SOFTWARE AND CULTURE: BEYOND THE INTERNATIONALIZATION OF THE INTERFACE

Gregory E. Kersten[1], Mik A. Kersten[2] and Wojciech M. Rakowski[3]

[1] DSMIS, Concordia University, Montreal, Canada (gregory@mercato.concordia.ca).
[2] Xerox PARC, Palo Alto, USA (mkersten@parc.xerox.com).
[3] ANS, Bell Canada, Toronto, Canada (voitek.rakovski@bell.ca).

Abstract: Software applications are designed around user interaction. One interaction component is the user interface; the other deeper components represent the applications' logic and core functionality. Internationalization architectures recognize the need for localizing user interfaces to particular cultures. We continue the discussion on culture and software focusing on the software core rather than the user interface. This core corresponds to deep culture as opposed to the surface cultural manifestations embedded in the user interface. We argue here that deep culture can be embedded into application software in a modular way.

INTRODUCTION

The participants of the 1999 E-Conomy conference called for a pluralistic perspective on e-commerce and its technologies (Cioffi 1999). Technologies developed for e-commerce have a number of popular applications, including communication and discussions, decision-making and negotiation, voting and other forms of facilitating citizens' to participate in federal and local governments. A pluralistic perspective on e-commerce and other computing technologies requires the consideration of social and organizational cultures, including value systems, beliefs and norms. The state of application software today seems to indicate that U.S. technologists and, so called, software evangelists still determine the user experience based on their own cultural biases.

To date, the development of software for a culture different from the culture of its authors has been focused on adapting the user interface. This method is referred to as *software internationalization*. The underlying assumption behind software internationalization is that all of the culturally and linguistically sensitive software components can be separated from the locale-independent core of the application (Hall 1999, p. 298), (Nakakoji 1996; Hall and Hudson 1997).

The assumption of the culture-dependent interface and the culture-independent core has helped software companies to develop programs for international markets without re-writing the very same application for every new national market. The perspective that *all cultural aspects are encapsulated in the external layer of the software* has been fundamental in porting application to international markets. We will argue with this traditional engineering approach (Sommerville 1992), which (1) separates the human interface from the mechanics of the apparatus, and (2) assumes that changing the interface is all that is necessary to change the usability of that apparatus (Laurel 1991).

At the early stage of the innovation curve simplification and reductionism are often necessary to understand complex situations, construct models and build machines. The ability to match a problem's complexity with its representation and machinal embodiment increases with the discovery of methods and techniques that are specialized for a particular type of innovation. The discussion presented here is based on the assumption that the current state of the software development processes allows for a richer perspective on the culture-software relationship than that being employed today.

In this paper we present arguments behind the claim that cultural concerns penetrate beyond the user interface. Contrary to what the current methods of engineering of international software would suggest, we consider *the software core being culture-dependent*. There are two interconnected motivations behind this statement: (1) the software core is a technology which, according to some theories of technology, is rooted in, and shaped by, culture (Heidegger 1977; Feenberg 1991; Ferre 1995); and (2) and unlike many other technologies software describes and automates complex activities and whole processes that previously were undertaken by people and organizations. The core of a software artefact embeds decision-making, rules of behaviour and patterns of actions that depend on culture (Juustila 1995; Kaplan 1995; Hofstede 1997). Consequently, embedding the attributes of the users' culture requires changes to the design of software architectures that go beyond the current international standards for software architecture and localization.

This paper continues the discussion on culture and software and the dominant role of Western cultures, the US in particular, in software development (Taylor 1992; Juustila 1995; Kaplan 1995; Nakakoji 1996; Carmel 1997; Kersten, Matwin et al. 2000). Two perspectives on culture presented in Section 2, are followed, in Section 3, by three theories of technology. In Section 4 perspectives on culture are coupled with theories of technology. This theory-based assessment is illustrated in Section 5 with three examples of culture being embodied in the software core. In Section 6 a proposal for software "culturalization" is formulated. It is based on the recognition that deep cultures can be embedded in software. This should happen, as we conclude in Section 7, even if one assumes that the impact of national cultures diminishes.

CULTURE

Culture provides subjective insights and relative laws; it allows for the interpretation of actions and events. In explaining culture Hofstede uses the analogy of computers and programming and says that “Culture is … the collective programming of the mind that distinguishes the members of one group or category of people from another." (Hofstede 1997, p. 5). As such, it is a set of shared and enduring meanings, values, and beliefs that characterize national, ethnic, or other groups, and orient their behaviour (Faure and Rubin 1993).

Culture is a world of symbols constructed by people; it is a structure of meanings, beliefs and values that condition human behaviour allowing for its interpretation and purposefulness. The key issue for our discussion is whether there is an underlying universal basis in which different cultures are rooted, and whether there are general cross-cultural laws and archetypes of culture. There have been two opposite perspectives on culture and its laws.

1. Holistic perspective posits that there is neither a universal culture nor universal laws. The set of symbols unique for a given culture cannot be detached and interpreted as an instance of a culture-free biological and economic basis.
2. Reductionist perspective views culture as a symbolic discourse. Culture is a language and other symbols that can be interpreted as language; there are universal laws on the creation of the sets, schemas and networks for symbol manipulation.

