage and text processing, which refers to technical and algorithmic elements of the software that may be affected or changed when the software is localized. This includes character classification and transliteration, regular expressions, encoding schemes, collating sequence or sorting, and byte processing. For example, if one components of the software is required to display the list of customers alphabetically, how will the software handle this for different languages; i.e. the collating sequence? Similarly, regular expressions used in code for searching may have adverse effects when the locale changes from UK to China and the language from English to Cantonese. Finally, the size of text and data being processed and stored may be significantly higher for non-Western languages because the characters are stored using two bytes, as opposed to one.

The main idea or trick behind internationalization of a software product is the extent to which these three areas are separated from the core functionality code. To gauge a given application's level of internationalization, or how well it has been internationalized, one may analyze the software to check for elimination of reference to culture, politics and history. Also, the use of images (especially with embedded text) should be non-existent. Finally, the source code should be analyzed to ensure use of only controlled language-dependent content (no hardcoded language text) (Abufardeh S. O., 2009) (Esselink, 2000). Many

successful products have been developed without technical consideration of internationalization and although it is possible to modify an existing product, it is more costly and complex.

How exactly a team should localize (make it available for new locales) a software product that is already developed and deployed depends on numerous factors and characteristics of the project. This includes factors such as whether the source code is accessible (applicable in legacy systems, especially if developed using third-party of obsolete technologies), whether the product was internationalized (if yes, then it is simply a localization project without requiring major changes in the code), what programming language was used (how does the technology support localization?), the availability of localization tools. Additionally, the number and diversity of the desired target languages and availability of localization expertise will influence the decision. Finally, cost related issues, such as the cost of translating the content into different languages and the hours needed to technically modify the product, are of utmost importance in a business environment and the final decision will often depend on the answers of this factor. Chapter 5 presents a tool to navigate guidelines for multilingual software development that are relevant to the project at hand. The project owners can use this tool (called Guidelines Navigation Tool, or GNT) by linking their respective projects to numerous characteristics such as those mentioned above.

In addition to the above, during development, the implementation team should also be familiar with a number of fundamental multilingual software terms, standards, practices and concerns. Standards, technologies and guidelines are available from different organizations concerning multilingual software and often these are meant to enable multilingual software on a low level computing, which is handled by the hardware, operating system and the underlying programming language. Some of these standards (Wikipedia Character Encoding, 2011) (Indiana University, 2008) (Oracle Sun Developer Network (SDN), 2011) (Wikipedia Translation Memory, 2011) are concerned with representation of Western and non-Western characters in binary (also known as character encoding) and others with software localization in industry. These include:

- ASCII: it stands for American Standard Code for Information Interchange and it is a character encoding scheme used for English alphabets and special characters. It is a seven-bit encoding that represents 128 characters used mostly in American English; so for instance, the British Pound symbol is ignored.
- ISO 8859: this is an eight-bit extension of ASCII developed by ISO. It includes all 128 characters represented by ASCII as well as additional 128 characters. This standard has numerous variations, each adapted for a different locale. Examples include: Latin-1 for Western European languages, Latin-2 for Eastern European languages, Latin-5 for Turkish, 8859-6 for Arabic and so on.
- Unicode: this standard was developed by ISO and the Unicode Consortium in order to represent all possible characters used around the world using a single encoding scheme; i.e. Unicode. The latest version represents up to 109000 characters. While ASCII uses one byte (8 bits) to represent a

character, Unicode uses one, two, three and even four bytes depending on the target language. This means that Unicode documents may consume twice or more the memory compared to ASCII.

- UTF-8: is the leading Unicode Transformation Format (UTF) – others, which are not discussed include UTF-1, UTF-7, UTF-16 and UTF-32 – and it is the main character encoding scheme used for web based applications and it represents every character in the Unicode scheme. It is increasingly being used as the default scheme in different operating systems and when developing cross-platform and web-based systems. Because the first 128 characters in UTF-8 are 8-bits and correspond one to one with the older ASCII, this makes it backward compatible and efficient. Since UTF-8 encodes all Unicode characters, content in multiple languages can be stored, transmitted and displayed at the same time in the same file or window. This can be done, unlike ASCII extensions, without the need for a code page, which indicate the language or encoding scheme in use.
- Multicode: is an encoding system suggested by a researcher (Mudawwar, 1997) for multilingual computing, claiming it addresses many of Unicode's drawbacks. It did not gain industry acceptance, but this shows that improvements in lower level multilingual computing are possible.
- TM: is Translation Memory, which is a general term that refers to a database that stores texts that have previously been translated. They are meant to facilitate content translation.
- TMX: is Translation Memory eXchange and it is a standard used by suppliers of translated content to exchange translation memories (databases; see TM above) among one another.
- XLIFF: is an abbreviation for XML Localization Interchange File Format and it is an XML based standard created to unify localization in the software industry. Numerous tools are available to create and manage XLIFF files. This standard allows translators to focus on the translation and makes the engineering of internationalized software easier.

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.

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. Also, competition and the need for efficiency require adequately internationalized software products such that it can quickly and with minimum effort be localized for new target markets. Therefore, internationalization and localization of software must take into consideration more than simply the presentation layer or the user interface (Abufardeh & Magel, 2009). Multilingual software must take into consideration and make available in multiple languages the user interface, dynamic data retrieved from the backend, number formats that vary around the world, currency symbols used in the country, and currency digit formatting, using a comma or a dot, for example.

4.2 Economical and financial literature

In practice, often decision makers and project team members work under various pressures throughout the software project lifecycle. This could be pressure from top management, customers, competition, contractual obligations, and so on. This could lead to the team focusing on core functional requirements and ignoring other non-functional and long-term requirements. An example of this could be the ability to easily localize the software application for different locales. Therefore, it is important to know what benefits multilingual software development presents both in the short and the long terms, as well as understand the key cost components for a multilingual software project. This will help the team make better decisions.

It is a well established fact in the industry that without multilingual software it is difficult, if not impossible, for businesses to complete in the industry. Availability of software in multiple languages is essential to compete in the global IT market and customers are four times more likely to purchase a software product if the content is in their native language (Welzer, Golob, Druzovec, & Kamisalic, 2005).

In the technology world and more specifically the software industry, if a firm has a successful product, it is generally better to expand in new markets with the same product than it is to develop new products that may not necessarily succeed. Of course, this is not true for all companies and all kinds of products, but if we were to think about the idea of developing any new product, there are always risks relating to the implementation and even worse the difficulty of marketing and selling a new product. Developing new products, especially for smaller organizations, can also lead to loss of focus. Nevertheless, entering new markets poses its own technical and business challenges, and the internationalization and localization of the software product for the given market is one of the challenges (Jantunen, Smolander, & Gause, 2007). Nevertheless, with numerous tools, localization firms, and a mature industry, it is possible today for small or medium sized businesses to grow by entering into new markets around the world with localized versions of their products. This can be achieved today more than ever before because today the internet is available virtually in all countries of the world and the online users community has been constantly growing.

No longer is majority of software developed in the United States and for economical, educational and manpower related reasons, a large number of software is developed today in China, Japan, Pakistan, India, Russia, United Kingdom and smaller countries such as Estonia. Outsourcing as well as open source software developments are practices that encourage this. On the other hand, a larger number of countries and regions, such as the oil rich Gulf Corporation Countries (GCC) in the Middle East, seldom develop software, and cities such as Dubai in United Arab Emirates have evolved as business hubs through which a large number of international software vendors target their products and services to the region (Ashrafi & Murtaza, 2010). To do this successfully, in addition to customizing the core functionalities, developers must also localize the software. This is why close to 80% of software developed in English are localized and sold internationally and software developers simply cannot compete in the global IT market without internationalization (Abufardeh S. O., 2009).

Researchers and experienced industry practitioners have learned that the usability and overall efficiency of software applications increase if the cultures of the target users are taken into consideration; i.e. if software applications are competently localized. However, software internationalization and localization is time consuming and expensive (Reinecke & Bernstein, 2007), thus one must understand the cost components and devise cost-effective ways to develop multilingual software.

Developers might wrongly assume that the key cost involved in the development of an internationalized software product is an implication of the additional hours spent during development, i.e. the extra effort needed during the design and implementation phases, where the code for core functionality is separated from the code for translatable components of the software. One may even foresee additional costs linked to third-party tools and frameworks or the contractual cost of services provided by the selected third-party localization service provider. However, the main cost component in many projects is the cost of translating the content. This cost increases as the number of target languages and the content increase. In other words, internationalization of a software application is a decision the project owner (customer) must make carefully and there must be enough motivation (or return on investment) to justify the costs because not only the project cost increases when provisioning the application to support multiple languages, the cost associated with professionally translating the content can get out of control (Esselink, 2000). This cost increases with every new language and may become an overhead cost if the application's content is continuously being generated (in case of software that manages scientific publications, for instance). While many software projects are based on customization of commercially off-the-shelf software (COTS) or open source applications, like content management systems, which are already internationalized, the cost associated with localization, and translation of content in particular, still needs to be considered.

4.3 Administrative and managerial literature

New considerations must be made by the project manager for additional roles, tasks, deadlines and contractual obligations related to the internationalization and localization of the product. For example, a multilingual software project may include additional stakeholders such as localization consultants and freelance translators. Proper administrative and managerial practices may lead to significant cost cutting in translation costs and other multilingual software development activities. Additionally, novel approaches may allow teams to localize software applications that they had previously thought as impossible to competently localize due to economical restrictions.

The cost of internationalization and more importantly localization often force developers to ignore multilingual software development and limit their efforts to the development of core functionalities for the native market. While internationalization of a product may still be considered as it mostly requires additional effort during the design and implementation phases, localization costs are generally significantly higher due to the needed content translation and subsequent testing. With more content and target languages, the costs increase significantly. Although it is recommended that the internationalization and localization activities are embedded into the software implementation lifecycle (Esselink, 2000), if the

project budget does not permit localization, the software should at least be designed and implemented as one that is internationalized (at least in theory) to facilitate future localizations. Nevertheless, alternative approaches maybe utilized to localize software applications.

Reinecke and Bernstein (Reinecke & Bernstein, 2007) have suggested a culturally adaptive software, which is different to manual industry localization in that the software is designed and implemented such that it acquires details about the user's culture whilst being used and it adapts itself to the culture; the software localizes while being used. While this approach is not suitable for all software applications, it can prove to be a very useful approach for free and community software. Similarly, crowd-sourcing is another very effective approach to localize even large software applications as it was proved by the internet giant Facebook (Malik, 2009). In this approach, provided a large and international user base is available, the software publisher can utilize them to localize the application into numerous locales. Naturally, in both these alternative approaches, the application will need to be internationalized differently.

A large portion of the cost of localizing software is directly related to translating the content into multiple languages in a professional manner (Arefin, Morimoto, & Yasmin, 2011). As mentioned, one way to maintain high product quality in all localized versions of the product is to create an effective administrative process. This was demonstrated by a Norwegian company that had developed their product in Java and to enter new markets. The company localized the product using the Java I18N standard. Their localization process followed specific administrative steps: first, they located a freelance translator in the target market they were localizing the product for. Next, they flew selected translator to Norway for a short training on how to translate their product. Finally, the translator returned home and translated all of the software and related documents from home, transmitting all the localized material to the company as agreed. For this company that had one successful product and relied on localization of software to enter new markets, this process significantly reduced costs associated with localization and translation of content. The company found this process to be far cheaper than hiring full time employees or contracting a specialist agency and this allowed for continuity and access to new markets (Wigestrand, 1998). Online service providers can be utilized to find translators and other localization resources. Examples include:

- http://www.translationzone.com
- http://www.e-translate.com

As mentioned, the cost of generating and maintaining multilingual content, especially in data driven web applications, is high and this cost increases as the number of target languages or the content size increases. The main reason for this is the need for human professional translators. For websites, this means the creation and generation of HTML pages in multiple languages either statically or dynamically. To address this issue, Arefin et al. (Arefin, Morimoto, & Yasmin, 2011) propose an alternative technique based on natural language machine translation for multilingual content management in a web environment. In this approach, the content is stored in a single language and the contents required in other language versions are translated depending on the user. Although this technique is reported to outperform existing systems in

terms of time and space, further investigation is needed to gauge the quality of the translations and the reduced need for human translators.

Any software product is as successful as its development team, so it is of utmost importance that the project team members work well together. Development and maintenance of multilingual software inevitably means multinational and multicultural teams. Often large software publishers struggle with building such diverse teams locally. Smaller software developers simply cannot afford multicultural teams. Some solutions to this problem include the use of offshore service providers, finding local business partners and building geographically scattered teams. In all cases, the cultures and personalities of team members in such teams must be understood to build successful teams. Team building activities can contribute positively, but awareness of cultural dimensions as suggested by Prof. Geert Hofstede, for example, and personality traits, such as The Big Five framework from Costa & McCrae, are also very important (Abufardeh & Magel, 2010). Moreover, multilingual software development teams with members from different countries and cultures, pose unique challenges to the project (Hashmi, 2011). The geographical distance means a difference in work environments and time zones, which may leads to communication gaps, project delays, inconsistent quality and ambiguity. Cultural differences create mistrust, fear and unequal distribution of work. It also means different languages being used. All of this could increase project costs and cause reporting problems. A suggested remedy to these challenges is the utilization of cloud computing.

4.4 Multilingual software development and SDLC

A large number of different software implementation methodologies are used today around the world. Experts state that larger (in terms of team size and function points), life and mission critical software require stricter methodologies that enforce control and requirement documentation. On the other hand, in smaller and medium projects better results can be achieved using iterative methodologies and processed known as Agile (Cockburn, 2006). Experts have proposed a number of frameworks for different implementation methodologies to integrate internationalization practices into Software Development Life-Cycle (SDLC) in order to solve one of the key challenges to software internationalization (Abufardeh S. O., 2009) (Young, 2001). Discussing these methodologies is out of the scope of this project and we believe that the different methodologies use the same general SDLC phases and practices. The difference between the methodologies is often related to the suggested team structure, documentation, and configuration of implementation practices. Also, some of the methodologies may encourage new practices.

The subsections below present the reviewed literature mainly to derive relevant guidelines for software internationalization with relevance to the general SDLC practices, regardless of the implementation methodology. The subsections are categorized as Feasibility Study, Requirements Engineering, Architecture and Design, Software Coding and Development, Testing, and Maintenance (or post-release):

4.4.1 Feasibility Study

This phase in SDLC determines the likelihood of a project's success before it starts. All software projects, intentionally or unintentionally, go through this phase and multilingual software projects are no different. It is important that the persons involved in this phase understand the implications of developing software for an international audience in order to competently carry out a feasibility study for the project.

With relevance to multilingual software development, while conducting feasibility study for software projects, the team might need to carry out one or more of the following activities depending on the nature of the project (Abufardeh S. O., 2009):

- Carry out a market analysis: this activity refers to studying market trends and size, growth rate, political and economical conditions, and profitability. It includes studying customer needs and the prices and software features offered by the competition.
- Analyze the possible development scenarios: this activity refers to software design, implementation technology and implementation methodology used.
- Analyze the existing software: this refers to the study of an existing software application, including its documentation. It may include reverse engineering the product.

If new software is being developed, then the team will need to conduct market analysis and consider possible development scenarios to internationalize and localize the software. If existing and already internationalized software needs to be localized, then all three activities will need to be carried out. Similarly, in case of existing software that needs to be both internationalized and then localized, all three activities will need to be carried out.

At the end of this phase, it must be decided whether the software project is worth the effort and the budget for the project budget may be determined. Numerous alternative options for implementation can be considered to control the project cost and this include internationalization and localization activities. For example, among key decisions to be made is whether to outsource localization activities or carry all activities in-house, with each having advantages and disadvantages. This decision is influenced by factors such as content size, desired target languages, in-house expertise and the financial proposals submitted by localization vendors.

Similarly, during feasibility study of a multilingual software project, the availability of internationalization and localization tools, translators and localization engineers should be factored into the equation. Translation costs are often very high and dependent on target languages and content size, so for an accurate result in this phase, it is important to study the costs of available translation services in the market.

4.4.2 Requirements Engineering

Requirements engineering is in essence the first practical SDLC phase and it helps the development team understand what software application needs to be developed; what functions and features should the

software application consist of? In large and critical software projects, especially when a rigorous (non agile) software implementation methodology is being used, the success of this phase often determines the eventual success or failure of the entire project. Requirements engineering includes sub-activities such as requirements elicitation, requirements prioritization, requirements specification (documentation), and requirements change management. The requirements of software are categorized as either functional or non-functional (or quality requirements) (Davis A. M., 2005). In agile methodologies, these activities might be carried out less rigorously and the complete requirements engineering phase is repeated numerous times for each requirement (or user story) and during each iteration (Cockburn, 2006). For multilingual software development projects, there is a need to enhance this phase's activities and resulting artifacts in order to facilitate internationalization and localization activities throughout the SDLC. This is also important to manage issues and challenges created in requirements engineering activities due to the multilingual nature of the software application, the project stakeholders and the communication among team members.

The most important requirements of multilingual software that must be elicited during requirements engineering can be determined by asking: What languages shall be supported by the software? How will the software application switch from one language to another (statically; needs restart, or dynamically, without restart)? What modules must support multilingualism? On what level should the software, or its individual modules, support multilingualism (only user interface or backend processing and storage)? In particular, the following quality requirements are applicable for multilingual software (Venkatasamy, 2009):

- Maintainability: to what extent can the multilingual components be modified post-release?
- Reusability: can a developed multilingual component be reused?
- Understandability: how well the different localized versions are understood by native users?
- Adaptability: how easily can the users adapt the software to their native locales?
- Language Neutrality: this quality reflects the externalization of all language and culture related elements in the source code such that the core functionality is independent of the locales. When the domain and locale elements are tightly coupled, modification of either becomes increasing difficult.

Requirements engineering challenges caused by internationalization are the combination of the same challenges internationalization caused to requirements elicitation, requirements prioritization, and requirements specification. Internationalization causes additional challenges to elicitation by increasing complexity in the stakeholder network of a product and worsening existing elicitation issues. As the number of stakeholders increases, so does the need for languages; different languages will need to be used to elicit requirements from users in different countries. During requirements analysis and prioritization, software internationalization complicates matters by increasing the complexity when determining the value of customer requests and this poses a new challenge to requirements prioritization. For example, a requirement highly prioritized by users in one locale may not even exist in another locale. The difficulty of requirements specification is also increased in multilingual project due to the possible need for documenting and communicating requirements in multiple languages. For example, different user groups will need

documentations in different languages and marketing of system features will need to be communicated to partners in different countries, so they work on sales readiness in their markets (Abufardeh S. O., 2009).

