A New Hybrid Process for Software Development and Localisation

Mathurin Soh, Marcellin Nkenlifack, Laure P. Fotso

Abstract—In this paper, we propose a new approach to make it easier and safer localisation in the development of software systems. It allows describing the cultural elements and mechanisms to develop specific development process oriented models, adapted to the localisation. This approach relies on the identification of linguistic and cultural considerations in all stages of the development process takes place in two phases: The first phase called internationalization or pre-localisation is to describe a high level of abstraction different cultural aspects of the needs of both functional and non-functional. These descriptions are then analysed and instantiated in a second phase called post-localisation, to adapt the code to facilitate the construction and verification of software system adapted to the client's culture. This new hybrid development process can take into account the cultural aspects from the design and benefit of internationalization and localisation, in the phases of the software life cycle. We illustrate our approach with the design and development of a billing application. The results show that the new proposed approach is better than the localisation after software development. The proposed approach produces more flexible applications, adaptable and maintainable with relatively low costs and delays.

Index Terms— Culture Considerations, Internationalization, Localisation, Software Development, Software Localisation, Process

1 INTRODUCTION

THE evolution of the discipline of software engineering, comparing it with other older and more mature sciences (such as architecture) was studied in 1990; software engineering is still in its infancy as a science and discipline and must still evolve and to mature to perfect its development[1]. But much progress has been made in recent years, experience accumulates and it is time for a new era for software development, with production software adapted to the culture of the end user. In the production of the software for different markets, there is a need for localization. This latter (often abbreviated to "L10n") is defined by the Localisation Industry Standards Association (LISA) as the process of taking a product and making it linguistically and culturally appropriate to the target locale (country/region and language) where it will be used and sold (Esselink, 2000, p.3). This process needs to take into account the requirements of companies and target markets, generally encapsulated in cultural considerations results in a software product adaptation in several languages or for a country or region.

Many localisation efforts are met with frustration the customer once the software is produced. For example, text messages are altered, the fonts are not accurate or sometimes are truncated, encoding of exotic languages does not look right, and in general, the software products might not work as expected. A typical method to develop multiple versions of a local software application comprises two steps.

1. A step of internationalization during which qualified for developers to outsource any specific code elements to regions (e.g., unit measurements, date formats, specific data writing direction, and sometimes laws and policies) in resource files;
2. A step called localisation that allows software to transform the original resource file in local resource files for specific regions and cultural areas [2].

The localisation of a software project often covers a long period of time during the evolution of software [3]. Although culture has been recognized as one of the factors in the design of the interface, in computer science and in engineering in general, software products are often considered to be culturally neutral[4]. Culture is then an important factor in the design, modelling and theory. However, we believe it is impossible for designers, application developers to separate from their cultures, cultural issues are ubiquitous in computers, and they affect the data representation, basic design data communication protocols, software engineering methods [5]. It is equally important to recognize the contribution of technology and software products in the crop. Similarly, in the design and implementation of technology and software products, cultural characteristics such as language, beliefs, values, morals, must be recognized because they are involved in satisfying customer requirements.

In section 2, we recall some terms and definitions, and summarize the state of the art on current localisation practices. Section 3 describes the new software development approach, its features and benefits. Section 4 is dedicated to experimentation and evaluation. Section 5 concludes this paper.

- *Mathurin Soh is currently pursuing a PhD degree program in computer science at the University of Dschang, Cameroon, P.O. Box 67. E-mail: mathurinsoh@gmail.com ; mathurinsoh.soh@univ-dschang.org*
- *Marcellin Nkenlifack, University of Dschang, Cameroon. E-mail: marcellin.nkenlifack@univ-dschang.org*
- *Laure P. Fotso, University of Yaoundé 1, Cameroon E-mail: laurepfotso@yahoo.com*

2 STATE OF ART

In its current state, the Software Engineering is much turned

towards the production of software systems and does not sufficiently include their adaptations. Thus, it can rightly be called a 'Software Development Engineering'. Indeed, in current methodologies of software engineering, software regionalization considerations are not sufficiently taken into account. In the literature, there is no conventional method of localisation as it exists in the standard software engineering. The localisation engineering evokes expectations based on the standard software engineering. However, software localisation although it requires adaptation, also happens to be full production business software.

Currently, the translation of the interface elements is entrusted only to professionals, making along the translation process, expensive and of poor quality [6]. As a result, software localisation deserves to be as structured and properly conducted by a range of methods, approaches and best practices. Software localisation passes also through successive stages as software development, from its conception to its death, and also therefore a life cycle. Whatever type designed and developed software, either a thin client on a desktop computer, a cloud service, a mobile applications or a website, the possible need for its localisation should be assessed from the very beginning even in the specifications or in the development process. Indeed, aspects of software localisation exist throughout the same process of development of this software, even before its localisation. In the figure 1, we highlight different phases of software development where cultural concerns can appear. These aspects of localisation are embedded in the steps of classical software development tasks.