The holistic and particular perspective distinguishes between *deep culture* and *surface culture*. Deep culture includes beliefs, ideas, language, rules, knowledge, procedures and norms. It manifests itself in symbols, artefacts and objects ranging from art to organizational structures to products and services, all of which are elements of the surface culture. The meaning of the symbols and artefacts is defined by the deep culture and so any separation of the surface causes a loss of the intended meaning. As a result the understanding of the values that underlie any particular cultural manifestation is necessary for the interpretation of this manifestation within the culture that it was created by. Similarly, if we want to create manifestation for a “foreign” culture we need to understand its underlying values.

The reductionist and generalizing perspective is rooted in the empiricist and utilitarian American sociology and anthropology (Kuper 1999). Initially, it was oriented towards discovering general laws of language and structures. Limited progress turned the proponents’ attention towards lower level constructs, including brain models, knowledge schemata and neural networks. There is no need to differentiate between deep and surface culture because there is an assumed meta-set of symbols and universal laws which can be used to construct a culture.

TECHNOLOGY

Technology is knowledge embedded in products and processes, and it is also the products and processes themselves. Hart-Davidson (1997) defines technology as the set of artefacts and the sets of cultural beliefs, practices, and texts that surround the production, use, distribution, and conceptualizations of those artefacts, designed to produce some cultural condition. The relationship between culture, society and technology has been studied within the philosophy of technology. The following three perspectives on technology have been proposed:

1. Instrumental perspective argues that technology is neutral and indifferent to the variety of ends towards which it can be employed.
2. Substantive perspective argues that technology constitutes a new type of cultural system that restructures the entire social world as an object of control.
3. Critical perspective suggests that technology is a rational process of development that is neutral *per se* but becomes value- and ideology-laden in the design, implementation and use of technical systems.

The instrumental position on technology assumes its universally rational character, employment of a common standard of measurement (e.g., efficiency), which is independent of producers, users and situations. Pacey argues that technology proceeds separately from cultures, values and societies (Pacey 1992). This perspective has been characterized as uncritically positive and self-limiting. It has been contrasted with a substantive theory according to which technology constitutes a new type of cultural system that restructures the entire social world.

In the view of substantive theory, computer technology is seen as the culmination of a variety of cultural and ideological forces which, depending on the context in which the technology is used can be either positive or negative. The substantive position is interesting because it attempts to integrate both negative and positive viewpoints on technology. It suggests that technology has become the defining characteristic of all modern societies regardless of political ideology. It is autonomous and—as Heidegger argues—it is relentlessly overtaking us (Heidegger 1977, p. 17).

Feenberg advocates a critical theory in which technology is a process of development suspended between different possibilities—a process in which social values and ideas are attributed in the design and development, and not merely the use of technical systems (Feenberg 1991, p. 14). The two tenets of the critical theory are that (1) technology may be used to advance and enrich social objectives, and (2) technology cannot be seen as separate from people.

We can now observe that each of the three theories of technology leads to different perspectives on the design and development of software technology.

1. Instrumental perspective posits that system design and development can be done in isolation of the users and their situation.
2. Substantive perspective posits that system design and development can be for the betterment of the users leading to new social and cultural values and systems.
3. Critical perspective posits that system design and development is never neutral and can be used to propagate and infirm social and cultural values and systems.

SOFTWARE TECHNOLOGY

Reductionist Culture and Instrumental Technology Perspectives

There are several possibilities in studying the relationship between culture and software that can be based on different perspectives on culture and technology. The viewpoint of most software designers and engineers can be tracked to both (1) the reductionist understanding of culture as language and other symbols, and (2) the instrumental theory of technology. In this context software is culturally neutral and can be adapted to every culture through the modification of its user interface. The modification, known as a locale, is then defined solely in terms of the requirements regarding the interface.

Consequently, the collection of symbols and conventions that characterize a particular culture or user community (e.g. transliteration, hyphenation, spelling, numbers, currency, time and date, colour) is perceived as being sufficient to adapt software to the given cultural context (Taylor 1992; Sauter 1997). This perception has resulted in the existing internationalization methods used for translating software from its source market to the target markets. Current internationalization methods enable the localization of user interface elements. These methods, while based on three presuppositions, namely, (1) the choice of character codes, (2) the use of locales, and (3) the use of resource files, led to the software internationalization architectures (Hall and Hudson 1997).

The software internationalization architecture that is rooted in the reductionist perspective on culture and the instrumental theory of technology is equivalent to the user interface localization architecture. An example of user interface localization is given in Figure 1. Comparative studies help determine the set of primitives that are common to all cultures in which localization is considered. Primitives which are specific to a given culture (e.g., characters), interpretations of common primitives (e.g., of different colours and icons) and other constructs comprise this culture's locale. These primitives are used to define a common graphical user interface (GUI) toolkits.

The abstract concept of the interface, indicated in Figure 1 as Text Window <<abstract>>, is a generalization of all possible user interfaces. To avoid extensive studies of different cultures, a reductionist approach is used whereby the text window is developed in Culture A, and the abstract text window is generalized by ensuring that it can support other cultures' locale. In particular, Culture B Text Window is obtained through the replacement of components of the Culture A Locale with components of the Culture B Locale.