Solutions to these requirements engineering challenges different from project to project and are influenced by project attributes such as project's size, its commercial nature and its criticality. Nevertheless, examples of some solutions include (Jantunen, Smolander, & Gause, 2007):

- Central management of some of the requirement, such as core system functions and quality requirements like security and scalability, which do not change from locale to locale.
- Decentralization of some of the product requirements and features to geographic areas. This is possible if requirement engineers, either from regional branches or local business partners, are available at client side that remotely coordinates with the central project team. In some cases, it might even be possible to decentralize everything from requirements engineering to implementation and maintenance, but this can lead to inefficiency and a complicated code base.
- Architecture and design to ease software customization and localization. In other words, the software product is architected with internationalization in mind.
- Team re-organization to support decentralized control. This point refers to the team structure, which for multilingual projects may need to be geographically dispersed and multicultural.
- Encourage the practice of maintaining a top 10 or 20 lists of requirements per locale, which can be compared across locales and then the most needed requirements are satisfied before the other requirement.

In addition to the functional and non-functional requirements, multilingual software applications have special requirements relating to language and culture. Language affects functional requirements and culture affects non-functional (quality) requirements. Internationalization requirements are culture and language independent and localization requirements are culture and language dependent (Abufardeh & Magel, 2009). Categorization of requirements, during prioritization and triage activities, allows developers to determine the complexity and then estimate the time, cost, effort and resources needed to implement the requirement. Modified requirements categorization needs to be used (presented as a guideline in Chapter 5) for multilingual software that takes into account the locale related and cultural needs. The same is true during requirements prioritization.

In commercial projects (with a customer contracting a software developer to implement the software application), the level of internationalization and localization differs based on the customer's requirements. A product developed in US, in light of its level of localization, could be: U.S. English, English handling European data, English handing Far-Eastern data, English handling bidirectional data, partial or full translation of English user interface, or full localization with local market features (Abufardeh S. O., 2009). If the software is developed in a country where English isn't the primary language, the main user interface would be in the native language, but the data handled by the software application can still be categorized as it would be in a project where English is the primary language. This separation between user interface and

data handling is important in commercial projects because each affects the complexity and the budget of the software application differently. Although it is always better to internationalize software such that it can be localized for all locales, commercial contracts are subject to specific time plans and budgets, therefore during requirements engineering, it is important to understand exactly what needs to be multilingual and in what locales. The sooner this information is available, the better one may estimate the project cost and timeframe. Thus if possible, this information should be acquired before finalizing the financial contract.

4.4.3 Architecture and Design

This phase represents the start of software implementation and technology, architecture and design decisions made in this phase often determine the strengths, features and weaknesses of the final software product.

In contemporary projects, where agile methodologies or processes like Scrum, XP, Crystal or Kanban are used, this phase and its activities are not as explicitly identified as in traditional or more rigorous methodologies like Waterfall or RUP. Nevertheless, architecture and design are of utmost importance, especially for non-trivial software applications. Architecture and design are even more important when developing multilingual software projects as the project cost is relatively higher, the user base is more diverse and the project is technically more complex (Cockburn, 2006). Regardless of the software development methodology, coding generally follows a minimum level of design. In less formal methodologies, this may result in short discussions among the developers and rough designs on the whiteboard. In case of multilingual software, once the design of a component is finalized, the coder will need to internationalize and localize the software during coding. In agile methods, the testing may also be done at this time. However, this is not always the case and often only internationalization is done.

Software architecture and design have numerous definitions in academia and industry, each often different in terms of the strictness of the documentation notation and purpose. They reflect the needs of the customer and the end users, and these activities can be carried out only after the functional and quality requirements have been thoroughly understood. The quality requirements such as availability, modifiability, security and usability are often achieved through deliberate architecting and designing using well known tactics, or architecture patterns and design patterns (Bass, Clements, & Kazman, 2003). Similarly, in order to architect and design multilingual software, it is important to utilize existing and proven tactics that lead to software applications that are internationalized, and as a result, these products can be efficiently localized into numerous languages. In this section of the literature review, the focus is on available architecture and design tactics with relevance to multilingual software development. This includes a wide range of topics from architecture reference model for multilingual software to user interface design. However, the focus is not on architecture and design practices and documentation notations, which as mentioned earlier, differ from project to project reflecting the implementation methodology and the nature of the software.

Abufardeh and Magel support the idea that to truly accommodate multilingual software that takes into consideration numerous cultural issues, a large number of interrelated parts (classes and layers, for example)

of the system must be modified. Thus, they argue, development of internationalized software can greatly benefit from Aspect Oriented Programming (AOP) as it leads to isolation of tangled concerns and unification of scattered concerns. Tangled concerns refer to requirements that are satisfied using classes of code that are tangled with other classes that support other requirements. Scattered concerns refer to requirements that are satisfied through multiple, or scattered, classes. AOP enables developers to remove scattered and tangled code during design and coding by identifying and separating of crosscutting behavior into independent modules or components. One common strategy of software internationalization that is used by developers, software frameworks and Software Development Kits (SDK) such as Android is the separation of code from text resources (such as user interface, help documents, dialog box, currency formats and prompt messages) (Abufardeh & Magel, 2009). The main assumption based on which the design of internationalized software takes place is that all of the culturally and linguistically sensitive components (such as the user interface) can be separated from the locale independent core of the application (such as algorithms and business logic) (Callahan, 2005).

When detailed design starts in the project lifecycle, which may take place during refactoring in projects using agile implementation methodologies (Cockburn, 2006), software modules can be categorized as one of the following five types:

1. Localized UI (controls): this refers to the user interface related objects such as buttons and prompts.
2. Core product: this includes the software application's main functionality. For example, in an accounting system, this would refer to functions that implement mathematical formulas.
3. Culture-specific logic: this refers to the code that keeps track of the differences between the different localized versions and changes the state of the software application accordingly.
4. Third-party software: non-trivial software applications often need to be integrated with third-party software applications and services. Such integration related code should be separated.
5. Data processing: this includes all database related functions.

During the design phase, it is crucial for the development of internationalized software applications that designers identify and isolate locale-specific codes and modules from the rest (Abufardeh S. O., 2009).

Software architects are, among other things, responsible for technology selection. In terms of multilingual software development, selection of a development technology that supports multilingual software implementation is highly important. Using web technologies or mature development environments such as Java and .Net guarantees extensive support for internationalization and localization, but more specifically, when specific Integrated Development Environments (IDE) or implementation frameworks are used, the following questions should be asked: Does a text editor exist that supports both the desired target languages and the programming language syntax? Can the given compiler compile multilingual code? Does the emulator or executor run the compiled software application in multiple languages? (Venkatasamy, 2009) Often the answers are positive to all these questions, yet technical challenges arise. Therefore, it should be checked if the presentation layer, or UI components, provided by the implementation technology supports

both multiple languages and the associated properties, like text directionality (left to right and right to left). Additionally, it is necessary check, especially for web-based projects, whether the backend servers, the environment and the database support the desired target languages. A comprehensive multilingual environment for software development must be setup and ensured by the software architect, which requires collaboration and coordination with the project manager and business managers.

Software development approaches have evolved in terms of architecture over the years, and each new approach has affected other aspects of software development, including multilingual software development, from the modeling notation (UML for objected oriented analysis and design, for example) to the programming language syntax (PHP5 support for object oriented programming, for example).

These approaches are classified as either Programming or Architecture. Programming approaches have evolved through time and the industry has witnessed numerous programming paradigms. Multilingual software development, in particular, can be associated with the following (Venkatasamy, 2009) (Heuer, 2004) :

- Monolithic: this approach represents single-tiered software applications where the code is not divided into modules or components (also criticized as spaghetti code), and the code handling all aspects of the application are intertwined. As a consequence, changes to one part of the software may affect all other parts. In terms of multilingualism, this, as an example, could mean hard-coding culture specific code and text with core functionalities and data access code.
- Structured (or Modular): this programming approach emerged to infuse organization and structure in the code base by categorizing code into logical components or modules, which resulted in improved quality and development time. These structures represent the architecture and design, and include chunks of code, each representing one aspect; i.e. the data, logic and presentation layers. This multitier approach allowed the development of exceedingly large and complex systems. In terms of multilingual software development, most software products have similar requirements. Thus, the multilingual functionalities of the software application are developed as functional libraries, which can be then used in different projects (this was advocated by the Multilingualization Group M17N, which is no longer available). More generally though, the basic idea is that the culture and language sensitive elements of the applications, both code and data are separated from rest of the application. More specifically, the externalization of translatable strings would allow translators to work without the need to access the code and this approach would facilitate changes, independent testing, reduced build times and concurrent development. All of these positive characteristics are associated with the widely accepted paradigm of developing loosely coupled components with minimum dependencies (Gorton, 2011).
- Object Oriented: all modern programming languages support this approach and in simple terms, this paradigm uses objects that reflect real life items to implement large software system. An object is a chunk of software containing related data and behavior (Wampler, 2002). Using this approach,

multilingual software applications are implemented using Classes, blueprints of Objects, and changes are limited to the immediate object. Limitations of this approach emerge when a large number of objects are interdependent, which may create undesired dependencies. Language related elements are incorporated in classes that include all multilingual functionalities that adhere to different locales, like: representation of numeric data like date and currency, sorting of strings, rendering data on user interfaces of different device using the correct layout and fonts. Together these classes form a class library that can be used to internationalize and localize different software projects (Schmitt, 2000).

- Component-based: the term component when used in the general software context refers to a chunk or module of software, but in the context of component-based software development, a component represents a more complex structure with specific interfaces. A component is an independent group of software code that together forms a logical unit that fulfills some complex functionality. Component may include of numerous structures or classes, with respect to the previous two points, however, they also include specific interfaces that govern the utilization of the component. It is argued that component-based multilingual software development offer more benefits compared to the object oriented paradigm. This is because components exhibit characteristics such as encapsulation and polymorphism, but minimize the tight-coupling resulting from excessive inheritance among classes through invocations known as delegation. The user interface, for example, can constitute of UI components such as multilingual text, multilingual calendar, multilingual registration form, and so on (Mowbray & Malveau, 2003).

The second approach for multilingual software development is Architectural, which provides a blueprint for the software to be developed and enables the team to assess the quality and correctness of the software meet requirements, and ensures the software is developed systematically. Software architecture is defined by IEEE as "the fundamental organizations of system, embodied in its components, their relationships to each other and the environment and the principles governing its design and evolution" (IEEE Standards Association, 2000). The focus of this definition, as other commonly accepted definitions, is on abstraction of software into modules, and on the relations among these modules. In other words, the detailed functions and data structures in each module are not important at this point. Architecture also provides system wide guidelines that control how the system is implemented and how it may evolve. A good architecture should facilitate the implementation of the functional requirements and should ensure the end product meets highly prioritized quality requirements (Gorton, 2011) (Mowbray & Malveau, 2003). Literature on software architecture do not explicitly address internationalization and localization and the focus is often on domains, like enterprise, e-commerce, automotive and so on (Bass, Clements, & Kazman, 2003) (Gorton, 2011) (IEEE Standards Association, 2000) (Mowbray & Malveau, 2003).

Over the years, software engineers have developed software using different architectures with varying success. The architectures of successful software are obviously repeatable to build similar software and

meet similar quality requirements. The abstraction of specific kinds of architectures of successful software products proven in specific domains, in order to build software for similar domains, is referred to as the Architecture Reference Model. It provides a template for software architects when designing new systems, possibly using different technologies, in the domain and context that the reference model was abstracted for. An analogy of this in computer programming is the use of design patterns, where programmers use proven programming tactics to solve well known problems, regardless of the programming language. 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. Examples of references models include: Real-time Control System (RCS) for real-time intelligent control systems, Service Oriented Architecture Reference Model to build distributed systems based on the service oriented paradigm (OASIS, 2006). Reference models allow architects to identify and understand the components that must be included in order to meet the requirements, and for this and the other purposes of a reference model, (Venkatasamy, 2009) suggested an Architecture Reference Model for Multilingual Software (ARMMS) in his PhD thesis that can be utilized by architects to build multilingual software. ARMMS proposes a new model, named Aspect-based Language Library, for multilingual software development that can be utilized by architects and designers during implementation and can be clearly understood by developers. This model aims to address multilingual software and was devised after thoroughly studying the inadequacies in existing models using the design space approach. The new model achieves the expected qualities of multilingual software. This has the potential, like other modeling languages, to be used in model-based development and automated code generation. Additionally, in the same thesis, the author develops a reference model for multilingual software and implements sample projects using the proposed Architecture Reference Model for Multilingual Software (ARMMS). This can be utilized by architects to design multilingual software.

Although typically architects are not concerned with the user interface, database design and other detailed implementations, they are design issues important to various members of the implementations team. More often than not, the designs of these components of the software take place with some level of formality before the developers implement them.

UI is the most prominent, or visible by users, aspect of multilingual software applications. Designers should be sensitive to different users' groups' cultures and languages. Creation of prototypes is an effective tool to demonstrate the application early and gauge the users' responses. UI components should be flexible and easily modifiable to match different user groups, a good example of an implementation platform is how .Net UI components are easily configurable. UI design for multilingual software should start with identification of the parts and controls that need to be localizable. Developers should also pay special consideration to text contraction and text expansion issues, which refers to the change in the size of the text between different languages. This issue is very important today where an application may need to properly run on numerous devices with different screen sizes, such as laptops, notebooks, tablet computers, smart phones and standard

computer monitors (Davis, Tierney, & Chang, 2005) (Abufardeh S. O., 2009). Often, the language requirements would include support for bidirectional (shortened to Bidi or BiDi) languages. The key attribute is that the written text is often right to left (RTL) as opposed to left to right (LTR) as most world languages. From a GUI design perspective, problems may occur when the software is expected to support both Bidi languages (with RTL text orientation) and LTR languages such as English. In such cases, in addition to the expected culture and language sensitive translation, the orientation of the user interface layout as well as all UI controls (such as calendars, text input boxes, menus and scrollbars) would need to be mirrored or switched from and to LTR and RTL. The possibility and ease of doing this is often development technology dependent (for example, many Adobe Flex UI components did not support RTL orientation during its early stages in 2010). Although many development platforms have UI libraries that support multilingual text rendering, Davis et al. (Davis, Tierney, & Chang, 2005) proposed Web service based Adaptable User Interface (WAUI). As per their paper, the major contributors of WAUI include: separation of the UI from rest of the layers of the software application using web services and adaption of the user interface based on user's selected culture. The authors of WAUI identify limitations in the growing practice of XML-based UI implementations and claim that WAUI provides features like:

- Standard UI Language: as WAUI is based on XML, interaction with other logical application layers is simplified and this leads to a minimized dependency for the user interface.
- Common communications between application layers: the UI is standalone and dependent on XML based web services, which has programming level support by all implementation technologies.
- Internationalization: WAUI adheres to Unicode and this means support of all known language scripts. The logic layer should provision for multiple language sets for the text attributes of the UI components. Likewise, this allows for automated translation where a given language is not available. Paid and unpaid online service can be utilized for automate translation.
- Look and feel: WAUI allows users to set the look and feel of the applications.
- Customization: WAUI allows configuration of UI objects.

For sophisticated user interfaces, especially common in commercial websites, the use of graphics can lead to serious issues that inhibit the development of multilingual user interfaces. Among others guidelines for multilingual software development, how to correctly design and implement interfaces that heavily rely on graphics (image files with extensions such as gif, jpg and png) is discussed in Chapter 5.

Special attention should be given to the database when the software is expected to store data in multiple languages. Database should be designed to accommodate localized text and depending on the requirements, it should support data in one or more languages. This means, if the application supports user entered data in multiple languages, the database designer must consider character encoding for the different columns. Also, data may need to be stored in different languages in the database; this means the table structure must accommodate the different language versions (Abufardeh & Magel, 2009).

4.4.4 Software programming

Programming practices are often dependent on the implementation technology, which also drive the advantages and limitations (we discuss some technology and vendor dependent literature on multilingual software development in section 4.5). Software programmers developing multilingual applications must have a general understanding of additional issues relevant to software internationalization and localization. This includes properties of world languages that affect the software, required features of different tools and technologies being used, and programming techniques.

World Languages are either Pictographic, such as Chinese and Japanese, or Orthographic, such as English, Swedish and Arabic. Pictographic languages comprise of symbols that have meanings and when combined form sentences. Orthographic languages comprise of alphabets that when combined form words with meanings. Languages can be right-to-left (RTL) or left-to-right (LTR). Arabic, Urdu and Farsi are examples of RTL languages, while European languages are LTR (Abufardeh S. O., 2009). This is significant as it directly affects the orientation of the user interface. To understand this, consider a web application that must support English and Arabic (very common in the business world in the Middle East). In such projects, the RTL versions of the application are oriented right to left, and the LTR versions are oriented left to right. This means if the navigation menu and the logo are on the left side, when the language is switched, the layout should be mirrored to show the logo and the navigation menu on the right side. When the user switches languages, not only the content needs to be translated and localized, but even the layout of the application will need to be mirrored. This needs serious consideration during application design and coding. Things are worse for bidirectional languages, where the text is written RTL and the numbers are written LTR, and this can have implications on user input in particular. Additionally, the developer must take into consideration if the target computing environment supports cursive languages like Arabic, Urdu and Farsi, where the alphabets are connected to form words. Standard releases of Android OS from Google did not support this till as late as the end of 2011.

In monolingual software development, the selection of development technology and programming language would depend on the usual factors such as developers' experience, available libraries and supported platforms. However, the development technology must also be evaluated in light of its support of different world languages when developing multilingual software. For example, Adobe Flex user interface components did not support right-to-left languages in 2009-2010, and developers faced many constraints when designing multilingual user interfaces. Additionally, availability of internationalization and localization tools facilitates the process, especially when an existing monolingual software application needs to be internationalized. In existing products that need to be internationalized, developers need to identify all culture-sensitive elements in the source code to internationalize it. This can be a time-consuming and difficult activity, especially for larger software products, and specialized tools and approaches should be considered to automate the process. Not all hard-coded strings in an existing product that was not internationalized need to be externalized, and in order to automate locating the "need to translate"

hardcoded strings in the source code, one approach was tested and proposed by Wang et al. (Wang, Zhang, Xie, Mei, & Sun, 2009).