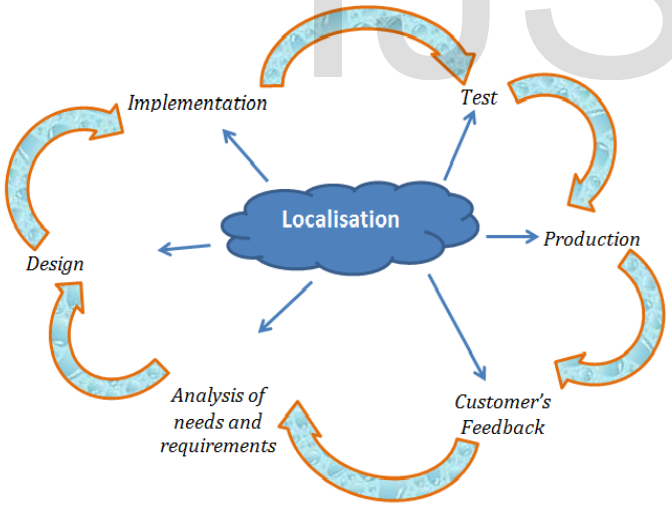

Fig. 1. Steps identified for the consideration of cultural aspects in software processes development

As shown in [7], in various situations, many software applications are not internationalized in the early stages of development. To internationalize such existing applications, developers need to outsource some constant strings "hard coded" into the resource files, so that translators can easily translate these applications into a local language without changing the source code. This internationalization of such applications has a cost and takes time. Although the results of [7] applied by [8] present an automatic approach of determination of elements to be translated into an application, and requiring thereby localisation for internationalization needs, the developer would save time thinking about the localisation aspects from the beginning of the aforesaid software development process. This could be avoided by following a rigorous localisation method.

Zeyad et al shows that just 38% of software systems are localizable, and mostly including web applications are localized [3]. If the cultural deaspects were considered, this would not only reduce the cost of software internationalization, but also the cost of localisation in one language or specific culture. For every successful localisation project software requires a step of Internationalization of the Software. This last step is to make a separation between the business code and parameters related to a region or a culture or a language. Language and cultural considerations are taken into account in basic activities covered in all models: the analysis; the design, the implementation, the validation (especially in the testing, the integration), the evolution (especially the maintenance). In these conditions, developers for example would not be facing any programming problems sometimes difficult for some languages.

According to Friedel Wolff [9], software development companies often think software localisation when trying to attract new customers in a country where they were not yet established. The localisation is thus a huge industry. Some engineering practices lead to a misconception about the localisation, stating that it has to come at the end of the project and be treated separately, or be partitioned from the rest of the development. This attitude leads to poor localisations and drastically increases the overall cost [10] of the software. The localisation of open-source software is of relatively poor quality [3]. We believe that the use of a suited rigorous methodology accompanied by collaboration between the stakeholders would improve the quality of software localisation.

3 IDENTIFICATION OF CULTURAL CONSIDERATIONS IN THE SOFTWARE DEVELOPMENT PROCESS

Software development requires an orderly sequence of execution of activities including course aims to achieve a given objective. The challenges that everyone faces with some languages (such as Brazilian Portuguese) do not relate to internationalization, localisation of the software being developed. These are very important accented characters in some languages; length of words which when developing software, the programmer must not forget to put more width to the labels for any text expansions/contractions; date formats and non-uniformly recognized hour in all languages and cultures; measurement units, some languages and cultures are different from a large number of units used internationally; keyboard used with grapheme are not the same for all languages; currencies and currency; the addresses are not uniform in all regions, languages and cultures; electronic payments; import duties are important for e-commerce applications; all languages of the world do not have the same writing direction. Some languages like Yemba in the West Region of Cameroon and Latin languages are written from left to right. On the other hand, the languages such as Arabic are written from right to left. The software should take into account its operations in the

various language areas; the writing system, some languages like Chinese have more than one; keyboard shortcuts; colors and textures are very important in the localisation as different colors and icons have also distinct meanings in different regions; signs, icons and symbols; sounds.

The main steps of the software processes have been defined by standards ("IEEE Software Engineering Standards Collection" and the norm "AFNOR Z67-150"). These steps are respectively the definition of software targets the expression of needs, the design, the coding, the testing, the production, the installation and the maintenance operation and the customer feedback. The success of a software development project also depends on the final product adaptability [5] considered in all previous steps. This is the idea of globalization / internationalization. Items that should be translated include constant strings, date / time items, number format objects, the culture-related items. In particular, the determination of these elements to be localised is often the most time consuming task[5]. Thus, the software localisation engineering is an essential complement to software localisation cycles and even to software testing. It covers all the critical steps for successful localisation and testing, including analysis software, the construction and mastering of engineering, fixing errors, and automation of the overall localisation process. In pursuing this demonstration, we realize that this is true for all life cycles of software. These life cycles do not consider sufficiently aspects of their localisation in different stages. The identification and isolation of the various localisation requirements at each phase of the development process result in the Figure 2 below.