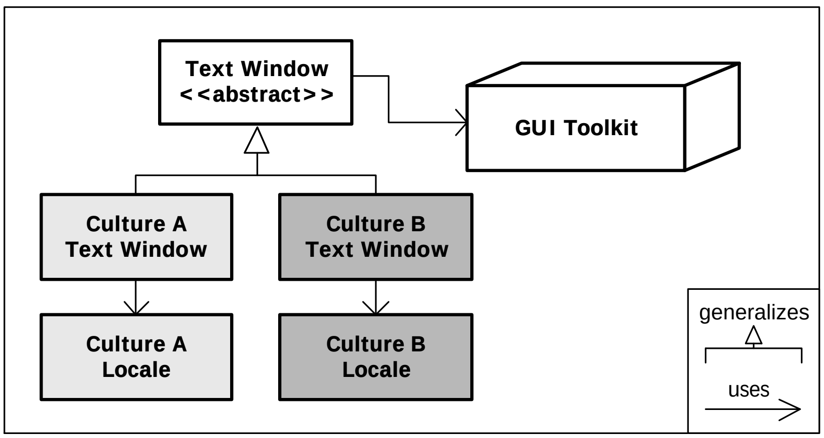

Figure 1. User interface localization.

Holistic Culture, and Substantive or Critical Technology Perspectives

In contrast to the above perspective on software internationalization we propose to consider (1) the holistic understanding of culture, and (2) the substantive and/or critical theory of technology. From the holistic perspective of culture and one of the two theories of technology it clearly follows that software technologies are not culturally neutral. Software internationalization produces applications whose user interface is internationalized and whose core is not.

In Figure 2 we illustrate the results of software internationalization. Software X developed in culture A is an artefact of Culture A. Cultural elements such as language, graphical symbols and conventions are used in the design of the software's user interface. Because of the internationalization requirements the interface is loosely coupled with the core. Loose coupling (indicated in Figure 2 with the uses arrow between software interface and core) allows for the replacement of an interface localized for culture A with the interface localized for culture B. This is an important distinction in the relationship between the *user interface and the application core*, paralleled by the relationship between *surface and deep culture*. The latter conjunction is tightly coupled.

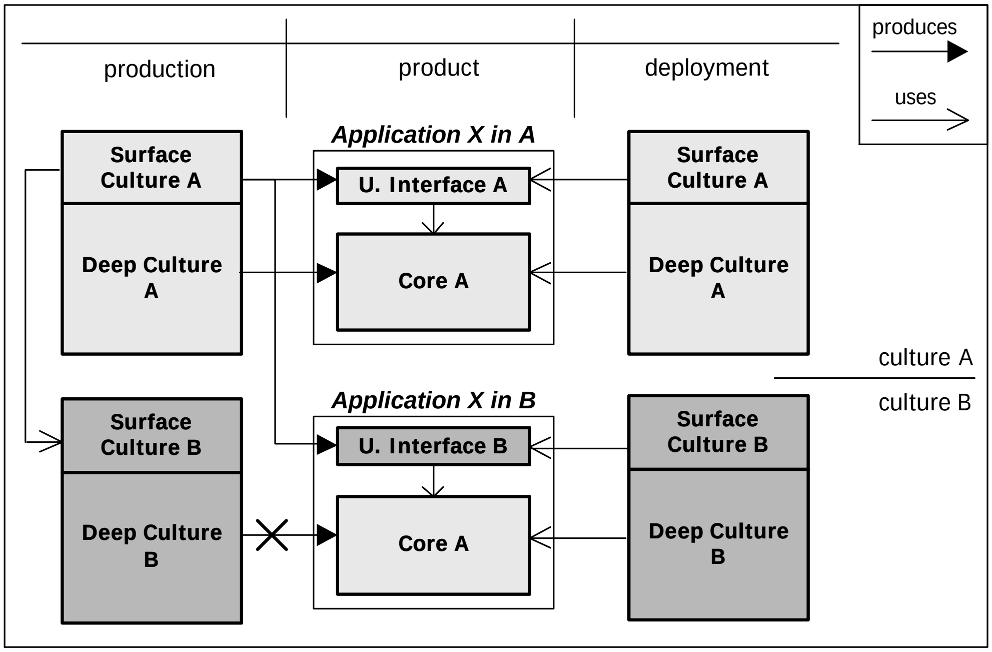

Figure 2. Internationalization of software produced in culture A.

The core of application X embodies deeper elements of culture A; those are models and procedures which are used to form messages, construct symbols, and create information and knowledge from data. Their specification and implementation is based on deep culture. Some of the models and procedures may be common to several cultures, their configuration, however, may reflect values and beliefs that are unique to a given culture.

One difference between the substantive and critical perspectives is in the impact of technology on society and its organizations. Observe that even in culture A the production culture will be somewhat different from the deployment culture. (For example, the UNIX operating system was made by programmers for programmers—its learning curve is steep as a result of numerous conventions that are particular to computer programming culture.) Taking this into account we note that the substantive theory states that a new culture emerges through the implementation and use of technologies. This new culture is neither the production culture nor the deployment culture. In contrast, the critical theory states that the production culture modifies the deployment culture. In particular, the use of Application X in B implants values and norms of A in culture B.