In order to internationalize a software component, the developer will need to externalize the text strings from the code, which means separation of any culture and language specific resource from the source code and storing it in external resource files. Development environments such as Android readily support this technique as the SDK inherently encourages storage of UI text in separate XML files, which are then called from the source code. How much a product needs to be internationalized is dictated by the localization requirements. The general rule of thumb is that the higher the localization requirements, the higher the internationalization needed (Shneiderman, 1992). Put another way, better internationalization eases the eventual localization activities, future updates, and minimize maintenance effort. Abufardeh (Abufardeh S. O., 2009) compiled a list of culturally dependent elements of software and stated that these elements must be externalized (in order to internationalize), this includes UI elements such as: messages and alerts, labels, help text, colors, icons and symbols, sounds, date and time, numbers, currencies, measurements, phone number formats, address formats, and page layout. Also, it includes functionality such as: searching, sorting, indexing, grammar and spelling. Today, this list will include additional UI elements optimized for mobile devices such as smart phones and tablet computers, provided by SDKs and mobile UI frameworks. It is important the coder understands the purpose of UI element and externalize it as necessary. During localization, the UI elements will need to be translated (Abufardeh S. O., 2009).

Main issue in data repository (database or resource file) localization is clearly defining what fields and tables require localization. For each target locale, the following issues (Abufardeh S. O., 2009) must be addressed:

- What text encoding will be supported?
- Will the target language filenames be supported?
- What format needs to be supported within the database for calendar, data/time and numbers?
- Are the data repositories shareable across language versions of the software application?
- What is default language for the data repositories (pre-populated databases)?
- In what languages will user entered database and file text be? How is language determined?
- What is the planned upgrade path for data repositories for future new languages?

Developers also must also take into consideration the underlying infrastructure and its support for different languages, including the users' operating systems. For example, many programming languages and technologies have UI components such as calendar and clock that adapt to the underlying settings of the operating system. In other words, if the user's Windows Sever is set to run in Swedish, the calendar developed in VB.Net would automatically be culturally sensitive. Developers must understand this and test the software on different operating systems. In distributed applications, where communication occurs over a network, it is important to ensure that the character encoding scheme is supported. For example, when

Arabic text is being transmitted between the client and the server, the network must support UTF-8 or Unicode (Abufardeh S. O., 2009).

Additionally, a developer may need to solve unique problems due to the multilingual nature of the system. For example, there might be a functional requirement to search the system for data in multilingual languages, but using a single query. This function is also known as Multilingual Information Retrieval (MLIR) and commonly used when searching using search engines such as Google. One method to tackle the MLIR issue that showed promising results during study is based on the Growing Hierarchical Self-Organizing Map (GHSOM) (Yang & Lee, 2008).

4.4.5 Testing

Quality assurance and testing activities are fundamental in non-trivial software projects and are carried out in different project phases. As opposed to monolingual software with English user interfaces, multilingual software require a different testing approach and method, which take into consideration the target cultures, languages, and conventions used for date formats, numbers, currencies and so on. Bidirectional software testing activities require additional consideration (Abufardeh & Magel, 2009).

With relevance to multilingual software development, the team must carry out two kinds of testing, namely internationalization testing and localization testing (Abufardeh S. O., 2009).

Internationalization testing is the verification process to ensure that a given software application, regardless of the primary language (mostly English), works as expected when localized in other languages. (Vine, 2004). Internationalization testing includes the following tests:

- Primary product testing: this is used to verify that the product source in its primary language (without localization) works properly in different target environments. Without this, it will be difficult to identify the root cause of the bug; whether introduced pre or post localization. This can be done using code review and testing working software.
- Pseudo-localization testing: this is performed to verify that the internationalized software can in fact be localized successfully. This kind of testing includes two activities: Pseudo-translation testing and Pseudo-mirroring testing. The former is done by replacing localization strings with temporary strings that needed not be accurate and complete at this stage, and the latter kind of testing is only needed for bidirectional software to verify that UI layout elements switch properly from and to LTR and RTL and vice versa. Pseudo-mirroring testing is only relevant if the software needs to be localized in bidirectional as well as unidirectional languages.

Localization of software is an expensive activity and the cost is increased as the number of languages increase. Localization testing is carried out to ensure that software functionality is not changed and no bugs were introduced after localization. The following (Renu, 2004) (Abufardeh S. O., 2009) tests should be carried out during and after localization:

- Translation testing: done to test if contents are translated thoroughly.
- Linguistic testing: done to verify the localized version of the application conforms to norms and expectations (and user requirements) in the target market.
- Usability testing: this type of testing needs to be carried out with end users because the goal is not to find technical defects but to gauge their satisfaction with the localization.
- Cosmetic testing: this is done to check if the introduction of a new locale has introduced visual defects. An example would be the text in the navigation menu being truncated once items are translated from English to Arabic, for example, as Arabic text may need wider space.

In order to effectively test multilingual software, preparation of a suitable test environment is needed for each of the supported languages, including the necessary keyboards, fonts and operating system. Similarly, all test cases and test suites (including unit tests and GUI test cases) should be internationalized and localized, just as the software application itself, in order to ensure quality (Vine, 2004). However, if software criticality and budget may dictate otherwise, it is often important to identify key functions and areas and prioritize the multilingual software testing activities. In addition to any traditional tools used during testing, the team should consider approaches and tools for multilingual software testing. For example, Guo et al. (Guo, Tay, Sun, & Urra, 2008) propose a novel tool for testing different localized versions of multilingual software. Using this tool, testers can test software that isn't in their native language. The proposed tool comprises of a knowledge base, test case generator and a test verifier. The test case generation accepts tests cases in one language and generates tests case in the desired target language using the knowledge base. The resulting test cases are then verified against the expected outcome by the test verifier. Such tools can cut costs by minimizing the need for foreign testing teams.

The translation and translation testing, if possible and feasible, should be integrated into the development process. Abufardeh (Abufardeh S. O., 2009) recommends a work setting where the extracted resources (software elements that need to get localized) along with help files and technical instructions are sent by the development team to the translation team (or person). The translation team, once done, sends the translated material to the localization team (or person, and could be the same as the development team). Once localized, the application is tested by the quality assurance (QA) team or customer (if customer, it is recommended this be done in a testing sessions and in a testing environment) and their feedback is sent back to the localization team, who may then ask the translator or developers to carry out the necessary changes as per the feedback from customer and QA team.

4.4.6 Post-release or maintenance

This phase refers to the period after the project is considered complete and is deployed for usage; the contractual liabilities are delivered or when the software is suitable for use. If a sequential process (like Waterfall) was used, this phase may be called maintenance or post-deployment. In iterative and agile methodologies and processes (like Scrum), this would be the period once all items in the product backlog

are complete and the project team is disbanded. In agile projects with particular long iterations, this may be considered the period after the release of user stories (functionalities) for use by end users. Depending on the nature of the software project and the team's strategy, the process of making a software application available in new languages will differ. For example, if the number of target languages is high, the team may decide to translate the software in stages. In this case, new languages may be introduced a long time after the project completion. In such cases, it is strongly recommended to architect and implement the software (internationalize) so that non-technical translators may easily translate content in new languages and, perhaps after managerial approval, the new language version of the software is automatically deployed and released in the production environment. In this regards, it is also important to establish administrative processes to approve translations, report bugs and change requests so that the localization occurs smoothly even after personnel change and the project team is disbanded (Cockburn, 2006) (Abufardeh S. O., 2009).

However, often there is a need to modify existing software applications to support new user groups belonging to different locales. If the software was internationalized during implementation, then there is only a need to localize the product for the new locale. This could even be achieved without the need to modify the source code or use the services of technical users, provided the software was properly internationalized. If the software is not internationalized, then the project team must analyze the existing software and decide on the most suitable and feasible solution, which might even be the complete redevelopment of the software from scratch.

The results of a feasibility study for a software project to internationalize and localize an existing software product that is not internationalized (please refer to section 4.4.1 for literature on multilingual software development with relevance to the feasibility study phase), one of the following may be decided on depending on numerous project factors:

1. Not to carry out the project: if the project team or source code are inaccessible and the project is forecasted to be over the budget and the costs are not justified.
2. To localize the software without internationalizing it: this may be the best option if the project budget is small and if the software needs to be localized to one additional locale.
3. To internationalize the project: although the most ambitious and costly choice, the long term benefits are numerous. In this case, the software is internationalized (architecture and source code are modified), so it is relatively easier to localize it in one or more languages.

Internationalizing existing software is challenging and complex. Numerous tools and techniques are available for consideration and it is highly recommended to use one or more of them.

One method to internationalize and localize existing software systems is to convert the source code to support Unicode, which can be achieved automatically using lexical analysis, as shown by Peng et al. (Peng, Yang, & Zhu, 2009). In their paper, the authors used lexical analysis to convert a system with 10 million lines of C/C++ code. The proposed technique scans the source code twice: first manually converting contextual code (cannot be automatically converted) after detection and analysis, then auto-converting rest

of the source code. This technique is especially applicable for C/C++ based systems and they converted an English system into Chinese using lexical analysis successfully.

Similarly, Daohe and Shusheng (Daohe & Shusheng, 2010) convert existing single-language software executables into another language by modifying the file structure and adding resource files using Assembly Language Kit Masm32. This technique can be used to modify Windows executables (exe files) generated by VC++, Delphi, VB and C++, but it does not work on encrypted executables. A technique such as this is very useful for legacy systems. Although it does not internationalize the system or offer long term benefits associated with internationalization, it offers a viable solution for legacy systems and numerous products, especially when the source code is not available, the project team is disbanded and the project budget is low.

One key activity that needs to be carried out when internationalizing an existing monolingual software system is the externalization of locale-dependent strings in the source code, which might have been hard-coded (intertwined with the source code). Locale, language or culture dependent strings include user prompts, navigation text, labels, error message and almost everything that the user interacts with through the user interface (UI or GUI), i.e. the presentation layer. Non-trivial software systems are built using thousands or millions of lines of code and identifying locale-dependent strings can prove to be an exhaustive and costly activity. To automate identification of "need to translate" strings, Wang et al. proposed a unique approach that they successfully applied on four real world open source applications (Wang, Zhang, Xie, Mei, & Sun, 2009). While numerous tools, like GNU gettext for PHP and Java Internationalization API for Java (both discussed in section 4.4 next), help developers externalize locale-dependent strings, there is a need to first identify suitable strings, which can be difficult in existing projects. Other tools help manage externalized resource files, which is not applicable for existing projects that are not internationalized. Eclipse IDE, for example, helps locate constant strings (UI text, not source code), but not all constant strings are locale dependent and need to be translated. The technique proposed by Wang et al. is based on string-taint analysis (tracing output strings to their sources) and offer methods to coop with possible complications.

Web-based applications are common and considering the number of monolingual websites (compared to multilingual websites), the importance of internationalization and localization is appreciated further. Considering the economical globalization, the spread of the internet around the world, the emerges of multi-cultural cities, and the large number of non-English and non-native (regardless of country) users online, it is important that websites around the world are also internationalized and localized. To address this issue, Ricca et al. proposed restructuring multilingual websites by extending XHTML so that multilingual websites are maintainable and consistent. The proposed solution, Multilingual XHTML (MLHTML), never gained industry support and no contemporary browser supports the proposed extension of XHTML (Ricca, Pianta, & Girardi, 2002). XHTML today is being replaced with HTML5, which provide numerous improvements, but still do not address multilingualism inherently as MLHTML did. In their paper, Ricca et al. also recommend algorithms to migrate existing monolingual websites to MLHTML to support multiple languages and a number of techniques are exploit to automate the process. In our view, in spite of its

elementary state (tested on only few static websites and RTL languages), MLHTML like approaches should be developed and adopted by mainstream developers to facilitate the development of multilingual websites and web-based software applications. This can lead to the availability of resources on the internet to all the people around the world, regardless of their locales.

In many cases, commercial tools and services (from Localization vendors discussed in section 4.5.3) can be utilized to internationalize and localize existing software. For example, Globalyzer, from lingoport.com, is a software tool that internationalizes existing software source code.

4.5 Technology and vendor dependent software internationalization

In addition to the literature that were reviewed in an integrative manner in the preceding sections, it is important to understand how multilingual software development is addressed by different solutions from vendors or when using certain implementation technologies. For example, if a given system is being implemented using Java or .Net, it is important to consider the recommendations, tools and libraries provided by Oracle Sun and Microsoft respectively. Similarly, if the project team decides to utilize a localization tool or the services of a localization vendor, it is important to understand the features and competencies of the tools and service providers (Zetzsche, 2005) (Abufardeh S. O., 2009).

Obviously, it is not possible to review literature on all technologies and solutions, and literature isn't available on all anyhow, in which case, online tutorials and reports were reviewed. (In Chapter 5, we present a table containing different technologies and related solutions for multilingual software development. Similarly, another table presents localization tools and localization service providers. For these, the guidelines for multilingual software development are limited to references to suitable libraries, solutions, tools or service providers; with limited descriptions and comparisons. Multilingual software development in each technology can be exhaustively researched in future studies.) Nevertheless, the leading programming languages and localization tools are reviewed in light of multilingual software development.

4.5.1 Programming languages

Reports in 2011 listed the top programming languages in terms of usage in projects and demand for competent developers. In no particular order, the lists included: Java, .Net (C# and Visual Basic), C, C++, JavaScript (combined with HTML, CSS, XML and AJAX), Perl, PHP, Python, Ruby, Objective C and ActionScript (TIOBE Software, 2011) (Taft, 2010).

In Java based projects, localization can be achieved without engineering changes; without changing the source code. The localization team only needs to add locale-specific elements such as translated text, fonts and input methods. Java supports internationalization through classes and packages that provide language- or culture-dependent functionality for its core, desktop, enterprise and mobile platforms. Detailed documentation and tutorials are available on the Sun Developer Network (Oracle Sun Developer Network (SDN), 2010).

Microsoft .Net Framework, including Visual Basic (VB), C Sharp (C#) and other programming languages, facilitate software internationalization and localization through numerous features and utilities, including localizability using resource managers and customizability using custom cultures. In particular, the CultureInfo class from the System.Globalization namespace handles all culture-related issues needed by .Net developers, including language direction, number formatting, currency and sorting. Visual Studio, the Integrated Development Environment (IDE) used to build .Net software, takes into consideration all issues important for development of global software, including testing. Arguably, .Net and Visual Studio provide the most comprehensive support for multilingual software development. Microsoft Developer Network (MSDN) contains numerous tutorials and resources to support developers build multilingual software (Microsoft Developer Network (MSDN), 2011) (Smith-Ferrier, 2006).

C and C++ languages do not inherently support Internationalization and Localizations with the same detail as Java and .Net; there is basic syntax support to set the locale, format numeric values and set currency symbols. The developers must themselves use suitable approaches and tools, for example those discussed in this thesis. Nevertheless, there are a number of tutorials and checklists available online that can provide a starting point (Oracle Sun Developer Network (SDN), 2010) (Loosemore, Stallman, McGrath, & Oram, 1994), and the localization tools discussed in section 4.5.3 can also be utilized.

JavaScript is a client-side scripting language for the web. Majority of mobile and desktop web browsers support JavaScript. JavaScript works in combination with HTML, CSS and even XML. More specifically, JavaScript is used for client side (web browser) programming to manage the user interface elements structured in HTML. Cascading Style Sheets (CSS) is used to control the look and feel of the user interface and XML (or another standard) may be used for information transmission with the backend. Non-trivial web-based applications are implemented using server side technologies such as Java, ASP.NET or PHP, which provide means for internationalization and localization. Nevertheless, with the emergence of AJAX and Rich Internet Applications (RIA) that communicate with the backend asynchronously, the role of JavaScript in Internationalized is more complex. A common approach observed in many websites is duplication of the front end (or even the backend along with the database) for different language. In such cases, the web-based application is replicated for different language versions, simply modified the culture-dependent elements and text. In such cases, the JavaScript code, for example to display the current date, is written multiple times in each version, each with a different language and format as per the culture. This means, even the corresponding HTML is duplicated. This can be simplified using some of the guidelines presented in section 4.4.4, for example the fundamental approach of externalizing culture-specific elements into resource files and modifying the source code to retrieve these elements from the resource file depending on the user selected language. Localization tools can also be used to simply the process, including Gettext discussed in section 4.5.3 below (Somerville, 2007) (Lingoport.com, 2008). Additionally, it is important to specify the correct file properties to reflect the supported languages; the character encoding in HTML files,

for example. In HTML, the character encoding needs to be set to reflect the language being used. Additionally, for Bidi languages the layout may also need to be set to RTL (W3C, 2007).

Perl facilitates multilingual software development somewhat similarly to C and C++, as discussed above, by providing language support to select and manage locales through function calls. For example, the POSIX::setlocale() function specifies the locale at runtime and POSIX::localeconv() function helps format numeric values depending on the selected locale. Similarly, Perl uses the LC_COLLATE environment variable for collation (ordering) of characters in different languages (Perl.org, 2010). Additionally, localization tools such as Gettext (discussed in section 4.5.3 below) and Maketext (claimed to address shortcoming of Gettext) are used when developing multilingual software using Perl (Burke & Lachler, 1999).

PHP has numerous libraries and frameworks than enable developers to effectively and efficiently develop and maintain multilingual software. Gettext is used to internationalize PHP-based software; it helps extract culture-specific elements from the source code and manage them in resource files. Similarly, Enchant provides general purpose spell checking, FriBidi to handle bidirectional scripts, iconv is used for character set conversions, intl enables programmers to handle collation and numerical data formatting in different languages, and mbstring String is used to manipulate strings with Unicode characters (PHP, 2012).

Python relies on Gettext for Internationalization and Localization.

Ruby facilitated multilingual software development through the Ruby I18n framework, which "provides all necessary means for internationalization/localization of your Rails application" (Fuchs & Minarik, 2011). Ruby I18N "gem", shipped with Ruby on Rails, offers an easy to use API for to internationalize and localize Rails applications. The internationalization process is simplified for Rails' developers in three steps: ensure i18n support, specify locale dictionaries, and configure locale usages. To localize, the localization team, including developers and translators, will need to carry out three activities as well: modify Rails' default locale elements (like date and time formats), externalize strings used in the application (like prompt messages) into the dictionaries, and store the resulting dictionaries.

Objective C (and Cocoa Framework) is used to develop applications for Apple operating systems, including Mac OS and iOS. The Mac OS X Developer Library lists specific steps to internationalize and localize Objective C based applications, including modification of Nib files (windows, views, and menus; basically the user interface) and source code functions to load cultural strings from the external resource files (Apple Inc., 2010). As with other languages, localization tools can be utilized by developers to expedite the process. Ibtool can be used to localize the nib files into multiple languages, resulting in different versions of nib files for different locales. iLocalize is another very useful tool; it is an application designed to help developers localize their applications (MacUpdate.com, 2005). Other tools include: NibLocalizer and iLingual.

Android SDK extends Java, which is used to develop apps for the Android OS. In theory, numerous options available for traditional multilingual software development in Java can also be used for multilingual

Android app development. Nevertheless, Android SDK facilitates and encourages separation of UI string from the code, storing both the UI layout and application strings in externalized XML files. This can facilitate internationalization and subsequent localization of Android apps (Android Developers, 2011).