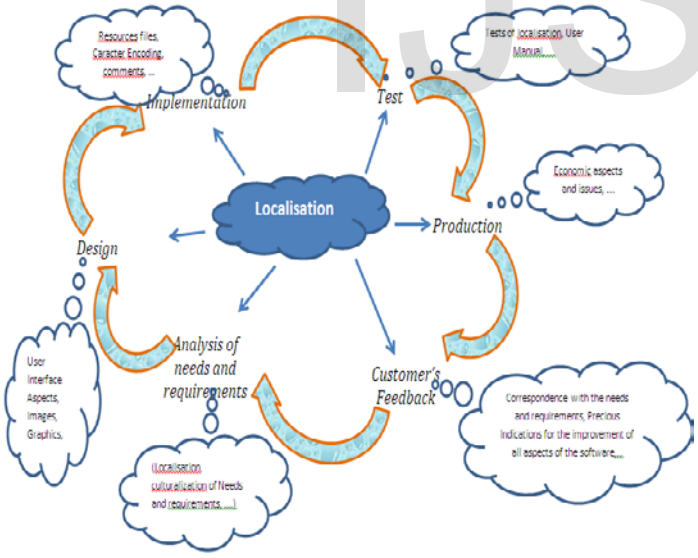

Fig. 2 Steps and related cultural issues throughout the software development process

It appears that each phase of the software systems development process is subject to issues of localisation and internationalization. For all the above remarks, we question the development of specific Localisation Software Engineering. This activity aims at reducing costs, improving quality in localising and shortening delays in the localisation process.

4. A NEW SOFTWARE DEVELOPMENT APPROACH

A process is "a set of interrelated or interacting activities which transforms input elements into outputs"(ISO 9000: 2000). This definition perfectly applies to the development / localisation applications from their gestations, their births, their growths, their developments, their deaths ... and to rebirths. This sequence of stages of life corresponds to the different steps defined in a software process. The objective of such a division is to define intermediate milestones for validation of software development. That is to say the conformity of the software with the needs expressed and the verification of the development process that is the appropriateness of the methods implemented. The software life cycle must go through the steps from the stage of expression of needs for establishment of such software until its disappearance. It is thus normal to consider more generally, the localisation needs from the needs and requirements analysis till the maintenance of the software to produce. The internationalization is done transversely during all the activities of the development process. It leads to the actual localisation from the beginning of the process. In this sense, it is assimilated to a pre-localisation phase. Schematically, the process is as shown in Figure 3 below.

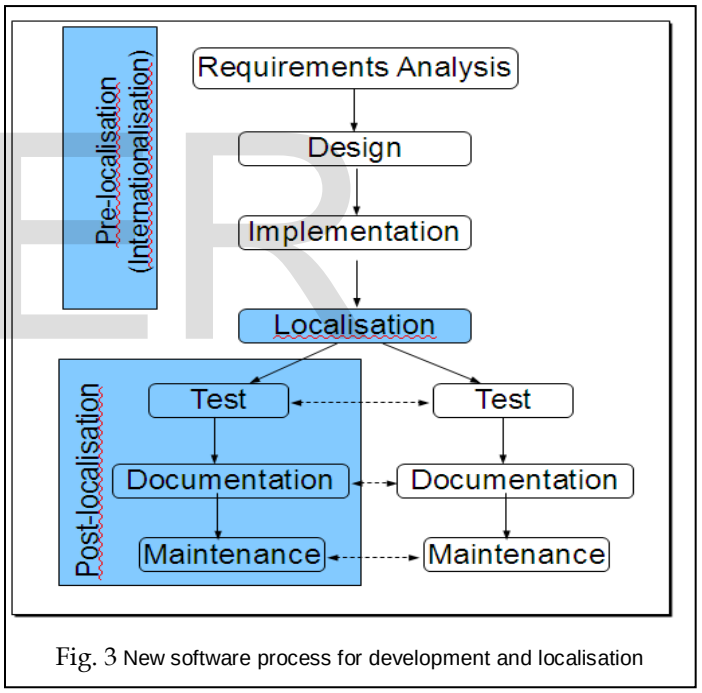

Fig. 3 New software process for development and localisation

In this scheme, the following activities are presented:

(A) Requirements analysis: In addition to the analysis of functional and non-functional needs, an analysis of localisation requirements is needed. It is done by identifying the features that will be subject to the localisation. The impact of these requirements has many challenges that arise throughout the development life cycle and software localisation [11].

(B) Design: This phase must integrate the design of internationalization that is, e.g set the text size to be internationalized, icons, parts of the user interface.