The goal of the internationalization architecture is, in general, to decouple culture-dependent elements from the culture-independent elements. As Hall states "all the culturally and linguistically sensitive software components need to be separated from the core of the application" (Hall 1999, p. 298). This indeed may be the case but our short review of this architecture and software localization clearly assumes that—as the result of the reductionist understanding of culture—the only culture-dependent component is the user interface. In the following section we briefly discuss examples of applications that were developed for the same purpose. The applications were developed in different cultures, and as a result the user experience that they offer is fundamentally different.

Three applications

There are many applications which implement the principles of problem understanding, decision-making, and communication. These principles are heavily culture-dependent (Steward 1992; Lootsma 1996). As an example, let us consider three well-known schools for problem solving and decision support: American, English and French, and, respectively, three examples of Decision Support Systems (DSSs): Expert Choice, Decision Explorer and Electre. These systems have been used at any organizational level and for every type of decisions, ranging from strategic to operational.

Electre

French culture, according to Hofstede (1997, p. 87), has high power distance and feminine orientation, while British and American have low power distance and masculine orientation. In France this leads to equal concern for quality of life, relationships and authority. High power distance leads to process standardization, bureaucracy and technocracy (Crozier 1964; Hofstede 1997). This would suggest support in which the consultant and the system provide technostructure and supply ideas leaving significant degrees of freedom to the decision makers accepting their authority (Mintzberg 1979).

French Electre is based on the concept of reduction of incomparability between actions through the use of outranking relations that represent decision-makers' preferences. Effort is made to exploit the relations between actions gaining information allowing for the determination of their partial comparability. In this way the software provides support to define a partial ordering of alternatives. While it aims at providing structure and support similar to Expert Choice, it gives the decision-maker control over the results without the necessity of providing a justification. The recognition of incomparability which allows the decision maker to incorporate aspects cannot be measured, explained or even voiced. As a result this method provides greater distance between the consultant, system, and the decision-maker, than the other two methods described below.

<u>Decision Explorer</u>

Organizations in Britain use adhocracy and mutual adjustment in decision-making. The process is often informal and involves people from different areas who communicate in order to define and solve problems (Mintzberg 1979). The informal approach to solve problems through communication allows equal involvement of both decision makers and consultants, who have equal responsibility.

The English Decision Explorer is based on the assumption that the search for the solution of a complex problem is equivalent to finding its appropriate representation. The roots of this software are in the development of cognitive mapping used to map thought processes. Users of the Decision Explorer are often consultants who help decision makers to construct the map of the decision problem. The underlying assumption is that if the problem and its potential implications are well understood, and can be assessed from different perspectives and in varying levels of complexity, then the solution becomes obvious. Decision Explorer concentrates on the manipulation of symbols and logical analysis of the relationships. It facilitates informal communication and requires the involvement of all those who have information about the problem at hand.

<u>Expert Choice</u>

In the US the divisional form and standardization of outputs play predominant in organizations. Consultants have a significant and active role in decision-making; it is expected that decision makers can provide all information required to solve a problem and determine an optimal decision. Efforts are made to specify the desired results and use well defined coordination mechanisms in decision-making (Mintzberg 1979; Hofstede 1997, p. 152).

The Expert Choice application implements a "measurement philosophy" in which every complex problem can be represented with a structure, which in turn can be compared with another structure. This implies that decision makers can—possibly with the help of a consultant—determine their own subjective measures of goodness (utility) allowing them to compare and order decision alternatives. This seems to coincide with the key characteristics of American organizational culture and the principle of technology dominating the policy and social choices (Zysman 1999).

Expert Choice and many other similar utility theory-based systems assume, by the very notion of utility, a full comparability of alternatives. Decision Explorer alleviates the problem of comparability by supporting insight and narrowing the space of alternatives. The French Electre family of systems attempts to bridge the two approaches in acknowledging incomparability and seeking for comparable elements.

GIS and other systems

Claims have been made that formalized approaches to decision-making, many of which form a core of DSSs, do not differ and are not a function of culture (Neganshi 1979; Al-Jafaray and Hollingsworth 1983). Similar claims are made regarding geographic information systems (GIS), which—according to Brodnig and Mayer-Schomeberger—are culturally appropriate and provide a fair representation of local cultures (2000, p. 11). They state that, while “certain purists consider GIS technology as a tool for epistemological assimilation and as such, as the newest link in a long chain of attempts by Western societies to subsume or destroy indigenous cultures. Spatial information technologies cannot capture… the cultural patterns imbedded in landscape and natural resources.” (op cit., p. 11). In doing this Brodnig and Mayer-Schomeberger and several other authors seem to forget that mapping requires decisions about representations in which strong cultural bias has been shown (Harley 1990; Cogswell and Schiotz 1996). Moreover, GIS and related technologies now incorporate models and procedures for symbol manipulation based on fuzzy logic, multiple criteria decision analysis, utility and optimization. For example, the DSSs reviewed above suggest that the use of utility has been accepted by the American culture but not by the French and English.

Some of the underlying reasons behind these surprising claims may lay in the American culture, which is highly individualistic, low-context, entrepreneurial and oriented towards tangible results rather than relationship, process and deliberation (Hall 1976; Carmel 1997; Hofstede 1997). These attributes are particularly visible in the software industry as a whole, including the culture of designers and developers. The latter may be seen as being “hyper-American” (Keniston 1997). The above three examples, other studies (Heaton 1998), as well as general studies of the philosophy of technology (Ferre 1995) indicate that there are significant differences in the software developed in different national and organizational cultures.