ActionScript is the scripting language used in Flash applications. Traditionally it was used in Flash animations, but with the emergence of the Flex and Adobe Integrated Runtime (AIR) frameworks, ActionScript is being used to develop highly user friendly user interfaces for business and other kinds of software systems. Flex Builder IDE, renamed to Flash Builder, is the development environment used to build Flash, Flex or AIR applications. Multilingual software development is facilitated by Flex Builder through modifying the project properties by setting required locales and by creating resource files to store the externalized strings (Adobe Developer Connection, 2011). Many UI components in Flex do not support Bidi languages and its fonts directly, but the Text Layout Framework, which is open source, or Arabicode, which is commercial, can be used to extend Flex Builder (Adobe Labs, 2011) (Arabicode, 2011).

4.5.2 Software development frameworks

In practice, software developers often utilize frameworks to implement non-trivial projects. Software frameworks enable developers to implement software more efficiently and productively by providing all the required low-level implementations. Frameworks are implemented using sound architecture and design patterns and provide a large number of libraries that facilitate coding. In addition to reuse (with possibility of extending the source code), frameworks enforce good coding practices and enable the development team to focus on implementing requirements rather than low-level details (Wikipedia Software Frameworks, 2011). A large number of frameworks are available, each implemented using different programming languages and with numerous advantages and disadvantages. Some frameworks support Internationalization and Localization, others do not, and customization will be needed, which obviously is not recommended (Baker, 2009). Examples of Java frameworks include Spring, Apache Struts and Java Server Faces (JSF). PHP frameworks include Drupal, Zend Framework and CodeIgniter. Similarly, numerous frameworks implemented using other technologies are available. Software development teams often specialize in one framework and when hiring new programmers, they prefer candidates that specialize in the same framework. When selecting a framework for a multilingual software project, it is very important to choose the one that extensively supports Internationalization (I18N) and Localization (I10N) (Wikipedia Web Applications Frameworks, 2011).

4.5.3 Localization tools and service providers

Localization tools available today can be categorized as those that require access to the application source code and those that localize the application without requiring access to the application source code. Most tools will support one or both of these techniques. In the former case, the resource files (containing elements that need to be localized) such as source code, text files and data repositories are edited before compilation. In the latter case, the compiled application files such as .exe or .dll in Windows OS are localized without the

need to access the source code (Esselink, 2000). Regardless of what tool is used, manual work is always necessary. The tools expedite the process and a good one should generally perform the following tasks on the resource files: translate strings from and to multiple languages, change font style and size, adjust UI controls on the screen, modify icons and shortcuts to match the target locale, and change text encoding (Zetzsche, 2005).

Additionally, whether a particular programming language or framework inherently supports multilingual software development or not, it is a good idea to consider the different tools and services available in the market. Numerous localization tools and service providers are available and more details can be found online, which should be thoroughly analyzed and compared by project leaders before a decision is made. Leading localization tools include (Zetzsche, 2005) (Abufardeh S. O., 2009):

- **Alchemy Catalyst** (Wikipedia Alchemy Catalyst, 2011) (Alchemy Software, 2011)**:** Alchemy Catalyst is commercial and is one of the leading software internationalization and localization tools available. The company that owns it is large with offices in a number of countries. Catalyst constitutes of a number of graphical tools and support different platforms and development technologies, including all leading database systems. Catalyst also includes a quality assurance tool that can be used to validate the results of the localization, including errors in layout and spelling errors. Catalyst, among other, supports all Microsoft development platforms, a number of mobile computer environment (based on our superficial research on the company website, none of the contemporary mobile platforms such as Android and iOS are supported), web technologies such as HTML, PHP, XML and so on, Java platforms such as J2EE, J2SE and J2ME, and data sources including Microsoft SQL Server and Oracle.
- **SDL Passolo** (Wikipedia SDL Passolo, 2011) (SDL , 2011)**:** Founded in Germany, SDL Passolo is a commercial tool that allows the developer to use the visual localization environment to select the best user interface translation, to edit the translation in runtime, to customize the tool to meet the needs of the user in an integrated development environment, and integrate the environment with other development environment. SDL Passolo supports platforms such as Microsoft .Net, Java, Oracle, MySQL, MS SQL Server, and web technologies such as HTML and XML. Also, like Catalyst, Passolo provides rich training, knowledge base and support.
- **Gettext** (Wikipedia Gettext, 2011) (GNU Gettext, 2010)**:** Unlike most other tools, this internationalization and localization tool is free and open source. GNU website states that "Gettext is an important step" to their translation effort and it "offers itself to programmers, translators and users through an integrated set of tools and documentation". The available documentation online explains how Gettext works with programming languages such as C, Objective C, Python, Smalltalk, Java, C#, Pascal, Perl, PHP and others. Also, it seems like that Gettext categorizes its components into three "views", namely, the programmer, translator and the user.

Many other tools exist, some work with only certain development platforms and others support a wide range of languages and features. A good commercial Microsoft platform tool is RC WinTrans (Schaudin, 2011). Multilizer (Multilizer, 2011) is another commercial tool that supports more than 30 file formats and provides graphical interfaces and creative methods for content translation. Other small and emerging tools include: ResX Localization Studio for .Net (ResX Localization Studio, 2011), and Lingobit Localizer (Lingobit, 2011) a commercial tool that works with a number of major technologies including Delphi and Java. It is often a good idea to search for localization service providers or tools that are developed with only a certain language in mind as the fewer the languages, the more specialized the support and this can include automated language translation of content, which can lead to huge cost cutting. Visual Localize is one such tool that is specialized in localizing software from English to German (Visual Localize, 2011).

Many localization vendors can also be utilized if the budget permits. Such companies can add tremendous value to large and complex projects through their experience with capabilities ranging from translation to software customization. Examples of localization service providers include: Rubic.com, ENLASO.com and Lingoport.com.

4.6 Miscellaneous literature

There are a number of quality attributed related to software that may be affected by the internationalized software, such as security and performance. Similarly, an understanding in other technologies might be needed, including Computer Aided Translation (CAT) and multilingual speech recognition and text to speech conversion.

Security and performance are two quality requirements that are highly affected when internationalizing an application, so developers should pay additional attention to these requirements if they are important for the project (Abufardeh & Magel, 2009). Internationalized software requires flexible input data forms as opposed to input and output being in a single language. SQL injection is a common technique used by hackers to exploit vulnerable data forms in applications and when an application is expected to handle a larger number of characters (from different languages), the potential for such security breaches increases. Similarly, it should be ensured that the encryption algorithms support the languages being used. Internationalization of software may also affect the performance of data-intensive applications due to the language-related flexibility that must be provided to support multiple target users. This requires special design and coding consideration. As an example, if we consider a web-based application where large content may be retrieved from the backend and rendered into the HTML Document Object Model (DOM) at runtime, this can cause serious performance and usability issues even with today's high-speed network connections and high performance client computers. This issue can be seriously aggravated if the application is multilingual and content may need to be retrieved in numerous languages at once.

Often in projects, it is necessary to integrate software components with third-party products and services. In such cases, it is important to identify the integration requirements placed by the third-party developers. In

particular, the third-party product or service will need to support the same language as the new product (Mowbray & Malveau, 2003) (Gorton, 2011) (Esselink, 2000).

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, 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 (Dillinger, 2010) (Google, 2011) (Malik, 2009).

5 Results

This chapter presents this research project's main results, namely the guidelines for multilingual software development and a means of retrieving the relevant guidelines. First the results of analysis of sample industry projects and related interviews with project members are presented. Second the guidelines for multilingual software development extracted from literature, technical papers and industry sample projects are listed categorically. Finally, a means of retrieving relevant guidelines from the compiled list is proposed, which is then explained using realistic examples.

5.1 Industry Sample Projects

A total of ten industry projects were interviewed. As an eleventh interviewee, one of the authors shared his experience as project manager in a monolingual distributed software project. More detail about the purpose of the interviews and the interviewees is provided in Chapter 3 titled Methodology. Here transcripts of the interviews are presented.

5.1.1 Multilingual Industry Projects

Three bilingual university web portals in Sweden were analyzed and project team members interviewed. Like most other universities in Sweden, the content and web-based services are provided in English and Swedish. These projects are specifically interesting because both versions are actively used and the volume of content that is expected to be provided in both the language versions is relatively high. The purpose of such university websites and web portals is to provide services to students, faculty members and others. It provides basic information about the university and publishes articles, research papers and reports. This indicates a large size of static content that can be challenging to maintain in multiple languages, English and Swedish in these cases. In other words, if a PhD thesis written in English is published, it's not straight forward to make this available in Swedish; hence only web links could be provided in the Swedish version to the thesis that is written in English web portal.

For the first university the web master working at the university as well as the project manager from the company that the university had outsourced the implementation and maintenance of the web portal to were interviewed. This project was implemented using a Java-based open source Content Management System (CMS). Since this was a customization project; the CMS software was already developed and available for use, no particular software implementation methodology was used. This highlights a mismatch between industry projects like these and some of the literature on multilingual software development that emphasize the integration of internationalization and localization practices in the software development lifecycle (Abufardeh S. O., 2009) (Young, 2001). The interviewees also confirmed the ongoing nature of the project; although the web portal was implemented in twelve months by a team of eight members, bilingual content is continuously updated and added and sometimes this requires technical support from the company. For the university providing an English version was essential for international students and non-native faculty members. The project manager from the company emphasized on the importance of using an

internationalized CMS as it is a main selling point in most business opportunities. The CMS currently does not inherently support BiDi languages but this can be achieved if requirements arise without much technical difficulty. However, in terms of supporting multiple cultures, the main technical challenge arises when there is a need to integrate with third-party software. As stated in numerous literature (Esselink, 2000) (Reinecke & Bernstein, 2007) (Peng, Yang, & Zhu, 2009), the main challenge in this project was identified as the creation and maintenance of content concurrently in Swedish and in English. The desired improvements included the ability to write content in two languages at the same time as it expedites and eases the translators' efforts, and the availability of in-built machine translation components. This strongly supports the notion that the main challenge in multilingual software development, especially in projects with large static content, is related to the translation of content. With time, this has led to difference in the two language versions mainly because the university found it infeasible to consistently ensure content is available in both languages. As a result, often a page needed for international students, is only partially translated in Swedish, and similarly the opposite is true. Since the CMS expects the same page in the other language version, this forced the web masters to write short texts in the other language. However, soon it was realized that this is not feasible, and the CMS was modified to handle page redirects; if a page isn't available in the other language, the user is automatically redirected to another suitable page. This is an example of the project team deciding to rather modify the software than to translate large contents. Architecturally speaking, the CMS used supports two mechanisms two implement bilingual web portals. The first way is to configure the CMS so that the two language versions have duplicated resources and in the second way the resources and reused and textual content is loaded dynamically at runtime. The latter way is easier to maintain and it is easier to add new languages, but because the pages structure and resources such as images are not duplicated, all language versions must have the same pages. This means each page must be translated in all other languages. Although the CMS is internationalized, both the interviewees stated that adding an additional language version would be a very difficult task in terms of time, budget and resources. In practice, the CMS administrator would have to update the configuration to specify the new language and then all of the existing pages would have to be translated into the new language.

Web masters of the second and third universities' web portals confirmed similar motivations, as the first university, to develop and maintain their respective websites and portals in Swedish and English concurrently; for international students and faculty members. Similarly, Content Management Systems (CMS) were used, but commercial ones, thus no particular implementation methodologies were followed and the focus of the team of five and seven members respectively was on adding content and agreeing on the visual theme. While both these universities implemented their systems using different technologies, the challenges they face are also mainly related to translation of the content. As a result, although the web portal is available in English and Swedish, the content and pages are not identical. Obviously, it is not suggested that this is a problem as it is natural to adapt the content two different user groups, but as it was the case with the first university, often the web masters and system administrators compromised because of difficulties associated with professionally translating large content.

In addition to the universities, a number of Swedish government organizations' websites were selected that often must communicate with non-Swedish speakers. Like the universities, government organizations used different content management systems to implement their websites with team with three to seven members. This, again, means that software project implementation methodologies are irrelevant as systems available commercially off the shelf are used, and the inherent limitations with relevance to support for multiple languages are inherited.

Another Swedish government organization, which provides services to businesspersons, maintains a bilingual website in English and Swedish. This is supplemented with pages in 14 additional languages that provide basic contact information to visitors, which include investors and businesspersons from around the world. Unlike most of the other government and educational software projects, this one was not based on commercially off the shelf CMS or portal systems. The technical webmaster, also the web developer during the development phase, informed that the main difficulties to implement a multilingual website were translation of content, which had to be outsourced to professional translators, and working with non Latin (non Western) scripts. The development team commenced implementation using ISO 8859-1 character encoding set, which they later realized was inappropriate for non Western languages such as Arabic, Farsi, Hebrew and Kurdish. The interviewee stated that in retrospect, a more suitable character set, such as UTF-8, would have been selected. Analysis of the website reveals additional shortcomings that the interviewee did not identify. For instance, although single pages in Bidi languages are available now, the overall website orientation and layout remains left to right for these languages, and this will result in technical challenges and require changes to the source code if the website was to be made available completely in a Bidi language in future. This project demonstrates the lack of awareness about multilingual software development and how a team had to learn the hard way mid project that they had taken wrong decisions with regards to character encoding.

The project manager of another government agency stated that less than an year ago, its website was re-launched, just two years after its first launch, and one of the reasons was to provide information more comprehensively in English for its visitors from other countries in Europe. This website was implemented using a CMS and the re-launch mostly included the design of a new template and site structure. In spite of this, the English version still does not include all the content available in Swedish. The interviewee also mentioned plans to introduce additional language versions, including Finnish, German, Spanish and Somali. This is another example of a local government organization confirming the need for multilingual software development, yet not having a clear plan to carry out the actual implementation in multiple languages. The main challenges identified were the need for money and time to create multilingual content and then sub sequentially the need for resources to maintain a multilingual website with diverse content.

The website of one Swedish public organization uses a very effective way to present its content in multiple languages without spending any money! This is done using Google Translate gadget, which lets the website's visitors select any of the many supported languages and then translates all the textual content

within a few seconds. The webmaster interviewed informed that a commercial CMS was used to implement the project and the translation feature is integrated into the system. Obviously, Google Translate has a number of drawbacks: the quality of translation is poor, the webmasters cannot control the translated content, the layout and orientation of the page does not change for Bidi languages, and webmasters cannot do much if a particular language is not supported by Google. Nevertheless, given the nature of this project (a government organization with mostly Swedish visitors and few international visitors) these drawbacks are often acceptable, especially considering the expensive alternatives. Automated machine translation gadgets, like Google Translate, can only translate non markup textual content in web pages and this means that images with embedded texts must be avoided at all costs, as it was done in this project.

The final government organization with a multilingual website that was interviewed appeared to have a clear understanding of multilingual software development. Two of the project team members who were interviewed explained that one of the requirements was to present all the basic information in 14 languages, with the rest of the website only in Swedish. When any of the languages other than Swedish are selected, limited basic information is provided, each language with varying content. The interviewees understood that the translated versions were incomplete and they justified this with the high costs of translation, as a result of which even the menus were not translated. For right to left languages, the developers did change the direction of the html pages, but the overall layout of the website remained left to right. This project included competent members who understood the shortcomings of their website in terms of multilingual features and had an idea about possible solutions, but were simply bogged down by the costs of translation, especially since this project included more than 14 languages. This emphasizes the need for effective machine translation services and the importance of efficient administrative processes (to cost effectively utilize translators available around the globe).

The developer of a software tool used to design textual effects, including multilingual output, for television broadcasts, explained the importance of considering the characteristics and requirements of different languages in order to correctly produce multilingual output. For example, it was mentioned that the source code had to be reviewed thoroughly to update the selected character encoding to Unicode, which supports scripts of all languages. This project is different in the sense that it is used by technicians to output multilingual content for broadcasters.

5.1.2 Monolingual Industry Projects

Among the monolingual sample projects analyzed were from two Swedish government organizations. The project manager of the first explained that the CMS used to implement the website was updated to the latest version that provides better support for multilingual content. Although the website was monolingual, available only in Swedish, and with limited content available in English, the interviewee still perceived the project to be a multilingual software project. It was explained that multilingual interfaces and contents were important requirements in this project as it's needed to effectively communicate with minorities and immigrants in Sweden, although only English and Swedish versions were supported for now. This suggests

a possible misunderstanding of what is expected of multilingual software. Nevertheless, this project shows that multilingual software development is not only important for commercial reasons (to enter new markets in different countries), but also a necessity in many multicultural and multilingual countries. The CMS used in this project, which is a leading CMS in Europe, did not handle right to left languages (bidirectional or Bidi). While the implementation team managed to handle non Western and Bide content, the pages orientation and layout was still a problem. This shows that even widely sold and successful commercial systems do not fully take into consideration multilingual software development and that procurers fail to evaluate competing products thoroughly. For example, in this case although the organization had plans to publish content in Bidi languages, a product that did not inherently support such languages was purchased. Additionally, although an important requirement, the website was not multilingual; available only in Swedish with a handful of pages available in English. It was explained that translation of contents in English was found to be a costly activity, so only "the important" information was made available in English, indicating an inaccurate feasibility study.

The IT coordinator of the second monolingual Swedish government website explained that only Swedish speakers were considered as there were no legal requirements to do otherwise. Nevertheless, the project team did include a handful of pages in English, including basic information about the organization and its contact details. The main reason not to localize the website, although supported to a large extent by the purchased CMS, was the cost of translation, which would have been continuous as content is always being updated. The interviewee insisted on the team's decisions as the possible non-Swedish speaking visitors do not justify the costs of translation. In such cases, alternative methods of translation, such as using free translation services or crowd sourcing, can prove to be effective.

One of the authors of this paper (Muhammad Murtaza) worked as project manager and leader developer in a monolingual distributed software project. This project was expected to be only in Arabic. The cost of translation was not an issue as the size of static content was limited, but as a bespoke project with a limited timeframe and strict requirements, the team simply did not have to worry about other languages and locales. Having said that, the team foresaw potential future implementations (to meet similar requirements by different clients) in different languages, and thus all the static texts were externalized in XML file. Additionally, where possible, the user interfaces were made direction neutral; i.e. the forms with labels and input boxes were laid out vertically, as opposed to horizontally (where the controls will need to be mirrored to support languages with the opposite directionality). Due to time and budgetary constraints, the internationalization features were never tested. Additionally, the database and the user interface were never tested with non Arabic data. This project was developed by a team with some experience in multilingual software development, yet the product was justifiably not thoroughly internationalized. In this case, this was due to the commercial constraints and simply because the clients did not have any multilingual requirements, neither in the short nor in the long terms.