(C) Implementation: During the implementation, the internationalization must be made for example in the use of global variables, internationalization files, resource files, directories

dedicated to internationalization.

(D) Localisation: After cross internationalization in the previous steps, follows the localization, the "production" of the software for a given language and culture. At this point, we can assume that the initial development is done in a language and driver for the software in that language, localisation step is not necessary because the software is initially produced in that language. But for any other language other than the "pilot language," we must do the locating step of adapting the software as localisation requirements initially analysed (for the analysis stage), designed to step in design and implemented during the implementation step.

(E) Test: In addition to the regular development process tests, we also need the localisation of that test would be to reassure the needs that had to be localized are effectively.

(F) Documentation: Documentation of the localisation is not just a step. It is conducted transversely since the needs analysis.

(G) Maintenance: Maintenance of localisation is to change everything that is related to the localisation.

In Figure 3, the dotted lines between steps after localisation just indicate that regular activity can generate the corresponding localisation operation. For example, if maintenance of a feature is triggered, it can also require maintenance of the corresponding localisation.

5. ILLUSTRATION AND EXEPRIMENTATION

5.1 Development of a billing application by the classical approach

A) Analysis phase and needs collection

To establish an invoice, the application must take into account information that will be provided by the user. A file representing such invoice will be generated and the user can print it. Each invoice is stored (it is actually information about the bill) and the user can consult and print as many times as he wishes.

- **Identification of actors, use cases**: The actors are individuals who interact with it. In our case the only actor is the cashier. This is the person who is responsible to collect the money and hand back a bill to the client. The different features that can trigger this actor in the billing system are: Establish an invoice; Consult a bill as shown in Figure 4 (a).

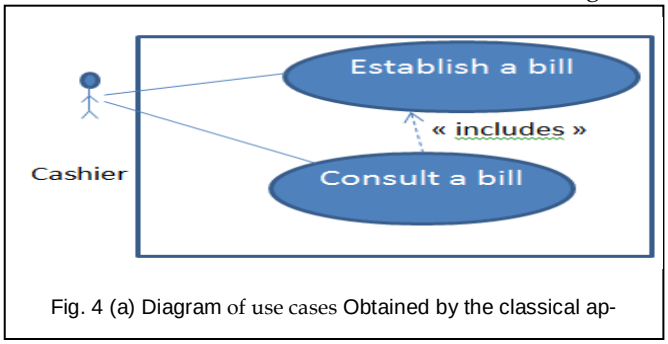

Fig. 4 (a) Diagram of use cases Obtained by the classical ap-

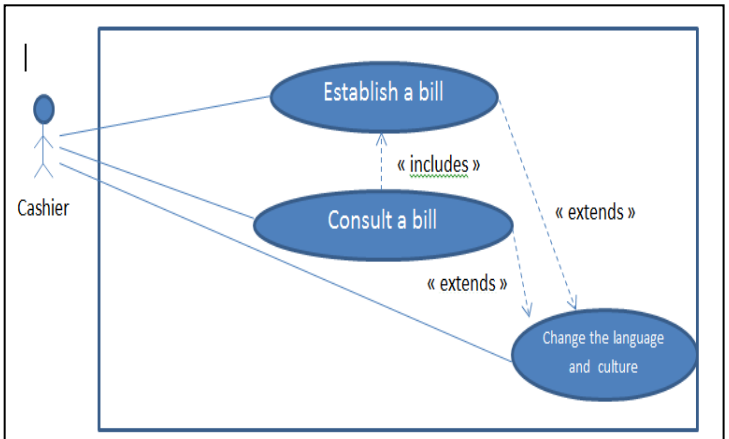

Fig. 4 (b) Diagram of use cases Obtained by the new approach

- **Diagram of classes:** The diagram shown in Figure 5 identifies the system entities and their relationships with each other. The different classes identified are: Product, Customer, and Bill. The only relationship between a customer and a product is that a customer can buy a product. From this relation, result a class called "class association", the Invoice class.

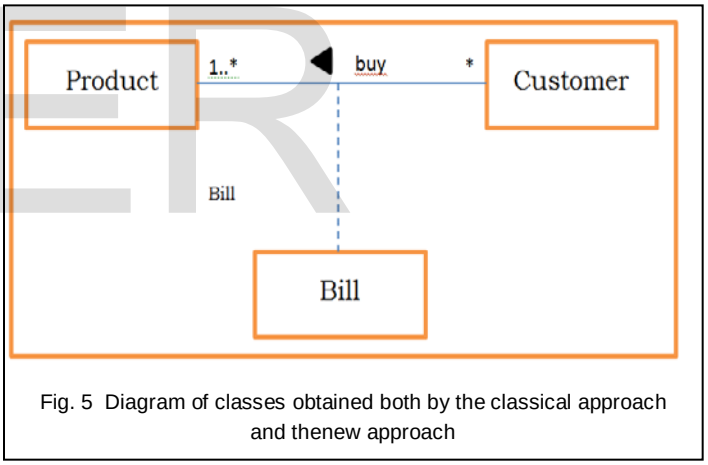

Fig. 5 Diagram of classes obtained both by the classical approach and thenew approach

- **Diagram of interactions with the system:** The previous diagram of classes, does not describe the interactions between the various entities of the system. To consider this important aspect, we develop better interaction diagrams describing the exchanges during the realization of use cases.