SOFTWARE CULTURALIZATION

Cultural Characteristics

The three systems discussed in Section 5, are examples of culture-dependent applications. They capture mechanisms that depend on culture and evolve together with other social systems: management, coordination, cooperation, decision-making and problem solving.

We are concerned here with applications which are based on organizational and intra-organizational models, individual problem solving approaches and procedures, group and meeting systems, and systems for knowledge extraction and manipulation. We are also concerned with software agents representing and acting on behalf of individuals and organizations, and systems used for games and simulations. These systems solve problems and make decisions using rules, models and procedures that are rooted in a culture and incorporate its values and symbols. Procedures and algorithms to construct and solve problems are implemented in the software core. If they are culture-dependent the core is also culture-dependent.

The depth of culture-dependence varies across application domains. A desktop productivity application, a web-based banking facility, and an e-commerce framework all capture different degrees of the culture within which they were created. The culturally dependent functionality in a productivity application, such as a drawing program, may be low because the core goal of the application is independent of culture. Internationalization makes this application more usable and deeper localization may be unnecessary if the core tasks are independent of cultural concerns. In contrast, an e-business framework, web-based customer relationship management and a web-based banking interface are forms of real-world social interaction. In order to be intuitively understandable these applications must capture a cultural model that describes how they will use motifs, metaphors, information architecture and navigation in order to mimic cultural interaction. The applications for which cultural-dependence must be considered during design are those whose core requirements involve culture-dependent tasks.

The applications for which cultural-dependence must be considered during the design process are those whose core requirements involve culture-dependent tasks and processes. To determine these tasks and processes we need to study how they are undertaken in the target cultures and compare with their equivalents in the production culture. This is challenging from both research and engineering perspectives; IS researchers and IT developers need to take into account many different perspectives from philosophy, sociology and anthropology. The difficulty in determining what can be generalized across cultures and what needs to be customized to a given culture is one of the reasons why there has been little formal research on the software-culture relationship. The few exceptions include accounting and taxation software whose core functionality depends on localized rules and conventions. In contrast, cultural differences have not been addressed in software that addresses the relationship between people and organizations, for example, customer-relationship management (CRM) and enterprise resource planning (ERP) systems, in which the cultural dependencies are perhaps less direct but just as fundamental.

Culture-dependent and Culture–independent Components

The application model partitions a system into *presentation*, *business logic*, and *data repository* tiers (Netscape 1999; Ben-Natan and Sasson 2000). The purpose of the presentation tier is to deliver a user interface to the end user of the application. This tier controls the look-and-feel of an application and responds to user events. The business logic tier maintains the application-specific processing and business rules, which define the application processing logic and process flow. It typically makes use of a component technology allowing for the scalability of applications and use of the same components in different applications.

We propose *software culturalization*, which extends the concept of software internationalization to the business logic of applications. Figure 3 illustrates the localization of an application's business logic in a similar manner as the interface localization presented in Figure 1.

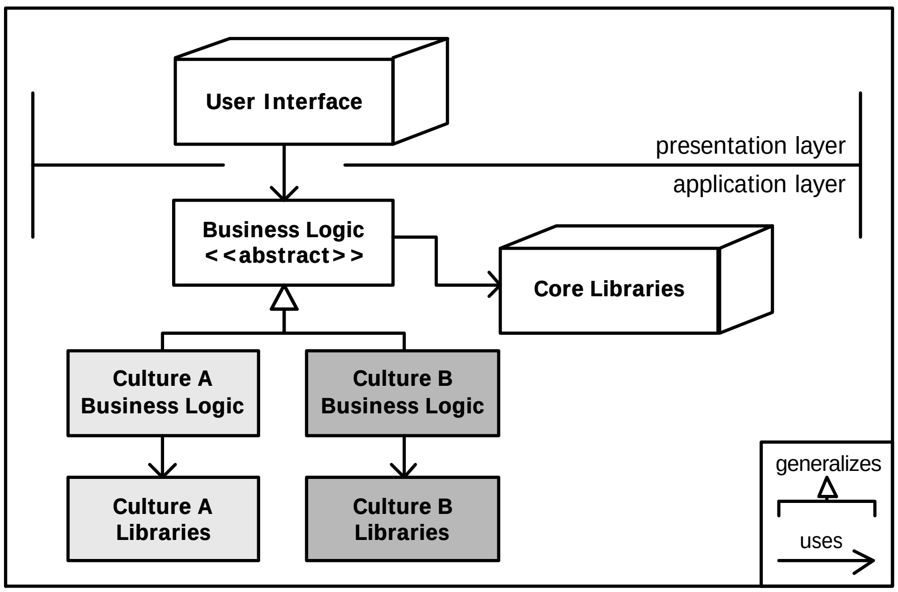

Figure 3. Business logic localization.

Software culturalization distinguishes, similarly as software internationalization, culture-dependent components from other components of the core. This is not to say that there are components that are completely independent of any culture; this would contradict the critical theory of technology on which we based the proposed architecture. We note, however, that different cultures share certain values and beliefs, and that there are certain mechanisms used to represent economic processes across many cultures. There are significant differences between, for example, Canadian, French and Polish cultures, and yet the functions used to calculate employees' salaries or the procedures used to manage inventories are the same.