5.1.3 General observations and reflections

Often decisions makers choose to implement their systems using existing commercially (or open source) off the shelf software. This is particularly evident with the use of CMS software in Sweden. With respect to multilingual software development, this practice can simply technical matters and only the extent of internationalization should be evaluated in competing products. It was observed that although multilingual interfaces was a key requirement, the ability of the competing products to meet this requirement was often either ignored or superficially discussed.

Translation of content into multiple languages and the sub sequential management of multilingual texts are considered a key challenge and obstacle for multilingual software development and maintenance, in particular when the project includes a large amount of static content (informational pages, scientific reports, publications and continuously updating news and segments). Most interviewees, in spite of numerous insufficiencies with relevance to internationalization and localization in their projects, considered the technical development of multilingual software (internationalization) as an easy task, especially when the project source code and developers were still accessible. Additionally, the costs associated with translation were perceived as high, often due to the uncertainty involved and because static content was to be continuously added and modified. Most interviewees seemed oblivious to cost-effective techniques of translation, including the use of freelance translators that can be contacted online.

When asked if the motivations to develop multilingual software were worth the effort, over 90% answered “yes”. When asked if they were to redo their projects, would they consider internationalization such that future localization efforts are minimized, about 50% answered “yes”, 30% answered “maybe”, and 20% (two respondents) replied “no” indicating they were satisfied with their solutions. When asked about the perceived technical difficulty level for introducing the software in a new language version, approximately 30% answered either “easy” or “somewhat easy”, 40% “neutral”, and the remaining 30% replied perceived it either “somewhat difficult” or “difficult”. This indicates that the interviewees perceive their products, most of which are commercially off the shelf CMS software, capable of easily being localized. In other words, 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.

5.2 Guidelines for Multilingual Software Development

Guidelines for multilingual software development, as extracted from literature and industry, are categorically presented here. The categories help logically organize, manage and access the large and diverse number of guidelines (these categories reflect the sections used in Chapter 4 Literature Review and some guidelines crosscut categories so the relevance to the category is used to decide where to add them). Each guideline in each table includes a unique ID (IG01, IG02, EF01, EF01 and so on), which are used in the Guidelines Navigation Tool presented in the next section. Guidelines for multilingual software development with relevance to programming languages, localization tools and vendors are not exclusively

included in the tables below. When need be, reference is made to section 4.5, which includes related information (section 4.5 is supplemented by the Bibliography at end of the document).

Categorically organized guidelines for multilingual software development are as follows:

Introductory Guidelines	
IG01	To internationalize a software product, three aspects must be taken into consideration. First, locale and culture, second UI and documentation, and third data storage and text processing. See 4.1 for details.
IG02	A software product's level of internationalization is measured by checking for: • Elimination of reference to culture, politics or history. • Non-existence of graphics with embedded texts. • Source code must be free of hardcoded locale-dependent strings See 4.1 for details.
IG03	To develop multilingual software, the project team must be familiar with tools and technologies relevant to the development technology being used. These tools can be used to expedite the internationalization and localization activities, including in already existing systems that might not have been localized. Check section 4.5 for examples. Additionally, it is important to understand numerous terms and standards like: ASCII, ISO8859, Unicode, UTF-8, Translation Memory (TM), TMX, and XLIFF. See 4.1 for details. 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, 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.
IG04	Programmers in the team must understand that multilingual software development is more than content translations. In addition to language, it includes consideration of culture, business practices and users' preferences (dialects and languages within a country, for example).

Table 5-1: Introductory Guidelines

Economical and Financial Guidelines	
EF01	It is important to know what benefits multilingual software development presents both in the short and the long terms, as well as understand the key cost components for a multilingual

	software project. This will help the team make better decisions and not lose focus from internationalization and localization related activities later on during development. Without a multilingual software product it is difficult, if not impossible, to complete in the global IT industry. This is why close to 80% of software developed in English are localized and sold internationally (Abufardeh S. O., 2009). Customers are four times more likely to purchase a software product if the content is in their native language (Welzer, Golob, Druzovec, & Kamisalic, 2005).
EF02	Software publishers must utilize tools, technologies and trends to develop multilingual software and take advantage from the associated financial benefits. With numerous tools, localization firms, and a mature industry, it is possible today for even small or medium sized businesses to grow by entering into new markets around the world with localized versions of their products. This can be achieved today more than ever before because the internet is available virtually in all countries of the world and the online users community has been constantly growing (Jantunen, Smolander, & Gause, 2007).
EF03	Usability and acceptance of the software product is increased if it is localized. However, software internationalization and localization is time consuming and expensive (Reinecke & Bernstein, 2007), so it is very important to estimate associated cost components carefully, especially those related to the translation of content into different languages. The more content there is to translate, the more costly and complex the efforts. Similarly, the more target languages to translate content to, costs and complexity increase significantly.
EF04	The cost associated with professionally translating the content can get out of control. This cost increases with every new language and may become an overhead cost if the application's content is continuously being generated or added (in case of software that manages scientific publications, for instance). While many software projects are based on customization of commercially off-the-shelf software (COTS) or open source applications, like Content Management Systems (CMS), which are already internationalized, the cost associated with localization, and translation of content in particular, still needs to be considered. See sections 4.2 and 5.1 for details.
EF05	Unless it is absolutely clear that a product will always be monolingual; the chances of the users' groups diversifying are very low, the costs associated with internationalization and localization should be considered. Even if a chance of localization is low, it pays to internationalize the product during development as it permits localization and reduces future costs significantly. If there is no chance of localization at all, then complexities and costs associated with multilingual software development can be ignored. See section 5.1 for details.

Table 5-2: Economical and Financial Guidelines

Administrative and Managerial Guidelines	
AM01	Multilingual software development affects all aspects of software development through its development lifecycle. Additionally, new considerations must be made by the project manager for additional role, tasks, deadlines and contractual obligations related to the internationalization and localization of the product. For example, a multilingual software project may include additional stakeholders such as localization consultants and freelance translators.
AM02	Proper administrative and managerial practices may lead to significant cost cutting in translation costs and other multilingual software development activities. Additionally, novel approaches may allow teams to localize software applications that they had previously thought as impossible to competently localize due to economical restrictions. A competent team can overcome budgetary limitations of multilingual software develop due to high costs associated with software localization by utilizing novel approaches. For example, culturally adaptive software can minimize localization costs but require more effort during implementation. This approach is different to manual industry localization in that the software is designed and implemented such that it acquires details about the user's culture whilst being used and it adapts itself to the culture; the software localizes while being used. Similarly, crowd-sourcing is another very effective approach to localize even large software applications as it was proved by the internet giant Facebook (Malik, 2009). In this approach, provided a large and international user base is available, the software publisher can utilize them to localize the application into numerous locales. These methods are more suitable for free software or when limited or trial versions are made available for free use.
AM03	In commercial software, high localization can be achieved whilst minimizing costs if effective administrative steps are put in place, for example (Wigestrand, 1998): 1. Locate a freelance translator in the target market. 2. Fly the selected translator to home country for a short training on how to translate and localize the software product; agree on formats and files. 3. Establish an agreement about timeframes for localization and fly the translator back. 4. The translator then transmits all the localized material to the company as agreed.
AM04	Utilize online resources to find translators and localization resources: • http://www.translationzone.com

	• http://www.e-translate.com
AM05	Internet connected software applications can minimize the need for professional translators by utilizing online machine translations services like Google Translate API or Bing Translator. Large amounts of content in particular, as found in systems based on existing CMS, can maintain content in one language and use machine translation to translate and render the content in other languages. This minimizes costs, the need for localization experts and translation time, and translators can be used to improve final localization quality. Such approaches are suitable when translation quality does not need to be of high quality. See sections 4.3 and 5.1 for details.
AM06	Development and maintenance of multilingual software often means multinational and multicultural teams. Ways to do this include (see section 4.3 for details): • Use of offshore service providers • Finding local business partners • Building geographically scattered teams In all cases, the cultures and personalities of team members in such multinational and multicultural teams must be understood for success (Abufardeh & Magel, 2010).
AM07	When working with partners, employees or customers across the globe, geographical distance means a difference in work environments and time zones, which may leads to communication gaps, project delays, inconsistent quality and ambiguity. Cultural differences create mistrust, fear and unequal distribution of work. It also means different languages being used. All of this could increase project costs and cause reporting problems. Project managers must consider all these factors and a suggested remedy to these challenges is the utilization of cloud computing as suggested by (Hashmi, 2011). Cloud computing and available services can be utilized to increase interoperability, diversification, and the alignment of business with technology.

Table 5-3: Administrative and Managerial Guidelines

Guidelines for Feasibility Study	
FS01	During feasibility study, determining whether the project can and should be implemented, one or more of the following activities should be carried out (see section 4.4 for details): 1. Carry out a market analysis. 2. Analyze the possible development scenarios. 3. Analyze the existing software. If new software is being developed, the team will need to conduct market analysis (or customer

	requirements) and consider possible development scenarios to internationalize and localize the software. If existing and already internationalized software needs to be localized, then all three activities will need to be carried out. Similarly, in case of existing software that needs to be both internationalized and then localized, all three activities will need to be carried out.
FS02	At the end of feasibility study, it must be decided whether the software project is worth the effort and the budget for the project budget may be determined. Numerous alternative options for implementation can be considered to control the project costs with relevance to internationalization and localization. For example, among key decisions to be made is whether to outsource localization activities or carry out all activities in-house. This decision is influenced by factors such as content size, desired target languages, in-house expertise and the financial proposals submitted by localization vendors. See Chapter 2 and section 4.4 for details.
FS03	During feasibility study calculations, consider the following: • Costs of localization tools • Costs of translators for the estimated content size and target languages
FS04	If high quality translation is required, for example in commercially sold software products, then the services of localization vendors and professional translators, who often could be provided by the same company, will be needed if in-house expertise is insufficient. Make contacts with such service providers and get estimated costs. If high quality content translation isn't important, then less costly alternatives can be used, for example see guidelines AM02 and AM03. Also, see section 4.5 for localization vendors and service providers.

Table 5-4: Guidelines for Feasibility Study

Guidelines for Requirements Engineering	
RE01	The most important requirements of multilingual software that must be elicited during requirements engineering can be determined by asking: What languages shall be supported by the software? How will the software application switch from one language to another (statically; needs restart, or dynamically, without restart)? What modules must support multilingualism? On what level should the software, or its individual modules, support multilingualism (only user interface or backend processing and storage)?
RE02	The following quality requirements are relevant to multilingual software and must be considered by the project team members: maintainability, reusability, understandability, adaptability, and language neutrality. See section 4.4.2.

RE03	Requirements engineering challenges caused by internationalization are the combination of the same challenges internationalization caused to requirements elicitation, requirements prioritization, and requirements specification. Internationalization causes additional challenges to elicitation by increasing complexity in the stakeholder network of a product and worsening existing elicitation issues. Some of the solutions to these challenges include: • Central management of some of the requirements • Decentralization of some of the product requirements and features to geographic areas • Architecture and design to ease software customization and localization • Team re-organization to support decentralized control • Encourage the practice of maintaining a top 10 or 20 lists of requirements per locale See section 4.4.2 for details.
RE04	Language affects functional requirements and culture affects non-functional (quality) requirements. Internationalization requirements are culture and language independent and localization requirements are culture and language dependent (Abufardeh & Magel, 2009).
RE05	Categorization of requirements, during prioritization and triage activities, allows developers to determine the complexity and then estimate the time, cost, effort and resources needed to implement the requirement. A modified requirements categorization needs to be used for multilingual software that takes into account the local and culture sensitivity. A column with a name like "culture sensitive" should be added to the spreadsheet or table next to columns like risk and complexity where the elicited requirements are maintained. This column can have values like none, low, medium or high. For example, if a requirement affects all layers and modules and different language versions handle it different, it must be "high". Likewise, if a requirement, like a formula to determine the age from date of birth, remains the same in all language versions, it will be "none". Similarly, requirements prioritization needs to be modified to account for local and culture sensitivity, and different prioritization methods may be modified for this purpose. For example, the MoSCoW (Must, Should, Could, Wont) prioritization technique can be appended with a column to indicate the need to internationalize/localize a requirement. In such a case, the "Must Have" for a requirement would mean "Must Internationalize" (Abufardeh S. O., 2009).
RE06	Commercial contracts are subject to specific time plans and budgets, therefore during requirements engineering, it is important to understand exactly what needs to be multilingual and in what locales. The sooner this information is available, the better one may estimate the project cost and timeframe. Thus if possible, this information should be acquired before finalizing the

	financial contract. See 4.4.2 for details.

Table 5-5: Guidelines for Requirements Engineering

Guidelines for Architecture and Design	
AD01	The quality requirements such as availability, modifiability and usability are often achieved through deliberate architecting and designing using well known tactics, or architecture patterns and design patterns (Bass, Clements, & Kazman, 2003). Similarly, in order to architect and design multilingual software, it is important to utilize existing and proven tactics that lead to software applications that are internationalized, and as a result, these products can be efficiently localized into numerous languages. See 4.4.3 for details. Security and performance are two other quality requirements that are highly affected when internationalizing an application, so developers should pay additional attention to these requirements if they are important for the project. See 4.6 for details.
AD02	Development of internationalized software can greatly benefit from Aspect Oriented Programming (AOP) as it leads to isolation of tangled concerns and unification of scattered concerns. See 4.4.3 for details.
AD03	The common strategy of software internationalization that is the separation of code (core functionality like algorithms and business logic) from textual resources (such as user interface, help documents, dialog box, currency formats and prompt messages). The main assumption based on which the design of internationalized software takes place is that all of the culturally and linguistically sensitive components can be separated from the locale-independent core of the application. During the design phase, it is crucial for the development of internationalized software applications that designers identify and isolate locale-specific codes and modules from the rest. See 4.4.3 for details.
AD04	It is important for architects/implementers to select the implementation technology for a multilingual software product that supports multilingual software development. The availability of tools and IDEs that support multilingual software development must also be checked. Answering the following questions can improve the decision making process: • Does a text editor exist that supports both the desired target languages and the programming language syntax? • Can the given compiler compile multilingual code? • Does the emulator or executor run the compiled software application in multiple

	languages? See 4.4.3 for details.
AD05	It is important for architects and designers to understand how multilingual software is developed using different programming or architectural paradigms like Object Oriented Programming, Component-based Development and so on. See 4.4.3 for details.
AD06	For bespoke multilingual software development projects, consider using the Architecture Reference Model for Multilingual Software (ARMMS). It proposes a new model named Aspect-based Language Library for multilingual software development that can be utilized by architects during implementation and can be clearly understood by developers. See 4.4.3 for details.
AD07	UI is the most prominent aspect of multilingual software applications. Designers should be sensitive to different users' groups' cultures and languages. Creation of prototypes is an effective way to demonstrate the application early and gauge the users' responses. UI design for multilingual software should start with identification of the parts and controls that need to be localizable. Developers should also pay special consideration to text contraction and text expansion issues. This issue is very important today where an application may need to properly run on numerous devices with different screen sizes, such as laptops, notebooks, tablet computers, smart phones and standard computer monitors. UI layout and directionality should render different types of languages, RTL and LTR. For applications with multilingual support, a language neutral icon should be prominently displayed, for example on top of the screen, which allows users to change the language version. In bilingual systems, it is expectable to display the alternate language version using text written in that language. See section 4.4.3 for details.
AD08	For sophisticated user interfaces, especially common in commercial websites, the use of graphics can lead to serious issues that inhibit the development of multilingual user interfaces. To correctly design and implement interfaces that heavily rely on graphics (image files with extensions such as gif, jpg and png), ensure that: • Locale-dependent text is externalized from the graphics; use "layered graphics". For web-based applications, CSS and JavaScript can be used to display text on the graphics in real-time depending on the selected language. • Graphics are direction-neutral to avoid redundancy; neither RTL nor LTR. • Ensure graphics are culture neutral; colors have different meanings in different cultures
AD09	Special attention should be given to the database when the software is expected to store data in multiple languages. Database should be designed to accommodate localized text and depending

	on the requirements, it should support data in one or more languages. See 4.4.3 for details.
AD10	For Commercially Off The Shelf systems, analyze the system for its support for multilingual implementation. To be specific, check for: • The number of concurrent languages supported by the architecture • Support for Bidi languages • Functions and features that help write and maintain multilingual content Content Management Systems, for example, are regularly used and there are numerous limitations in some and others provide multiple approaches to multilingual content management. See section 5.1.
AD11	If software frameworks are selected for implementation, check whether I18N and L10N are supported and to what extent. See section 4.5.2 for details.
AD12	Often in projects, it is necessary to integrate software components with third-party products and services. In such cases, it is important to identify the integration requirements placed by the third-party developers. In particular, the third-party product or service will need to support the same language as the new product.

Table 5-6: Guidelines for Architecture and Design

Software Programming Guidelines	
PG01	See section 4.5 for multilingual support, tools and technologies available for the programming language being used in your project.
PG02	Software programmers developing multilingual applications must have a general understanding of additional issues relevant to software internationalization and localization. This includes properties of world languages that affect the software. World Languages are either Pictographic, such as Chinese and Japanese, or Orthographic, such as English, Swedish and Arabic. Pictographic languages comprise of symbols that have meanings and when combined form sentences. Orthographic languages comprise of alphabets that when combined form words with meanings. Languages can be right-to-left (RTL) or left-to-right (LTR). See 4.4.4 for details.
PG03	The development technology must be evaluated in light of its support of different world languages when developing multilingual software. Also, availability of internationalization and localization tools facilitates the process, especially when an existing monolingual software application needs to be internationalized, this should also be checked.

PG04	In order to internationalize a software component, the developer will need to externalize the text strings from the code, which means separation of any culture and language specific resources and elements from the source code and storing it in external resource files. This includes: • UI elements like message boxes and alerts, labels, component colors, icons, symbols, page layout and orientation, date and time formats including the calendar component, currencies symbols and formats, measurements, and phone and address formats. • Functionality such as searching, sorting, indexing, grammar and spelling checking.
PG05	Main issue in data repository (database or resource file) localization is clearly defining what fields and tables require localization. For each target locale, the following questions must be addressed: What text encoding will be supported? Will the target language filenames be supported? What format needs to be supported within the database for calendar, data/time and numbers? Are the data repositories shareable across language versions of the software application? What is default language for the data repositories (pre-populated databases)? In what languages will user entered database and file text be? How is language determined? What is the planned upgrade path for data repositories for future new languages?
PG06	Developers also must also take into consideration the underlying infrastructure and its support for different languages, including the users' operating systems. For example, many programming languages and technologies have UI components such as calendar and clock that adapt to the underlying settings of the operating system. See 4.4.4 for details.
PG07	In distributed applications, where communication occurs over a network, it is important to ensure that the character encoding scheme is supported. For example, when Arabic text is being transmitted between the client and the server, the network must support UTF-8 or Unicode.
PG08	There might be a functional requirement to search the system for data in multilingual languages, but using a single query. This function is also known as Multilingual Information Retrieval (MLIR) and to tackle it, consider the method based on Growing Hierarchical Self-Organizing Map (GHSOM) as suggested by (Yang & Lee, 2008).
PG09	In existing products that need to be internationalized, developers need to identify all culture-sensitive elements in the source code to internationalize it. This can be a time-consuming and difficult activity and specialized tools and approaches should be considered to automate the process. Check section 4.4.4 for details of a method suggested by Wang et al.