IJSER

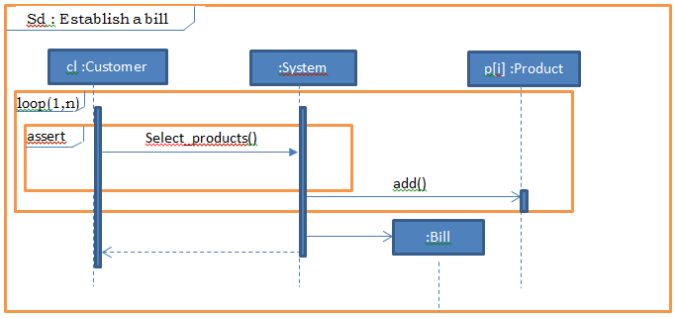

Fig. 5 Diagram of interaction for the Etablishment of a bill

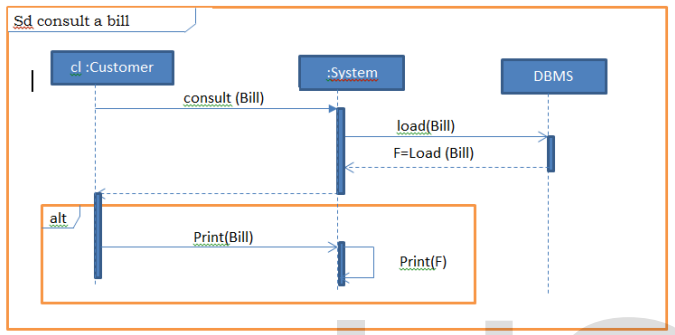

Fig. 6 Diagram of interaction for the consultation of a bill

B) Implementation phase

The implementation is done using the Java language through the Integrated Development Environment (IDE) NetBeans 8.1. The database is developed with the tool PostgreSQL 9.4. Figure 7 below shows an interface of the resulting program.

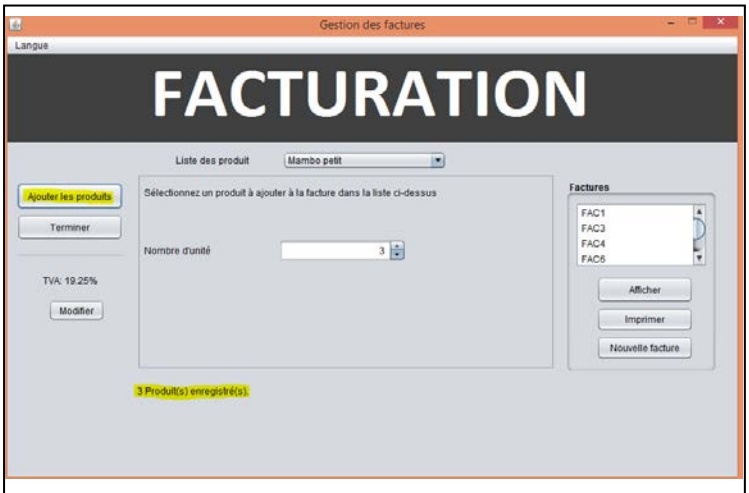

Fig. 7: Product Selection Interface to charge

C) Test phase

The use of this program consists of a selection of products chosen by a buyer. This action is entitled to a product registration in the database, as shown in Figure 7. The invoice is generated after validation of products chosen by the buyer as shown in Figure 8 below.

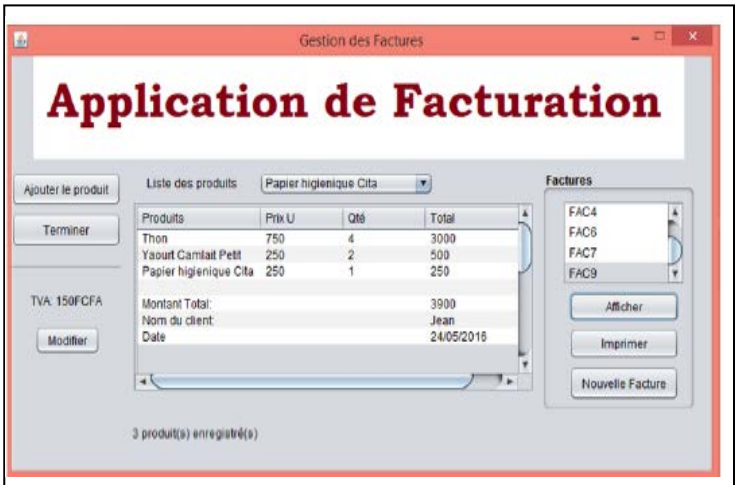

Fig.8: Application interface (Invoice established)