To illustrate the proposed architecture let's consider a shopping cart application used in e-business. The abstract business logic module may contain concepts of a shopping cart, customer and products. Let us assume that shopping decisions in culture A are made by groups of shoppers (e.g., families) but in Culture B by individual customers. Furthermore, products may be either purchased or bartered in A while they can only be purchased in B. These behaviour related to these culture-specific shopping habits and patterns is captured in the libraries.

The construction of an application's business logic, for example, Culture A Business Logic, requires the instantiation of relevant classes from Core Libraries using entities from Culture A Libraries as culture-specific parameters according to the Business Logic design. The Core Libraries provide culture-independent components, for example, member-functions that handle addition/removal of products to/from a shopping cart. The Culture A Libraries contain culture-dependent components, for example, the attributes characterizing shopping groups and attributes of product that shoppers may barter.

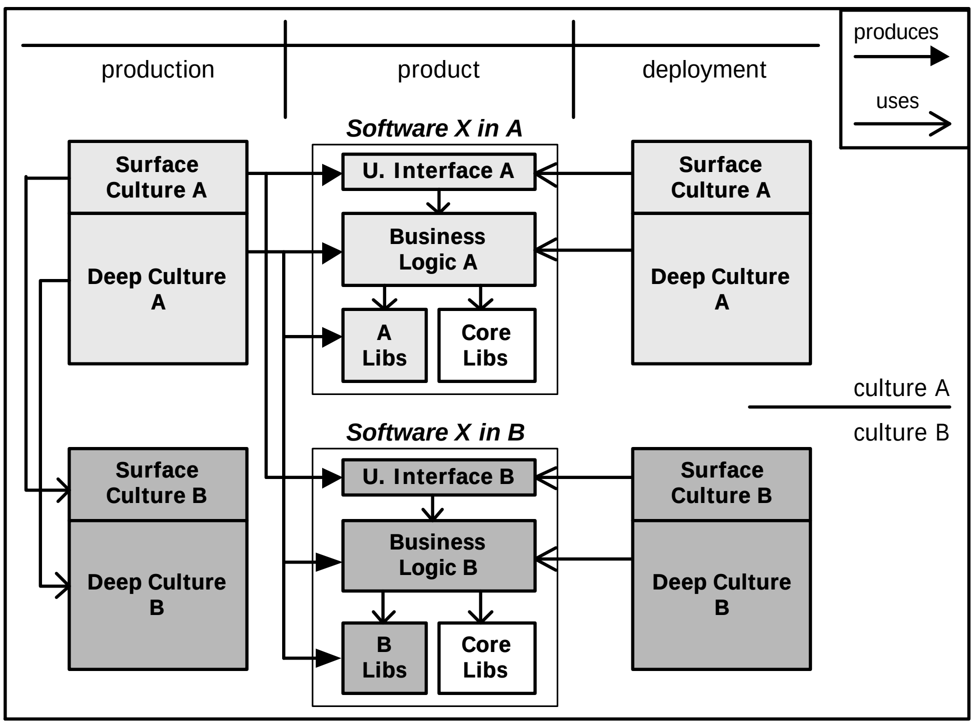

Figure 4. Culturalization of software produced in culture A.

Implementation of the cultural aspects in both interface and business logic allows for a significantly deeper software localization. In Figure 4 we illustrate software X developed in culture A and localized in culture B. The resulting software X in B includes—in contrast to the localization depicted in Figure 2—these components of the deep culture of B, which are directly relevant to the business logic.

The business logic is adapted to the particular culture in which the application will be deployed, and as such the mismatch evident in Figure 2 is removed. This removal requires, however, knowledge of the deployment culture and the ability to determine culture dependent models and procedures. It also requires that the modular culture-dependent and culture-independent components be integrated in the application core. The “Core Libraries” are reused in the products and the individual “Business Logic” strategies are adapted according to the architecture shown in Figure 3. The only part of the software entirely culture-dependent and not reused from culture to culture is the culture-specific libraries (A Libs and B Libs in Figure 4).

Culture and Software Characteristics

Determining which software characteristics are culturally dependent is key. In Section 6.1 we mentioned that this requires a multidisciplinary perspective. It also requires a reorientation form the focus on the problem in its isolation to the focus on the context in which the problem exists. For example, the problem of purchasing needs to be studied as it occurs in different cultures rather than solely in the production culture. The difficult but critical aspect of the study involves the underlying mechanisms and processes that people and organizations undertake in purchasing. These may involve the issues that are relatively easy to describe such as payment alternatives, but also such hidden issues as trust and independence.

In a typical organization of employees in an enterprise, the structure an enterprise, and the relationship among enterprises differ. Often an organization’s use of software for managing and organizing, communicating and collaborating continuously increases in scope and depth. These two phenomena have been widely researched. Study of their juxtaposition is necessary to determine those underlying structures and processes that have to be embedded in software but which differ in different cultures (national as well as organizational).

It has been long recognized that management, communication and collaboration depend on culture and evolve together with other social systems. The current perspective rooted in modern engineering is to ignore the “cultural nuances and differences” as much as possible. The engineering design process isolates the problem from the environment (cultural and other) and concentrates on its modelling, prototyping, testing, debugging, etc. Moriarty (2000) notes that this form of engineering integrates theory and practice outside of the context in which the product is developed and in which it is to function. Following Borgmann (1984), he proposes focal engineering that is concerned with the role of products in supporting “the good, in the sense of human life of harmonious connections and continuities.”