Table 5-7: Software Programming Guidelines

Guidelines for Software Testing

TG01	Multilingual software require a different testing approach and method, which take into consideration the target cultures, languages, and conventions used for date formats, numbers, currencies and so on. Bidirectional software (for users that speak Urdu, Farsi, Arabic or Hebrew) testing activities require additional consideration.
TG02	The team must carry out two kinds of testing in multilingual software projects, namely internationalization testing and localization testing. Internationalization testing is the verification process to ensure that the software application works as expected when localized in other languages. Localization testing is carried out to ensure that software functionality is not changed and no bugs were introduced after localization. See 4.4.5 for details of each type.
TG03	The team must prepare a suitable test environment to manage each of the supported languages, including the necessary keyboards, fonts and operating system. This can be costly.
TG04	The translation and translation testing, if possible and feasible, should be integrated into the development process. Abufardeh (Abufardeh S. O., 2009) recommends a work setting and process for a software development team. See section 4.4.5 for details.
TG05	It is strongly recommended that all test cases and test suites (including unit tests and GUI test cases) should be internationalized and localized, just as the software application itself, in order to ensure quality. This multiplies the effort of the testing team.
TG06	Multilingual software projects require multilingual testers to ensure each localized version is of expected quality. This can be expensive and time consuming. Projects with tight budgets can consider using the approach suggested by Guo et al. that allows testers to effectively test localized versions in non-native languages; an English speaking tester can test Arabic software without knowing Arabic. See 4.4.5 for details of this approach.

Table 5-8: Guidelines for Software Testing

Post-release and Maintenance Phase Guidelines	
PR01	In projects with large content and numerous target languages, often new language versions and contents are added post-release. The software product should be architected and implemented with this in mind, so non-technical translators and users can manage new content and language versions without technical assistance. This has numerous benefits, but to ensure software quality, administrative processes should be put in place so technical users and managers can approve changes. This is even more important for the longer term where staff change.

PR02	If the software was internationalized during implementation, then there is only a need to localize the product for the new locale. This could even be achieved without the need to modify the source code or use the services of technical users, provided the software was properly internationalized. In this case, do what was done in the past to localize the product. For new languages, consider Administrative and Managerial guidelines.
PR03	If the software is not internationalized, then the project team must analyze the existing software and decide on the most suitable and feasible solution, which might even be the complete redevelopment of the software from scratch. Consider technology dependent guidelines in 4.6 and architecture, design and programming related guidelines above. After due diligence (or feasibility study), the project team can decide on one of the following and accept related implications: 1. To cancel the project: if the project team members or source code are inaccessible and the project is forecasted to be over the budget and the costs are not justified. 2. To localize the software without internationalizing it: this may be the best option if the project budget is small and if the software needs to be localized to one additional locale. 3. To internationalize the project: although the most ambitious and costly choice, the long term benefits are numerous.
PR04	Internationalizing existing software is challenging and complex, especially if source code and project team members are not available. Numerous tools and techniques, for example that demonstrated by Daohe and Shusheng, can be considered in such cases. See 4.4.5 for details.
PR05	If source code is available, tools and techniques, with varying quality of internationalization, can be used to internationalize an existing monolingual product that minimize costs and improve efficiency. This includes the use of lexical analysis as shown by Peng et al., automated identification of "need to translate" strings as shown by Wang et al., and the use of tools like GNU Gettext. See 4.4.5 and 4.5 for details.
PR06	If project budget permits, utilize the services of professional localization vendors and/or buy specialized localization tools. See 4.5.3 for details.

Table 5-9: Post-release and Maintenance Phase Guidelines

5.3 Guidelines Navigation and Retrieval

The goal of this section is to present a means of accessing the relevant guidelines only from the large number presented in the previous section to encourage and facilitate multilingual software development. Any active software project in industry is in one of the following software lifecycle stages (Cockburn, 2006) (Davis A. M., 2005):

- Feasibility Study: at this stage the main goal is to determine whether the project is feasible in terms of time, budget and return on investment. Multilingual software development requires consideration of additional cost areas, without which the results of the feasibility study would be inaccurate.
- In-development (or implementation phase): this phase reflects bespoke software development projects currently under development, regardless of the implementation methodology. Multilingual software development affects all activities from requirements engineering to software testing.
- Post-release (or maintenance phase): once the software application, or specific functions in iterative projects, can be used by the end users, the maintenance phase commences. All existing software projects are considered to in this phase and it is important to provide guidelines to facilitate their internationalization and/or localization.

Therefore, three tools reflecting these stages have been devised to enable decision-makers access useful guidelines for multilingual software development only relevant to their project. For example, if a project is yet to commence and is still in the feasibility study phase, guidelines with relevant programming or modifying existing source code are irrelevant, and guidelines that provide insight about possible costing seem more appropriate. Sections 5.3.1 to 5.3.3 present three tools to navigate through the guidelines, one for each of the three phases. These tools have been named the Guidelines Navigation Tool (GNT), depicted below:

Model 5-1: GNT Types

To further increase the relevance of the guidelines, each GNT is supplemented with a number of Project Properties (PP), which in essence provides more information about the projects. For example, in the Post-release Phase GNT, PPs include "availability of the source code" and "available budget". The guidelines for multilingual software developer for a project where the source code is no longer accessible (in third party legacy systems for example) and where the budget is low would obviously be different to guidelines given when the source code is available and the available budget is high.

5.3.1 Feasibility Study GNT

Feasibility Study GNT is shown below (view horizontally from the right side). The included Project Properties need additional explanation:

- **Number of Target Language:** this refers to the number of languages the multilingual software is required to run in. More languages require more development and translation efforts.
- **Static Content Size (words):** software projects often include a large number of static content, for example privacy policy or the "about us" page. The larger the content, the more costly the professional translation services will be.
- **Content Cycle:** this PP refers to the frequency of change to the static content. Active content regularly change, like newsfeeds, inactive contents remain the same for large periods of time.
- **Development Type:** implications of multilingual software development in terms of project costs differ in a bespoke development project from Commercially Off The Shelf systems; thus the guidelines given to better understand the cost areas and project components are also different.
- **Software Pricing:** this refers to the intended project pricing plan; free represent software that the users won't have to pay for, commercial represent software only available if paid for, and mixed represents software with limited free versions. This PP mostly influences the guidelines related to translation of content in multilingual software.

5.3.2 In-Development (implementation) GNT

In-development GNT is shown below (view horizontally from the right side). The included Project Properties need additional explanation:

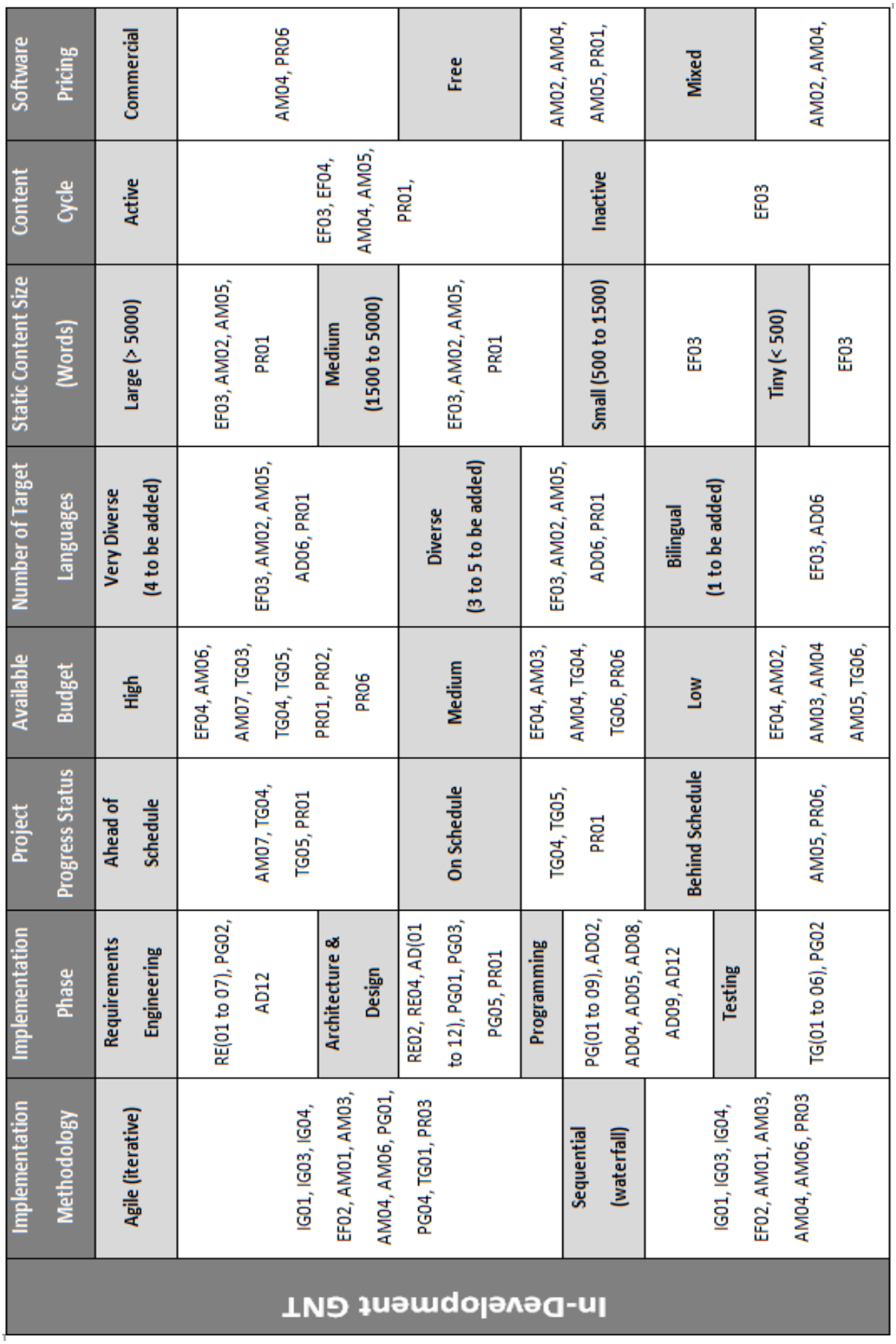
- **Implementation Methodology:** this refers to the implementation methodology; iterative or sequential. Some guidelines for multilingual software development are only relevant to agile and iterative methodologies, others to sequential processes.
- **Implementation Phase:** this PP presents guidelines for different SDLC phases and activities, like Requirements Engineering, Architecture & Design, Programming (coding), and Testing.
- **Project Progress Status:** this takes into consideration whether the project's development is ahead of, on or behind schedule. Some guidelines are time-consuming so would not be suitable for projects running behind schedule.
- **Available Budget:** this PP takes into consideration the available budget. For projects running on a low budget, suggested guidelines for multilingual software development are relatively less costly.
- **Number of Target Language:** see 5.3.1.
- **Static Content Size (words):** see 5.3.1.
- **Content Cycle:** see 5.3.1.
- **Software Pricing:** see 5.3.1.

5.3.3 Post-release (maintenance) GNT

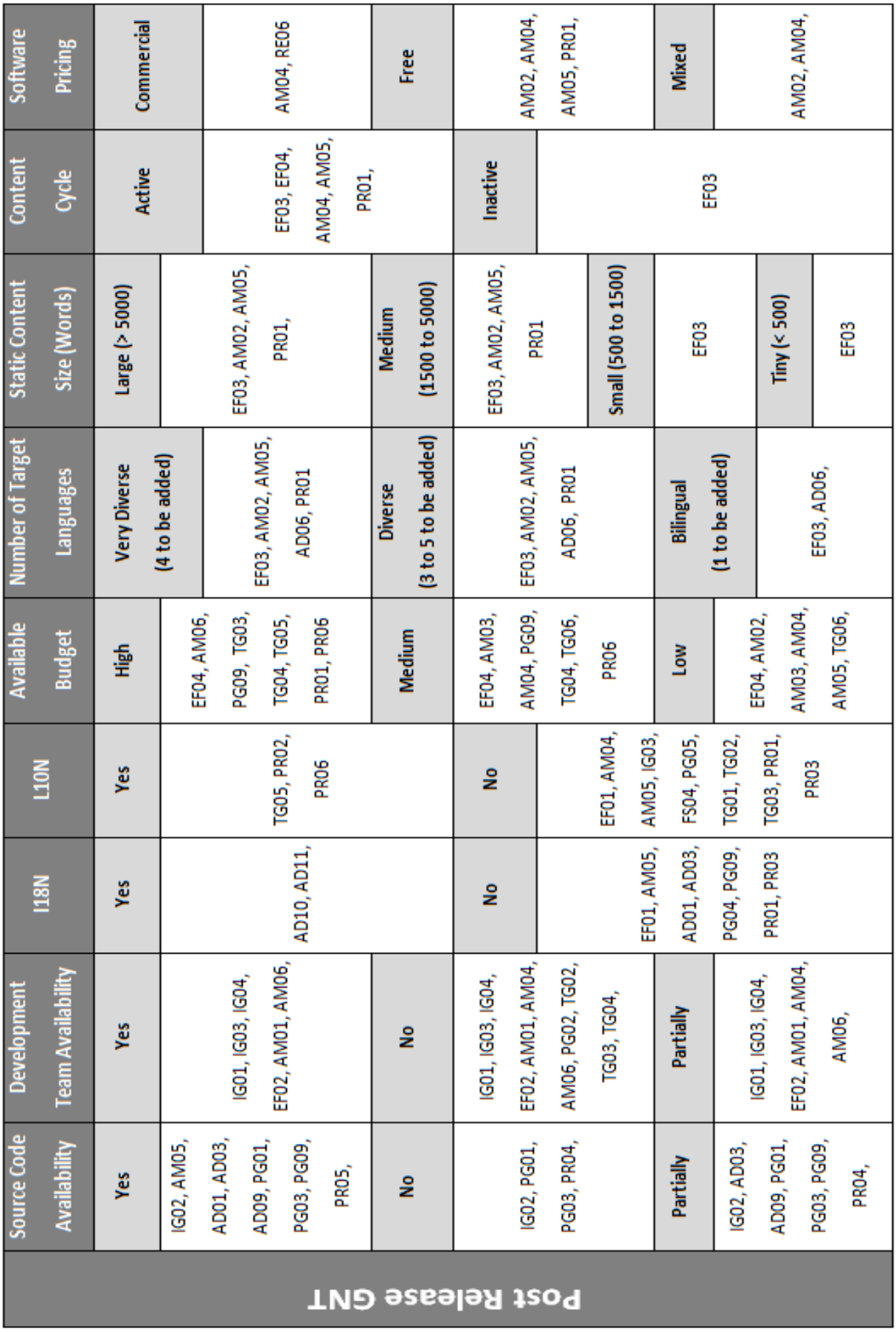
Post Release GNT is shown below (view horizontally from the right side). The Project Properties need additional explanation:

- **Source Code Availability:** this checks if the source code is available; without the source code there are very few guidelines for multilingual software development.
- **Development Team Availability:** this checks if the development team of the product is still accessible and available. Without the development team, it's far more difficult to internationalize a product and alternative guidelines might be more suitable.
- **I18N:** this PP checks if the software product is already internationalized (facilitates localization).
- **L10N:** this PP checks if the software product has been localized before (already multilingual).
- **Available Budget:** see 5.3.2.
- **Number of Target Language:** see 5.3.1.
- **Static Content Size (words):** see 5.3.1.
- **Content Cycle:** see 5.3.1.
- **Software Pricing:** see 5.3.1.

Model 5-2: Feasibility Study GNT



Model 5-3: In-Development GNT



Model 5-4: Post Release GNT

5.4 GNT Scenarios and Usage

This section presents three scenarios that exemplify how GNT can be used for real life projects.

First, the user must decide which of the three GNTs should be used. If the project is still under study and yet to start, then the Feasibility Study GNT is applicable. If the project implementation has already commenced and is in one of the SDLC phases, then the In-Development GNT should be used. If the software product already exists or released for use, then the Post Release GNT is applicable.

Next, each GNT comprises of Projects Properties (PP), as explained in the previous sections, and the user must select answers (or values) for each of the PPs shown in the GNT that match the concerned software project or product. Each answer has corresponding guideline codes (as given in the tables in section 5.2) and the user must note all this codes, moving from one PP to another. Once all the PPs are covered in the GNT, duplicated ones should be ignored. Finally, the unique guidelines should then be used to check the detailed guidelines as listed in section 5.2.

The following scenarios practically show this:

Scenario 1:

If a software is in the feasibility study phase and it must be available in more than 5 target languages, the static content size (number of words that will need to be translated in different language) in the software is small (less than 1,000 words), the software is a bespoke development project and the software will be available for use for free.

Since the project is in the feasibility phase, the Feasibility Study GNT should be used. Using the information given in the scenario, the answers of the PPs in the GNT will result in the following guidelines (duplicated guidelines have been deleted):

EF03, AM02, AM05, AM04, IG01, IG03, AM01, AM03, AM06, AM07, RE01, AD01, AD04, AD08, PG01, PG02, TG03, TG06, PR03.

The details of these guidelines codes can now be retrieved from section 5.2.

Scenario 2:

If a software product is currently being developed using Agile practices in Java, where the project is in the testing phase and is ahead of schedule and more than 70% of the project is complete. The available budget is low, but the team decides that the software needs to be localized in 4 more languages. The static content size (size of text that needs to be translated) is large (more than 5,000 words) and the content regularly changes. The system will be made available for free use.

Since the project is in the development phase, the In-Development GNT should be used. Using the information given in the scenario, the answers of the PPs in the GNT will result in the following guidelines (duplicated guidelines have been deleted):

IG01, IG03, IG04, EF02, AM (01 to 07), PG01, PG04, PR03, TG (01 to 06), PG02, PR01, EF04, EF03, AD06, EF03.

The details of these guidelines codes can now be retrieved from section 5.2.

Scenario 3:

An existing software system currently available in only one language needs to be localized into another language. The source code and project developers are both accessible. The software is not internationalized and not localized, where the budget is low and one language needs to be added. Also the static content is tiny and regularly changes (active). The system will be made available for commercial use.