5.2 Development of a billing program by the new proposed approach

A) Analysis and needs collection

The application must allow to take into account information that will be provided by the user. These include products chosen by a purchaser and their quantities. An invoice will be generated and the user can print it. Each invoice is stored and the user can consult and print as many times as possible. However, the application must be fully functional in the Yemba cultural area. The application will thus have the opportunity to be transfer from one language to another and will integrate the Cameroonian languages. End users will be able to change the user language by selecting one that suits them. The target language and pilot chosen for this illustration is the Yemba language in the Menoua Division, West Region of Cameroon.

B) Design phase

• **Identification of actors, use cases:** The only actor in the system is the cashier. This is the person who is responsible to collect money and to return an invoice to the customer. The different features that can trigger this actor in the billing system are: Establish an invoice; Consult a bill and change the language. These use cases are given in the uses cases diagram of the previous Figure 4 (b).

• **Diagram of classes:** The diagram shown in Figure 5 let to identify system entities and their relationships. The different classes that we identified are: Product, Customer, and Invoice. The only relationship between a customer and a product is that a customer can buy a product. From this relation, we have a "class association", named Invoice class.

• **Diagram of interactions with the system:** The previous diagram of classes, does not describe the interactions between the various entities of the system. To account for this important aspect, we develop better interaction diagrams describing the exchanges during the realization of use cases.

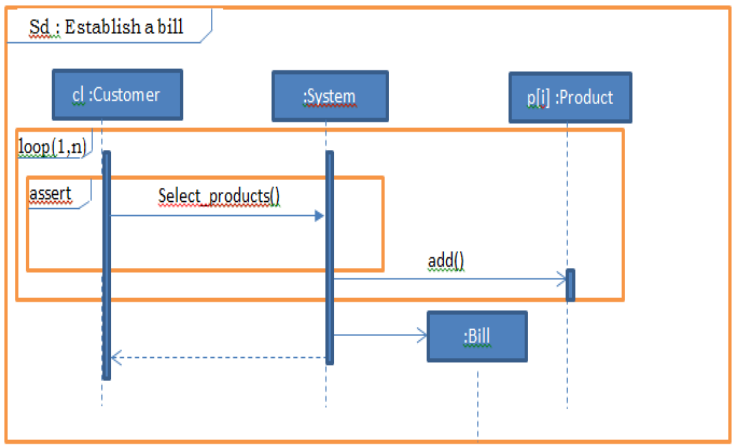

Fig. 9(a): Interactions for the Etablishment of a bill

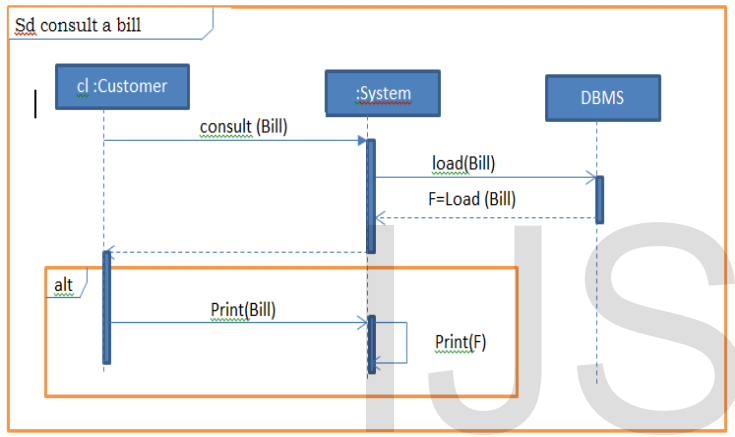

Fig.9(b): Interactions for the Consultation of a bill

C) Implementation Phase

The application obtained after implementation with the Java language through the Integrated Development Environment (IDE NetBeans 8.1) is interfaced by the figure 11. The database is developed with PostgreSQL 9.4. The products' names in the database are not localised into the target language, because as suggested by [9] most cases, product and brand names should be leave unchanged during localization.