The recognition of context in the engineering design process goes beyond the user-product relationship that is of the user interface concern. The consideration includes the user-product-world relationship and the recognition that the product plays many different roles in this relationship including changing the world Moriarty (2000). This broader and deeper perspective recognizes that software impacts the way people and organization function, and requires that the designers take this into account. One obstacle in implementing this perspective may be in the role context plays in different cultures. In low context cultures communication have precise meaning and require little relationship to the broader context in which they occur; words are simply the context or the content of the communication. In high-context cultures communication have often ambiguous meaning open to different interpretations; words and media are part of larger context and need to be valued in accordance to several variables, including the previous relationship, the purpose of the conversation and the non-verbal activity.

Software development in low-context cultures may be seen more in purely utilitarian terms than in the high-context cultures where it may be seen as an element which affects, and is affected, by a larger context. In the low-context societies the user interface may need not be related to the users' background, culture and values, but to their abilities which are considered in abstract from their roots (e.g., ability to recognize and interpret meaning of a colour or an icon). The mainstream U.S. culture is, according to Harry (1992, pp. 111-112), "in comparison to that of many other countries, markedly 'low-context' in its reliance on positivistic criteria for truth and in its tendency to exclude and treat as irrelevant the complexities of human perception and personal interaction." (See also, Hall (1976) and Hofstede (1997), among others.) If the key market for application is low-context then the main concern of software developers may be software usability and its effectiveness rather than roles it may have in shaping the users culture and modifying their behaviour. This may be especially the case if software is seen purely as a product or a tool which should have well define use and capability. Ambiguity in its use and multiple interpretations are not desired. Development of software for context-rich societies is more difficult but, as we suggest in the next section, may now be feasible.

CONCLUSIONS

Software has a unique place in the world of technology because its extreme malleability and complexity makes it embodies numerous aspects of a particular culture's methods, knowledge and philosophy. The importance of computational methods and of information and knowledge has pushed forward the development of information technology (IT). IT in turn has provided resources that change the individual, organizational and national environments. By creating and manipulating information and meanings people and organizations can frame the world to reach goals with IT mediating an interpretation of the world (Poole and DeSanctis 1990). In this context culture and technology affect the information interpretation and framing.

The emergence of active decision support systems, software agents, meeting systems and brainstorming tools, knowledge management systems and other programs that contribute to the behavioral patterns of individuals and organizations affect the individual and collective “software of the mind”. Information systems in any organization embed behaviours that constitute organizational and national cultures.

In the 1990s the necessity for adapting software to international markets was recognised and resulted in software internationalization architectures. Object oriented technology made it possible for the separation of GUI objects from the culture-specific locale. Current software development paradigms such as procedural and object-oriented programming fail at capturing units of software modularity that crosscut modules. However, the component-based architectures and those derived from “meta-level” development methodologies appear to be better suited for deeper localization. For example, a promising technology called aspect-oriented programming is arising from Xerox PARC, and has been demonstrated to capture crosscutting concerns in new units of software modularity called aspects (Kersten and Murphy 1999). Since cultural concerns crosscut the system architectures, aspect-oriented programming may prove to be a valuable method of capturing these concerns.

In contrast to the implied universality of the user interface-based internationalization, the proposed application internationalization architecture is not culture independent or universal. We do not posit that separation of the Core Libraries from the culture-dependent libraries (see Figure 3) is possible for all cultures. We recognize the inherent cultural bias in every software product yet suggest that there are values, business processes and mechanisms shared by different cultures. Market-oriented economies and organizations employ many mechanisms across different cultures, the same goes for democracies and so on.

The recognition of the cultural bias embedded in software and the acceptance of the need for application internationalization presented in Section 6, requires both a broader and a deeper perspective to recognizing the role of culture in software design and development. Broader because interdisciplinary approaches are required to determine national and organizational requirements. Deeper approaches are necessary because software internationalization has to move beyond the surface manifestations of culture. The recognition of culture as a primary concern in the design of the application core is a key factor in the successful deployment of applications targeted at the international market.

Acknowledgements: We thank Dr. Rustam Vahidov and the three anonymous reviewers for their comments and suggestions. In particular we wish to acknowledge the impact of one reviewer’s comments on the final version of the paper. This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada and the FRDP of the Concordia University.

REFERENCES

Al-Jafaray, A. and A. T. Hollingsworth (1983). "An Exploratory Study of Managerial Practices in the Arabian Gulf Region." Journal of International Business Studies **14**: 143-152.

Ben-Natan, R. and O. Sasson (2000). IBM Web Sphere Starter Kit. New York, McGraw Hill.

Borgmann, A. (1984). Technology and the Character of Contemporary Life. Chicago, University of Chicago Press.

Brodnig, G. and V. Mayer-Schoneberger (2000). "Bridging the Gap: The Role of Spatial Information Technologies in the Integration of Traditional Environmental Knowledge and Western Science." Electronic Journal on Information Systems in Developing Countries **1**(1): 1-16.

Carmel, E. (1997). "American Software Hegemony." The Information Society **13**(1).