Since the product already exists, the Post Release GNT should be used. Using the information given in the scenario, the answers of the PPs in the GNT will result in the following guidelines (duplicated guidelines have been deleted):

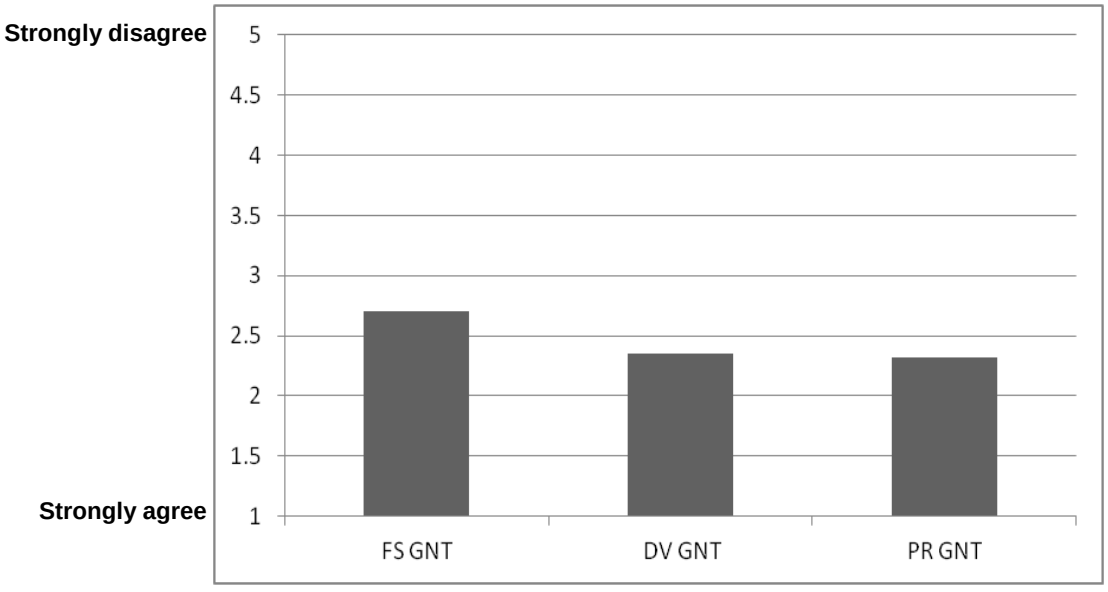
IG (01, 04) AM (01, 05), AD01, AD03, AD09, PG01, PG03, PG09, PR05, EF (01, 04), PG04, PR01, PR03, FS04, PG05, TG (01, 03), TG06, AD06, RE06.

The details of these guidelines codes can now be retrieved from section 5.2.

6 Validation of Results

To validate and get feedback on the guidelines and the suggested usage of GNTs and associated PPs, industry professionals and academicians were given a number of scenarios related to software project. To be specific, for the Feasibility Study GNT three scenarios reflecting different answers to the PPs were given. For the In-Development GNT four scenarios were given. Finally, for the Post Release GNT two scenarios were given. A total of five respondents completed the survey and the rest did not attempt or left the survey incomplete. The feedback was given by either strongly agreeing, agreeing, being neutral, disagreeing, or strongly disagreeing. For the all the given guidelines, on average the respondents replied by agreeing. For feasibility study related guidelines, on average the respondents leant towards being neutral, neither agreeing nor disagreeing. For development and post release related guidelines, one average the respondents replied by agreeing. This is depicted in model 6-1 below.

In the survey not all the guidelines were evaluated and some were a combination of multiple guidelines given in section 5.1. Although not all the guidelines are covered in the survey (see Chapter 3 and Chapter 7 for more information about the validation done, its context and limitations), from among the ones included, the respondents together agreed and disagreed with some. In particular, AM03 was disagreed with by 80% of the respondents and the reasons for this are not totally clear. Perhaps the disagreement is because AM03 is extracted from a paper published in 1998 and seems to be only relevant when high quality translation is an absolute necessary and it suggests flying in the translator, which might not be necessary today as web conferencing is common and effective today.



Model 6-1: Average responses per GNT

On the contrary, AM05, FS03 and QD08 were agreed or strongly agreed with by 80% of the respondents. Similarly, 100% of the respondents responded favorably towards FS04 and RE05.

7 Discussion

This thesis compiles numerous solutions and practices documented in literature and utilized by industry practitioners to present a comprehensive source with numerous guidelines for multilingual software development that are applicable to a wide range of software projects. Means of easily navigating and retrieving these guidelines are also presented. Therefore, in essence, this thesis project contributes by compiling all previous works and presenting them in an accessible manner.

The industry interviews and the observations made strongly suggest that there is a need to raise awareness about multilingual software development and its direct and indirect benefits. The guidelines for multilingual software development that have been extracted from literature and industry, and categorically presented in this thesis, provide a bird's eye view of the technical and non-technical activities related to software internationalization and localization. This facilitates multilingual software development by reducing the complexity and time required to understand the different aspects related to the topic. The reviewed literature is thorough and includes all aspects of software, including the development lifecycle. It has also been attempted to incorporate latest industry trends, including agile methodologies and mobile apps development.

Nevertheless, this research project only takes into consideration software projects and the possible relation with or application in the field of embedded software development, for example, is unclear and not studied. Additionally, as it can be understood, it is not possible to cover every possible practice, technology, tool or guideline for multilingual software development. For example, not all tools and programming languages are studied in light of multilingual software development, but the main ones have been included.

Few of the guidelines for multilingual software development have been extracted from the analyzed industry projects and the related interviews. Unfortunately, a large number of the sample projects are very similar in nature; websites that are implementations of CMS software. Ideally, diverse projects implemented using different technologies and in different domains would have possibly provided a more diverse set of guidelines. Nevertheless, the industry interviews did help understand the main concerns and challenges as well as the motivations for multilingual software development.

Although it is mentioned a number of times that a wide range of solutions already exist for multilingual software development and there is a need to raise awareness, it is not meant by this that there is not a need for further advancements. In particular, the field of machine translation still needs improvements to translate content with high quality, which is a major hurdle for software localization.

The Guidelines Navigation Tool (GNT) does not accurately help retrieve guidelines. Originally the idea was to take into consideration every answer for the PPs in the GNT and then suggest unique guidelines. However, given the very large number (over 100 in case of two GNTs) of possible unique combinations of the answers, it would have taken a very long time to suggest precise guidelines. To remedy this, it was decided to write the guidelines in a more generalized manner and simply link them to various answers to the

PPs in the GNTs. Perhaps, in the future this can be made more accurate, especially if the GNT is made available in form of a software application then automatically helps the user retrieve the relevant guidelines.

Moreover, currently some suggested guidelines may conflict one another and the reader must subjectively handle this. This is mainly due to the fact that the guidelines are written in a generalized way and are not suggested after taking into consideration the unique combination of answers to the PPs in the GNT. Also, some guidelines in the Feasibility Study GNT are suggested to indicate possible cost areas once the development commences.

7.1 Implications for Industry

Software developers and purchasers in industry can utilize the results of this project to plan and implement multilingual software in a more informed manner. A key reason for a lack of or incorrect multilingual software development is the lack of awareness. 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. For instance, purchasers often fail to evaluate competing software products in light of the multilingual requirements. Similarly, project managers and administrators struggle to incorporate additional personnel in multilingual software, and inexperienced developers wrongly estimate, or worse wrongly implement, such software. While larger organizations have the budget to experiment and devise suitable solutions, small and medium enterprises often ignore internationalization and localization, which leads to losses both in the short and long terms.

The compiled and presented guidelines can allow industry practitioners to make better and more confident decisions that can lead to financial benefits and software of higher quality. Hopefully, this in time can lead to better awareness and an increased number of multilingual software around an increasing globalized and multicultural world.

7.2 Implications for Academia

The majority of guidelines presented in this project were extracted from published literature on specific elements of multilingual software development. By organizing all categorically, the domain of internationalization and localization can be viewed from a high level and this helps identify areas that need further research. For example, the participants in this project identified lack of high quality automated translation tools as a major hurdle to implement multilingual software. This indicates that alternative methods and improved automated translation tools are much needed and can add tremendous value towards multilingual software development. Similarly, research is needed to incorporate multilingual software development practices among emerging agile methodologies and mobile platform implementations.

In addition to identifying gaps and possible research areas, this work has identified key factors that discourage multilingual software development in industry, especially among organizations with limited budgets and resources. Academicians can collaborate with industry to remedy these issues.

8 Conclusion

Multilingual software development is necessary for software developers today to remain competitive in the global information technology industry connected through the internet. The awareness on the importance of multilingual software development, both for commercial gains and to meet the requirements of an ever increasing multicultural and multilingual user base, can be raised by providing suitable guidelines to the software project decision makers.

A diverse list of guidelines for multilingual software development that crosscut all the phases of software development life cycle is compiled and categorically presented in this paper. This includes guidelines relevant to multilingual software development in general, administrative and financial implications, feasibility study, requirements engineering, architecture and design, software programming, and software testing. Also, guidelines for multilingual software development for existing software products (post release) and the most commonly used technologies are given.

To enable decision makers and project team members to access only the relevant guidelines in timely and effective fashion, a Guidelines Navigation Tool was devised.

Bibliography

Abufardeh, S. O. (2009). *A framework for the integration of internationalization and localization activities into the software development process.* Fargo: North Dakota State University Fargo, ND, USA ©2009.

Abufardeh, S., & Magel, K. (2009). Software Internationalization: Crosscutting Concerns across the Development Lifecycle. *New Trends in Information and Service Science* (pp. 447-450). Beijing: Conference Publishing Services.

Abufardeh, S., & Magel, K. (2009). Software Internationalization: Testing Methods for Bidirectional Software. *Fifth International Joint Conference on INC, IMS and IDC* (pp. 226-231). Seoul: IEEE Computer Society.

Abufardeh, S., & Magel, K. (2010). The impact of global software cultural and linguistic aspects on Global Software Development process (GSD): Issues and challenges . *New Trends in Information and Service Science* (pp. 133-138). Gyeongju: IEEE Seoul Section.

Acclaro TM. (2011, February 1). *Agile Software Development and Localization*. Retrieved December 1, 2011, from Acclaro Newsletter: http://www.acclaro.com/newsletter/software-localization-agile-development

Adobe Developer Connection. (2011, December 1). *Flex Documentation*. Retrieved December 20, 2011, from Adobe Developer Connection: Flex Documentation

Adobe Labs. (2011, December 1). *Text Layout Framework*. Retrieved December 20, 2011, from Adobe Labs: http://labs.adobe.com/technologies/textlayout/

Alchemy Software. (2011, 12 1). *Alchemy Software*. Retrieved 11 1, 2011, from Alchemy Software: http://www.alchemysoftware.ie/

Alexa.com. (2011, December 24). *Top Sites in Sweden*. Retrieved December 24, 2011, from Alexa.com: http://www.alexa.com/topsites/countries/SE

Alsanad, A. (2011, May 14). *Arabic Android*. Retrieved 12 20, 2011, from Arabic Android: http://ardoid.com/

Android Developers. (2011, January 1). *String Resources*. Retrieved December 15, 2011, from Android Developers: http://developer.android.com

Apple Inc. (2010, January 1). *Introduction to Internationalization Programming Topics.* Retrieved December 20, 2011, from Mac OS X Developer Library: http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPInternational

Arabicode. (2011, August 6). *FlarabyAS3Flex*. Retrieved December 20, 2011, from Arabicode: http://www.arabicode.com/

Arefin, M. S., Morimoto, Y., & Yasmin, A. (2011). Multilingual Content Management in Web Environment. *International Conference on Information Science and Applications.* Jeju Island, Korea (South) : IEEE Computer Society.

Ashrafi, R., & Murtaza, M. (2010). ICT Adoption in SME in an Arab GCC Country Oman. In E. Alkhalifa, *E-Strategies for Resource Management Systems: Planning and Implementation* (pp. 351-375). Pennsylvania: IGI Global.

Atkins-Kruger, A. (2010, December 10). *Google Rolls Out Foreign Language Search Features Globally*. Retrieved July 15, 2011 , from Multilingual Search: http://www.multilingual-search.com/google-rolls-out-foreign-language-search-globally/

Bai, G. (2011). *jQuery Mobile First Look.* New York: PACKT PUBLISHING.

Baker, M. (2009, October 2). *What is a Software Framework? And why should you like 'em?* Retrieved December 15, 2011, from Cimetrix Blog: http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice.* Boston: Addison Wesley.

Beninatto, R. (2011, May 29). *Google Translate API Deprecation Causes Commotion*. Retrieved Dec 5, 2011, from Localization Industry 411: http://www.l10n411.com/2011/05/google-translate-api-deprecation-causes.html

Bogdan, R. (2002). *Qualitative Research for Education: An Introduction to Theories and Methods.* Allyn & Bacon.

Burke, S. M., & Lachler, J. (1999, December 1). *Localization and Perl: gettext breaks, Maketext fixes.* Retrieved December 15, 2011, from cpan.org: http://search.cpan.org/dist/Locale-Maketext/lib/Locale/Maketext/TPJ13.pod

Callahan, E. (2005). Interface Design and Culture. *Annual Review of Information Science and Technology* , 39.

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game*. Boston: Addison Wesley.

Daohe, C., & Shusheng, Z. (2010). Realization of software multilingual based on Masm32 . *Computer and Communication Technologies in Agriculture Engineering (CCTAE)* (pp. 127-129). Chengdu : IEEE.

Davis, A. M. (2005). *Just Enough Requirements Management : Where Software Development Meets Marketing* . New York: Dorset House Publishing.

Davis, J., Tierney, A., & Chang, E. (2005). A User Adaptable User Interface Model to Support Ubiquitous User Access to EIS style applications. *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, (pp. 351-358).

Denzin, N., & Lincoln, Y. (2005). *The SAGE Handbook of Qualitative Research.* California: Sage Publications Inc.

Dillinger, M. (2010, July 1). *An introduction to Machine Translation.* Retrieved December 1, 2011, from http://www.mt-archive.info: http://www.mt-archive.info/AMTA-2010-Dillinger.pdf

Esselink, B. (2000). *A Practical Guide to Localization.* Amsterdam: John Benjamins Pub Co.

Feldman, A. (2011, June 3). *Spring Cleaning for Some of our APIs* . Retrieved August 1, 2011, from The Official Google Code Blog: http://googlecode.blogspot.com/2011/05/spring-cleaning-for-some-of-our-apis.html

Fuchs, S., & Minarik, K. (2011, December 1). *Rails Internationalization (I18n) API.* Retrieved January 2, 2012, from Rails Guides: http://guides.rubyonrails.org/i18n.html

Galin, D. (2003). *Software Quality Assurance: From Theory to Implementation.* Boston: Addison Wesley .

Glaser, B. (1992). *Basics of Grounded Theory Analysis: Emergence Vs. Forcing.* Sociology Pr .

GNU Gettext. (2010, 6 6). *Gettext*. Retrieved 11 1, 2011, from GNU: http://www.gnu.org/s/gettext/

Google. (2011, January 1). *Inside Google Translate*. Retrieved August 1, 2011, from Google Translate: http://translate.google.com/about/index.html

Gorton, I. (2011). *Essential Software Architecture.* Berlin: Springer.

Guo, X., Tay, W., Sun, T., & Urra, R. A. (2008). IEEE International Conference on Networking, Sensing and Control. *ICNSC* (pp. 751-755). New York: IEEE.

Hall, S. (2008). *Getting Started with IPhone SDK, Android and Others.* Queensland: Emereo Publishing.

Hashmi, S. I. (2011). Using the Cloud to Facilitate Global Software Development Challenges. *ICGSE Workshops* (pp. 70-77). ICGSE .

Henderson, N. (2009). Managing Moderator Stress: Take a Deep Breath. You Can Do This! *Marketing Research* , 28-29.

Hennink, M., Hutter, I., & Bailey, A. (2010). *Qualitative Research Methods.* London: Sage Publications Ltd.

Heuer, D. (2004, January 17). *Modular vs. Monolithic: The winner is?* Retrieved 12 25, 2011, from Free Code: http://freecode.com/articles/modular-vs-monolithic-the-winner-is

Hogan, J. M., Ho-Stuart, C., & Pham, B. (2004). Key challenges in software internationalisation. *Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation* (pp. 187-194). Darlinghurst: Australian Computer Society.

IEEE Standards Association. (2000). *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems.* New York: IEEE Computer Society.

Indiana University. (2008, April 7). *What are the differences between ASCII, ISO 8859, and Unicode?* . Retrieved December 20, 2011, from Knowledge Base: http://kb.iu.edu/data/ahfr.html

Internet World Stats. (2010, June 30). *INTERNET WORLD USERS BY LANGUAGE*. Retrieved August 5, 2011, from Internet World Stats: http://www.internetworldstats.com/stats7.htm

Jantunen, S., Smolander, K., & Gause, D. C. (2007). How Internationalization of a Product Changes Requirements Engineering Activities: An Exploratory Study . *Requirements Engineering Conference* (pp. 163-172). New Delhi: IEEE Computer Society.

Kane, N. (2011, July 9). *Should You Have a Multi-Language Website?* Retrieved July 15, 2011, from TheWebsiteCenter: http://www.thewebsitecenter.net/blog/2011/07/should-you-have-a-multi-language-website/

Krueger, R. (2000). *Focus Groups: A Practical Guide for Applied Research.* California: Sage Publications Inc.

Lingobit. (2011, 12 1). *Lingobit*. Retrieved 11 1, 2011, from Lingobit: http://www.lingobit.com/

Lingoport.com. (2008, March 27). *JavaScript Internationalization – the Good, the Bad, and the Ugly*. Retrieved December 10, 2011, from Lingoport Articles: http://www.lingoport.com/software-internationalization-articles/javascript-internationalization-the-good-the-bad-and-the-ugly/

Loosemore, S., Stallman, R. M., McGrath, R., & Oram, A. (1994, December 1). *Locales and Internationalization.* Retrieved December 15, 2011, from The GNU C Library: http://www.chemie.fu-berlin.de/chemnet/use/info/libc/libc_19.html

MacUpdate.com. (2005, January 1). *iLocalize*. Retrieved December 15, 2011, from MacUpdate: http://www.macupdate.com/app/mac/14191/ilocalize

Malik, O. (2009, March 11). *Is It Truly a Planet Facebook*. Retrieved July 15, 2011, from GIGAOM: http://gigaom.com/2009/03/11/it-is-truly-a-planet-facebook/

Martin, P., & Turner, B. (1986). Grounded Theory and Organizational Research. *The Journal of Applied Behavioral Science* , 141-157.