An XML file referred to as fr_lang.xml related to interfaces of resources is generated at the end of the transverse internationalization phase. Its configuration is as shown in Figure 10 (a). In this file, each user interface element is described by an XML tag that can have two sub tags. The first sub-tag <key> has the key to the cultural element, and represents a kind of variable. The second sub tag <content> is the value of the cultural element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- To change this license header,
choose License Headers in Project
Properties. To change this template file,
choose  Tools | Templates and open the
template in the editor.
-->
<root>
    <traduct>
        <key>baniere</key>
        <content>FACTURATION</content>
    </traduct>
    <traduct>
        <key>jMenu1</key>
        <content>Langue</content>
    </traduct>
    <traduct>
        <key>radioFR</key>
        <content>Français</content>
    </traduct>
    <traduct>
        <key>radioYE</key>
        <content>Yemba</content>
    </traduct>
```

Fig. 10: An extract part from the XML Resources File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- To change this license header, choose
 License Headers in Project Properties.
To change this template file, choose Tools
| Templates and open the template in the editor.
-->
<root>
    <traduct>
        <key>baniere</key>
        <content>Apʉ'te nzwiŋ ŋka¯p</content>
    </traduct>
    <traduct>
        <key>jMenu1</key>
        <content>Ashuŋne</content>
    </traduct>
    <traduct>
        <key>radioFR</key>
        <content>Afla´ŋsi</content>
    </traduct>
    <traduct>
        <key>radioYE</key>
        <content>Yémba</content>
    </traduct>
    <traduct>
        <key>ajouterP</key>
        <content>N´tswi¯'ne¯ tsitsi</content>
    </traduct>
```

Fig. 10(b): XML file for User Interface resources in Yemba

IJSER

Fig 11: Billing program User Interface obtained by the new approach

D) Localisation phase

During this phase, we adapt as much as possible the target culture elements in the resources localisation file, named fr_lang.xml and generated after the transverse phase of internationalization. As well generic file, we search all the entries corresponding to the input cultural elements (French). Then, the target cultural elements (Yemba) are replaced in the generic resource file. The localized file is shown in Figure 10 (b). The resulting Billing program has an User Interface shown is figure 11.

E) Test phase

This test phase is to ensure that all textual elements have been translated and all other parameters have been localised in Yemba, the target language and culture.

F) Documentation phase

For the documentation, it is aimed at producing a user manual.

(G) Maintence and Customer's feedback phase

During this phase, the customer reviews the program to ensure that his both functional and non-functional requirements are met.

5.3 Evaluation and Summary of Findings

Our assessment is made on the basis of the criteria and the usual evaluation software process metrics, including design time and development costs and the quality of development / localisation. According to [11] and [12], the cost, time and quality are the main difficulties in software localisation. The costs indicate the direct and indirect expenditures during the localisation process. The time references the time of producing a program through the localisation. Quality refers to the completeness, relevance, understand ability of localisation. Table 1 below summarizes the assessment.

TABLE 1
EVALUATION OF COSTS IN THE DEVELOPMENT OF A BILLING PROGRAM WITH CLASSICAL AND NEW PROPOSED PROCESS

Phase	Steps	Time (in hours)	
		Classical Approach	New approach
Pre-localisation (Internationalisation)	Requirements analysis	4	5
	Design	10	15
	Implementation	15	12
Localisation	Localisation	10	3
Post-localisation	Test	4	5
	Production	1	1
	Documentation	2	2
	Maintenance	2	2
	Customer's Feedback	1	1
Total		49	46

This study found that the full development of a billing program with the classical approach has a duration of 49 hours, and the same work will last 46 hours with the new proposed approach. According to this findings, the localization after system development of the billing program take more time than using the new proposed approach for the work. This shows that the integration of localisation into the development will be less time-consuming than the localisation after development of software. In this study, we investigated whether the integration of localisation into the development of software is better than the localisation after system development. Our results show that the dissociated approach is nearly less time consuming during the steps before the localisation, than the integrated one. But after, it is the contrary just because of the actual localization step. And globally, it is the integrated approach which has a smaller duration.

Using the results of the comparison of Agile software processes and classical software processes, done by [13], we presents in Table 2, a comparison between the proposed new software approach and the others.