Cioffi, J. W. (1999). The Digital Economy in International Perspective: Common Construction or Regional Rivalry. E-conomy Project, University of California, Berkeley, http://e-conomy.berkeley.edu/events/deip/summary.html. Accessed: December 2000.

Cogswell, C. and U. Schiotz (1996). Navigation in the Information Age: Potential Use of GIS for Sustainability and Self-Determination in Hawai`i. California Institute of Integral Studies, http://www.hawaii-nation.org/gis. Accessed: December 2000.

Crozier, M. (1964). The Bureaucratic Phenomenon. Chicago, IL, The University of Chicago Press.

Faure, G. O. and J. Z. Rubin, Eds. (1993). Culture and Negotiation. The Resolution of Water Disputes. Newbury Park, CA, SAGE.

Feenberg, A. (1991). Critical Theory of Technology. New York, Oxford.

Ferre, F. (1995). Philosophy of Technology. Athens, University of Georgia Press.

Hall, E. (1976). Beyond Culture. New York, Doubleday.

Hall, P. (1999). "Software Internationalization Architectures". Decision Support Systems for Sustainable Development in Developing Countries. G. E. Kersten, Z. Mikolajuk and A. Yeh, (Eds.). Boston, Kluwer**:** 291-304.

Hall, P. and R. Hudson (1997). Software without Frontiers. New York, Wiley.

Harley, J. B. (1990). "Deconstructing the Map." Cartographica **26**(2): 1-20.

Harry, B. (1992). Cultural diversity, families, and the special education system. New York, Teachers College Press.

Hart-Davidson, B. (1997). Locating the Techno-Discourse. Purdue University, http://omni.cc.purdue.edu/~davidswf/tds.begin.html. Accessed: December 2000.

Heaton, L. (1998). "Preserving Communication Context. Virtual Workspace and International Space in Japanese CSCW". Proceedings of the Cultural Attitudes Towards Communication and technology, C. E. a. F. Sudweeks, (Ed.). University of Sydney: 207-230.

Heidegger, M. (1977). The Question Concerning Technology. New York, Harper and Row.

Hofstede, G. (1997). Cultures and Organizations: Software of the Mind. New York, McGraw-Hill.

Juustila, A. (1995). "Interaction of Culture, Power and IT in Organisational Change". Proceedings of the Information Systems Research Seminar in Scandinavia.

Kaplan, B. (1995). “The Computer Prescription: Medical Computing, Public Policy, and Views of History.” Science, Technology and Human Values **20**(1): 5-38.

Keniston, K. (1997). Software Localization: Notes on Technology and Culture. Cambridge, MA, MIT Program in Science, Technology, and Society: 1-22.

Kersten, G. E., S. Matwin, et al. (2000). "The Software for Cultures and the Cultures in Software". Proceddings of the 8th European Conference on Information System. ECIS2000, H. R. Hansen, M. Bichler and H. Harald, (Eds.). Vienna University of Economics and Business Administration. **1:** 509-514.

Kersten, M. A. and G. C. Murphy (1999). "Atlas: A Case Study in Building a Web-based Learning Environment using Aspect-oriented Programming". Proceedings of the ACM Conference on Object-oriented Programming, Systems, Languages, and Applications. ACM Press: 340-352.

Kuper, A. (1999). Culture. The Anthropologists' Account. Cambridge, MA, Harvard University Press.

Laurel, B. (1991). Computers as Theatre. Reading, Addison-Wesley.

Lootsma, F. (1996). “Comments on The European School of MCDA.” Multi-Criteria Decision Analysis **5**: 37-38.

Mintzberg, H. (1979). The Structure of Organizations. Engelwood Cliffs, NJ, Prentice-Hall.

Moriarty, G. (2000). “The Place of Engineering and the Engineering of Place.” Techne **5**(2), http://scholar.lib.vt.edu/ejournals/SPT/v5n2/moriarty.html.

Nakakoji, K. (1996). “Beyond Language Translation: Crossing the Cultural Divide.” IEEE Software (November): 42-46.

Neganshi, A. R. (1979). "Convergence in Organizational Practices: An Empirical Study in Industrial Enterprises in Developing Countries". Organizations Alike and Unlike. C. J. Lammers, (Ed.). London, Routledge.

Netscape, I. (1999). Building Applications in the Net Economy. Netscape Inc., http://developer.netscape.com/docs/wpapers/platform/. Accessed: December 2001.

Pacey, A. (1992). The Culture of Technology. Cambridge, MIT Press.

Poole, M. S. and G. DeSanctis (1990). "Understanding the Use of Group Decision Support Systems: The Theory of Adaptive Structuration". Organizations and Communication Technology. C. Steinfield and J. Fulk, (Eds.). Newbury Park, Sage: 173-193.

Sauter, V. (1997). Decision Support Systems. New York, Wiley.

Sommerville, I. (1992). Software Engineering. Reading, Addison-Wesley.

Steward, T. J. (1992). “A critical Survey on Multiple Criteria Decision Making Theory and Praxis.” Omega **20**(5/6): 569-586.

Taylor, D. (1992). Global Software. Developing Applications for the International Market. New York, Springer Verlag.

Zysman, J. (1999). "Introduction and Overview: Common Stakes in the E-conomy". Proceddings of the The Digital Economy in International Perspective: Common Construction or Regional Rivarly, J. W. Cioffi, (Ed.). E-conomy Project, University of California.