Microsoft Developer Network (MSDN). (2011, December 1). *Microsoft .NET Internationalization*. Retrieved December 15, 2011, from Microsoft Developer Network (MSDN): http://msdn.microsoft.com/en-us/goglobal/bb688096

Mowbray, T. J., & Malveau, R. (2003). *Software Architect Bootcamp.* New Jersey: Prentice Hall.

Mudawwar, M. F. (1997). Multicode: a truly multilingual approach to text encoding. *Computer* , 37-43 .

Multilizer. (2011, 11 1). *Multilizer Benefits*. Retrieved 11 1, 2011, from Multilizer : http://www2.multilizer.com/mlzproducts/mlzbenefits/

OASIS. (2006, October 12). *Reference Model for Service Oriented Architecture.* Retrieved December 25, 2011, from OASIS: Advancing Open Standards for the Information Society : http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

OLIF. (2008, December 1). *The open XML language data standard*. Retrieved December 20, 2011, from OLIF: http://www.olif.net

Oracle Sun Developer Network (SDN). (2010, January 1). *I18N Guidelines for C and C++*. Retrieved December 15, 2011, from Oracle SDN: http://developers.sun.com/solaris/articles/i18n/Cguidelines.I18N.html

Oracle Sun Developer Network (SDN). (2010, December 1). *Java Internationalization*. Retrieved December 10, 2011, from Oracle Sun Developer Network (SDN): http://java.sun.com/javase/technologies/core/basic/intl/

Oracle Sun Developer Network (SDN). (2011, January 1). *XLIFF: An Aid To Localization*. Retrieved December 1, 2011, from Oracle Sun Developer Network (SDN): dsc.sun.com/dev/gadc/technicalpublications/articles/xliff.html

Peng, W., Yang, X., & Zhu, F. (2009). Automation technique of software internationalization and localization based on lexical analysis. *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (pp. 970-975). New York: ACM New York.

Perl.org. (2010, January 1). *Perllocale.* Retrieved December 15, 2011, from Perl 5 Documentation: http://perldoc.perl.org/perllocale.html

PHP. (2012, January 6). *Human Language and Character Encoding Support.* Retrieved January 8, 2012, from PHP Manual: http://www.php.net/manual/en/refs.international.php

Reinecke, K., & Bernstein, A. (2007). Culturally adaptive software: moving beyond internationalization. *UIHCII'07 Proceedings of the 2nd international conference on Usability and internationalization* (pp. 201-210). Berlin: Springer-Verlag .

Renu, R. (2004). *Software Testing - Effective Methods, Tools and Techniques.* Tata McGraw Hill.

ResX Localization Studio. (2011, 1 1). *ResX Localization Studio*. Retrieved 11 1, 2011, from ResX Localization Studio: http://resx-localization-studio.net/

Ricca, F., Pianta, E., & Girardi, C. (2002). Restructuring Multilingual Web Sites. *Proceedings of the International Conference on Software Maintenance* (pp. 290-300). Washington: IEEE Computer Society.

Schaudin. (2011, 11 1). *RC-WinTrans*. Retrieved 11 1, 2011, from Schaudin: http://www.schaudin.com/web/solutions.aspx

Schifferes, S. (2007, January 21). *Globalisation Shakes The World*. Retrieved July 21, 2011, from BBC News: http://news.bbc.co.uk/2/hi/business/6279679.stm

Schmitt, D. A. (2000). *International Programming for Microsoft Windows.* New York: Microsoft Press.

Schwaber, K. (2004). *Agile Project Management with Scrum.* Microsoft Press: Seattle.

SDL . (2011, 12 1). *SDL Passolo*. Retrieved 11 1, 2011, from SDL: http://www.sdl.com/en/language-technology/products/software-localization/sdl-passolo.asp

SEO Translator. (2011, April 15). *Website Localization Impacted by Google Algorithms*. Retrieved August 13, 2011, from SEO Translator: http://www.seo-translator.com/website-localization-impacted-by-new-google-algorithms/

Shneiderman, B. (1992). *Designing the User Interface - Strategies for Effective Human Computer Interaction.* Readin: Addison-Wesley Publishing.

SimulTrans. (2011, January 1). *Localization Return-on-Investment* . Retrieved August 10, 2011, from SimulTrans: http://www.simultrans.com/education/articles/27-projectmanagement/32-localization-roi

Smith-Ferrier, G. (2006). *.NET Internationalization: The Developer's Guide to Building Global Windows and Web Applications.* Boston: Addison-Wesley Professional.

Somerville, M. (2007, December 8). *JavaScript Internationalisation*. Retrieved December 10, 2011, from 24Ways.org: http://24ways.org/2007/javascript-internationalisation

Stivala, E. (2010, April 19). *Tips for planning multi-language websites*. Retrieved July 20, 2011, from Econsultancy: http://econsultancy.com/uk/blog/5763-tips-for-planning-multi-language-websites

Taft, D. K. (2010, December 20). *Application Development: Java, C, C++: Top Programming Languages for 2011*. Retrieved January 02, 2012, from eWeek.com: http://www.eweek.com/c/a/Application-Development/Java-C-C-Top-18-Programming-Languages-for-2011-480790/

TIOBE Software. (2011, December 1). *TIOBE Programming Community Index for January 2012*. Retrieved January 02, 2012, from TIOBE Software: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Torraco, R. J. (2005). Writing Integrative Literature Reviews: Guidelines and Examples. *Human Resource Development Review* , 356-367.

Venkatasamy, P. V. (2009). *An Architectural Reference Model for Multilingual Software: A Comprehensive Development Approach for Multilingual Software.* Saarbrücken: VDM Verlag Dr. Müller.

Vine, A. (2004). Internationalization and Unicode Conference. *W3C.* Washington: Sun Developer Network (SDN).

Visual Localize. (2011, 11 1). *Visual Localize*. Retrieved 11 1, 2011, from Visual Localize: http://www.visloc.com/

W3C. (2007, April 12). *Internationalization Best Practices: Specifying Language in XHTML & HTML Content.* Retrieved December 10, 2011, from W3C: http://www.w3.org/TR/i18n-html-tech-lang/

Wampler, B. E. (2002). *The Essence of Object-Oriented Programming with Java and UML*. Boston: Addison-Wesley Professional .

Wang, X., Zhang, L., Xie, T., Mei, H., & Sun, J. (2009). Locating need-to-translate constant strings for software internationalization. *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on Software Engineering* (pp. 353-363). Vancouver: IEEE.

Welzer, T., Golob, I., Druzovec, M., & Kamisalic, A. (2005). Internationalization as a part of the database development . *Computational Cybernetics* (pp. 145-147). Mauritius: IEEE.

Wigestrand, H. (1998). Innovative and interactive internationalization of software and documentation. *Professional Communication Conference* (pp. 141-144). Quebec: IEEE Professional Communication Society.

Wikipedia Alchemy Catalyst. (2011, 12 1). *Alchemy Catalyst*. Retrieved 11 15, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Alchemy_Catalyst

Wikipedia Character Encoding. (2011, 12 1). *Character Encoding*. Retrieved 11 30, 2011, from Wikipedia Character Encoding: http://en.wikipedia.org/wiki/Character_encoding

Wikipedia Character Encoding. (2011, December 1). *Character Encoding*. Retrieved December 20, 2011, from Wikipedia Character Encoding: http://en.wikipedia.org/wiki/Character_encoding

Wikipedia Gettext. (2011, 12 1). *Gettext*. Retrieved 11 1, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Gettext

Wikipedia SDL Passolo. (2011, 12 1). *SDL Passolo*. Retrieved 11 1, 2011, from Wikipedia: http://en.wikipedia.org/wiki/SDL_Passolo

Wikipedia Software Frameworks. (2011, December 1). *Software Frameworks*. Retrieved December 220, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Software_framework

Wikipedia Translation Memory. (2011, Janruary 1). *Translation Memory*. Retrieved December 20, 2011, from Wikipedia Translation Memory: http://en.wikipedia.org/wiki/Translation_memory

Wikipedia Web Applications Frameworks. (2011, December 1). *Web Application Frameworks Comparison*. Retrieved December 15, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

Wooten, A. (2010, September 24). *Internationalization Benefits*. Retrieved July 20, 2011, from The International Business Edge: http://www.globalization-group.com/edge/2010/09/internationalization-benefits/

Yang, H.-C., & Lee, C.-H. (2008). Multilingual Information Retrieval Using GHSOM . *Intelligent Systems Design and Applications (ISDA)* (pp. 225-228). Kaohsiung : ISDA.

Young, E. (2001). *A FRAMEWORK FOR THE INTEGRATION OF INTERNATIONALIZATION.* Vermillion: The University of South Dakota.

Zetzsche, J. O. (2005, May 06). *Tool Kit * Software Localization Tools? Do We Really Need Those?* Retrieved 11 03, 2011, from Translators Cafe: http://www.translatorscafe.com/cafe/article35.htm

Appendices

Appendix A: Sample Industry Projects Interview Questions

1. Name, email and company's name?

2. Your job title?

3. Your role in the software project?

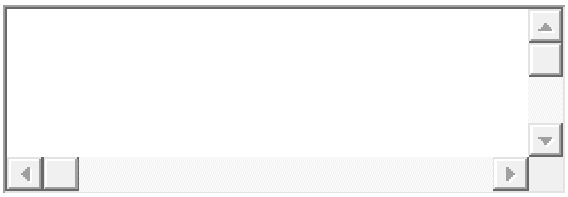
4. Please write a brief description of the project? Write about its purpose, main functions, end users, and implementation technology?

5. What was the project team size?

6. Which software implementation methodology was used, if any?

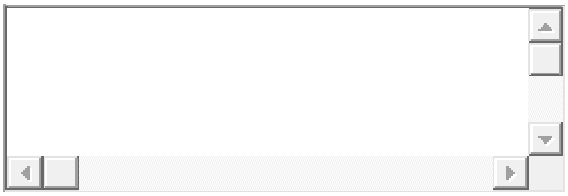
7. Is your project still under development (are new features still being added)?
   - ○ Yes
   - ◉ No
8. Are majority of your project's team members still available?
   - ○ Yes
   - ○ No
   - ○ Partially
9. Do you still have access to the project source code?
   - ○ Yes
   - ○ No
10. What was the motivation to internationalize (or not internationalize, if your software is in only one language) the software?

11. In hindsight, were these motivations worth the effort?
    - Yes
    - No
12. What were (or would have been, if your software is in only one language) the main difficulties to internationalize the software in the different project phases?

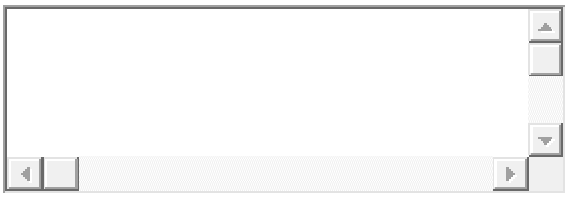


13. If you were to redo this project, what changes would you do in the different development phases, with relevance to software internationalization?

14. Technically, in terms of architecture, design or technology, how did you internationalize your software?
    Note: Ignore this question if your software is not internationalized (supports one language only).



15. Referring to the previous question, why did you choose this technical solution?
    Note: Ignore this question if your software is not internationalized (supports one language only).



16. In brief, what are the advantages and disadvantages of this technical solution in your view?
    Note: Ignore this question if your software is not internationalized (supports one language only).

17. In the different language versions of your software, what are the main differences, if any, in terms of content, functionality or features?
    Note: Ignore this question if your software is not internationalized (supports one language only).

18. Can you use the same technical solution if you were to in future implement a new version of the software for another platform (for example, mobile phone)?

    Note: If your software is not internationalized (supports one language only), answer this question instead:

    In retrospect, if you were to redo the project, would you consider internationalization?

    ○ Yes

    ○ No

    ○ Maybe

19. If you were to introduce a new language for the software, how difficult or easy would it be in terms of technical resources, time and budget?

	1	2	3	4	5		N/A
Easy	○	○	○	○	○	Difficult	○

20. Referring to the previous question, what is the exact process (steps), if any, to introduce a new language to your software?

Appendix B: Experts Survey for Results Feedback and Validation

Feasibility Study

Scenario 1: A new software is being developed and it must be available in more than 5 target languages, the static content size (number of words that will need to be translated in different language) in the software is small (less than 1,000 words), and the software will be available for use for free. Please rate the following guidelines.

Guideline 1: Freelance translators available online should be used to ensure quality translations. This is appropriate because the content size is small. Similarly, localization vendors can be contracted. This approach will ensure the best quality possible but will cost more too. Flying in freelance translators from different countries to improve understanding if the project is long term and complex is recommended.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: If translation quality for different versions is not very important, use free or paid automated machine translation services like Google Translator API or Microsoft Translate to automatically translate the different versions.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 3: The software will be available for use in 5 or more languages. In this case, the cost of localizing (making it available in different languages) the software can be significantly decreased by Crowd Sourcing. This means, the software will offer means by which the users themselves will thoroughly translate and retranslate the software until high quality is achieved. This will require minimum localization effort but increased architecture and coding efforts.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Scenario 2: New software is being developed and it must be available in more than 5 languages, the static content size (number of words) in the software is large (more than 5,000 words), and the software will be available for use commercially. Please rate the following guidelines.

Guideline 1: In commercial software, methods like crowd sourcing cannot be used and the translation quality must be high, which means automated machine translation services cannot be used and professional services must be utilized. If the project has a flexible budget, contract a professional Localization Vendor. Outsourcing all language related elements of the software will help focus on core functionality. This option is relatively costly. Some localization vendors include Rubic.com and ENLASO.com.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: Hire permanent translators for each language and translators with expertise in multiple languages are preferred. This will lead to increased personnel costs, but will ensure high quality and allow for a lot of flexibility since the translators will be available at all times in the company. Consider temporary contracts for short term projects.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 3: Use online portals to search and hire freelance translators. To improve understanding, fly in the different translators early in the project and agree on how fast translations will need to be submitted and in what format. This will be less costly than the other options, but the quality, speed and flexibility will be reduced.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Scenario 3: New software must be developed in only one language, with future possibility of adding new language versions. The static content size is small (less than 1,000 words), and the software will be available for use commercially. Please rate the following guidelines.

Guideline 1: Ensure the software is properly internationalized, meaning designed and implemented such that it can easily be localized without requiring changes in the source code. At this point, there is no need to hire translators or localization experts. In financial calculations, consider the cost of Localizations tools that need to be purchased to facilitate the internationalization. Similarly, the design and coding efforts will be increased.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: Only few target languages may be needed in the future. The static content size is small for this commercial software. Therefore, to ensure quality localization, hire professional translators or localization vendors in the future when the software needs to be localized.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

In Development Phase (Implementation Phase)

Scenario 1: A software system is being developed using Agile practices in Java. The project is ahead of schedule and more than 70% of the project is complete. The team decides the software needs to be localized in 4 languages. The static content size (size of text that needs to be translated) is large (more than 5,000 words). The system will be made available for free use.

Guideline 1: Although more than 70% of the project is complete, the software should be re factored so the architecture is internationalized. Java provides special classes for this purpose, which should be utilized.

Since the content size is large, crowd sourcing is highly recommended to translate the content. This will need extra development effort, but it's worthwhile because cost of translating large content in 4 languages will be minimized.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: Language and localization experts should be included in the project team in order to test the internationalization and subsequent localization. To save time, consider hiring freelance translators available online.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Scenario 2: A software system is being developed in PHP using a sequential methodology (like the Waterfall Process) and it is currently in the Requirements Engineering Phase. The project has a large budget and the customer wants this commercial product to be localized into 3 languages (English, Swedish and Arabic) and the static content size is small (less than a 1000 words). Please rate the following guidelines:

Guideline 1: During requirements triage and prioritization, modify the standard templates to include culture and locale specific requirements and effects. Ensure relevant sections of the requirements documents are available the languages suitable to the users (who will need to approve the requirements).

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: Hire localization experts familiar with the open source Gettext localization tool. Also, hire translators competent in the required languages. This should be done before the next phase. Ensure Gettext and other PHP language libraries such as FriBidi (for right to left languages like Arabic) and Intl to facilitate collation and management of numerical data.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Scenario 3: A software project team (from the IT department) is about to start the Architecture and Design phase for an in-house multilingual software project in a government office. The project budget is high, the system is critical and quality must be high across language versions. The specified requirements state the product must be easily localized without the need to change the source code in future. Please rate the following guidelines.

Guideline 1: Since the project budget is high and this is an in-house development, the professional services of localization vendors should be utilized. In other words, the aspects of the project that must be multilingual are outsourced to localization vendors. Check local companies or search the internet.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: The localization vendor will localize the system during implementation in the agreed languages, but to ensure future localizations can take place smoothly independent of the vendor, ensure administrative workflows are specified and tested. Technical details such as what resource files need to be translated should also be considered during training.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Scenario 4: A website is being developed for a multimedia firm using Agile implementation practices. Numerous graphics will be used and the website must be available in 2 languages. Both the static and dynamic content are available in both the languages. Please rate the following guidelines:

Guideline 1: Ensure no image has embedded text in it; use CSS and JavaScript to load Texts onto the images from the external resource files (containing strings for different languages).

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: During refactoring, remove all strings from the source code and put them in resource files. Depending on the selected language, the string should appear on the website, including images with embedded strings.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Post-release (Maintenance) Phase

Scenario 1: An existing software system currently available in only one language needs to be localized into another language. The source code and project developers are both accessible. Please rate the following guidelines:

Guideline 1: If project budget is high, the system should be internationalized such that future localizations are easily achievable without the need to modify the source code.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: Consider all technology relevant guidelines for internationalization and project relevant guidelines for localization (this thesis presents numerous guidelines categorically, some of which were presented in the previous scenarios).

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 3: If project budget is low, hire a translator and use a localization tool that can automatically localize the source code; Globalyzer is one such tool. This is a quick fix solution and one developer might be needed to fine tune the localized version. Freelance translator should be hired to verify the translated content.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 4: Use tools, like Gettext, to automate the extraction of locale-dependent strings from the Source Code. This must be used if the project source code is very large. This will also expedite the internationalization process and reduce costs.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Scenario 2: An existing legacy system needs to be localized into one additional language. The source code isn't available and neither is the project team. Please rate the following guidelines:

Guideline 1: If project budget permits, consider developing a new system from start. The analysis of the existing system and its documentation can expedite the development effort.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree

Guideline 2: If redevelopment is not possible, consider the possibility of localizing the system executables, since the source code is not available. For example, try using the method introduced by Daohe & Shusheng, where they converted an existing system using the Assembly Language Kit Masm32. This works on executables generated by VC++, Delphi, Visual Basic and C++.

□ Strongly Agree	□ Agree	□ Neutral	□ Disagree	□ Strongly Disagree