IJSER

TABLE 2
COMPARISON OF THE NEW SOFTWARE PROCESS FOR DEVELOPMENT AND LOCALISATION WITH CONVENTIONAL SOFTWARE PROCESSES

Criterion	Classical Software Processes	Agile Processes	New proposed approach
Request for Changes at any time during development of product	Here change request is always rejected throughout development.	Changes are acceptable at any time during development.	Request for Changes can be accepted at any time during development and localization of product because of a cross-cutting internationalisation
Delivery of product in time/on time/early	Usually, deadlines are not meet and mostly impossible to deliver product before estimated deadline.	Delivery of product is as per estimated deadlines i.e. always delivered in or on time.	Product are delivered on time
Quality of product is a major concern	In Waterfall model quality of product is not as desired by customer, because if user want some other changes then it is not possible in one go (during development time)	Quality is built-in; delivered product always satisfies the requirement or need of customer	The quality of product and of localization is a major concern[12]
Involvement of customer throughout development	After submitting the requirements in 1st phase, customer gets involved only on delivery of product.	Customer must be present at each and every phase of development.	The customer may always be present in each of the development phases, if the approach is combined with a collaborative approach to Agile location
Requirements	This model is used, if requirements of customer are clear and well defined	Agile model is used if requirements of customer are not clear or changes frequently.	Localisation requirements of customer are clear and well defined
Pattern	Waterfall model is a sequential model, means phases are always followed in consecutive manner.	As change occur frequently, so we can revisit any phase at any time.	The approach to development / localisation can be combined with other traditional or agile approaches
Development time	Development life cycle is longer as compare to agile model.	If requirements are not so clear, are gathered on daily basis, then adopting agile makes sense.	Development / localisation time are shorter compared to the localisation of a software after the development by any other method[11]
Risk factor	There is a lot of risk of not meeting customer's requirement	Risk is less in Agile development because customer is involved in each and every phase of development.	The risk is moderated for cultural requirements if the localization is collaborative
Problem identification	Late identification of problems	Progressive identification of problems	The identification of problems is combined with an Agile method

IJSER

In the table 2, using selected criterion, we investigated whether the integration of localisation into the development of software is better than the localisation after system development. The results also emphasize that the new proposed approach is better than the localisation after development, even if the software is analysed and produced through using the classical or the agile software processes.

6 CONCLUSION

We have presented a new approach to take into account cultural considerations in the development and localisation of software systems. This approach relies on the identification of cultural needs in all steps of development since the expression and requirements analysis. First, it describes at a high level of abstraction, different cultural aspects, both functional and non-functional, of a system in the specification phase and architecture. These descriptions are then analysed and grouped in a resource file to facilitate the localisation, which is followed by a post-localisation phase. We illustrated our approach by designing and developing a billing application by the conventional approach and by the proposed new approach. It follows that the latter gives better results as regards the localisation of the application cost, the time taken in this localisation as well as a better quality of localisation. It can produce applications, with a relatively low costs and it is less time consuming.

REFERENCES

[1] Mary Shaw : Prospects for an engineering discipline of software ; IEEE Software, novembre1990, pp. 15-24.

[2] Bert Esselink. A Practical Guide to Localization. Amsterdam Philadelphia, 2000.

[3] Zeyad Alshaikh, Shaikh Mostafa, Xiaoyin Wang, and Sen He. A empirical study on the status of software localization in open source projects. Pages 692_695. Proceedings of The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015, 2015.

[4] Tedre, M., Sutinen, E., Kahkonen E., and Kommers, P. 2006. Ethnocomputing: ICT in cultural and social context. Communications of the ACM, 49 (1). pp. 126-130. ISSN 0001-0782

[5] Keller, B., Pérez-Quiñones, M., and Vatrapu, R. (2006), "Cultural

issues and opportunities in Computing Education", 9th International Conference on Engineering Education, R1E-14, July 23-28, 2006.

[6] A. Fraisse, C. Boitet, H. Blanchon and V. Bellynck. A Solution for in Context and Collaborative Localization of Most Commercial and Free Software. In Proceedings of LTC 2009 the 4th Language and Technology Conference, vol. 1/1 : pp. 536-540. November 6-8, 2009. Poznan, Poland.

[7] Xiaoyin Wang, Lu Zhang, Tao Xie, Hong Mei, and Jiasu Sun. Locating needto-translate constant strings for software internationalization. pages 353_363. International Conference on Software Engineering (ICSE), 2009, 2009.

[8] Xiaoyin Wang, Lu Zhang, Tao Xie, Hong Mei, and Jiasu Sun. Transtrl : An automatic need-to-translate constant string locator for software internationalization. pages 555_558. International Conference on Software Engineering (ICSE), Tool Demo, 2009, 2009.

[9] Wolff, F. 2011. Effecting change through localisation: Localisation guide for free and open source software, IDRC, Canada.

[10] Simon Hill. What is localization and why should i care ? http://localizedirect.com/posts/what-is-localization/, consulté le 30 mai 2016.

[11] Jesus Cardenosa, Carolina Gallardo, and Alvaro Martin. A practical case of software localization after system development. International Journal of Information Technologies and Knowledge, 1:121-127, 2007.

[12] Malte Ressin, José Abdelnour-Nocera, Stephen Roberts, Empirically researching development of international software, Proceedings of International Conference on Software Engineering, June 2012

[13] Pankaj Vohra and Ashima Singh. A contrast and comparison of modern software process models. pages 23-27. International Conference on Advances in Management and Technology (iCAMT - 2013), Proceedings published in International Journal of Computer Applications R (IJCA) (0975 _ 8887), 2013.

[14] A. Fraisse. Localisation interne et en contexte des logiciels commerciaux et libres par des utilisateurs finaux. In Proceedings of Infol@ngues III 2009, NTICS, Langues et Humanités : Réalités et Perspectives. February 6-7, 2009. Tabarka, Tunisie.

IJSER