Koranteng Ofosu-Amaah

Cultural Sensitivity in Technology

Everything is local.

The "Coltrane" release of Lotus Freelance Graphics, the 1998 version of the presentation graphics program bundled in Lotus SmartSuite, was famously held up for a month when it was found out that one of the standard clip art images used in the out-of-the-box presentations had a tiny 20-pixel image of Taiwanese currency (rather than mainland Chinese currency). Or was it the other way round? An eagle-eyed quality assurance engineer discerned the offending image. Needless to say, this was something that could not only cause offense but also affect purchasing decisions in the involved markets. Productivity suites like SmartSuite are very profitable products since they are among the indispensable tools in every office; everyone needs word processors, spreadsheets, organizers, and presentations.

I was a little canary in that Freelance mineshaft (or was it a minefield?) when a couple of months earlier in the project I discovered a seriously outdated map of Africa while integrating new clip art into the product. I wrote one of the "software problem reports" or "sprs" of which I am most proud; it was entitled something like "Upper Volta should be Burkina Faso." The spr also made a brief mention of the

fact that, as far as I could tell, there were similar problems in our maps of the former Soviet Republic since many of the states' names had changed earlier in the decade. I believe that my spr was deferred to a later release; it was too late in the product cycle, and we'd have had to go back to the company we licensed the art from, and so forth. As it turned out however, the China/Taiwan currency error caused the entire suite to be delayed while the test team painstakingly examined the clip art and other areas of the product, vetting for similar outrages that could endanger sales.

Incidentally, the acronym spr is a distinctively Lotus term. After a decade of indoctrination, I continue to use it rather than the more clinical "defect," which is the favored term at IBM, or "bug," which is the more widely used term generally. In fact, you can still tell whether a Lotus employee has the "old Lotus" DNA or the "new Lotus/IBM" variety by the terminology they reflexively use.

As a further aside, the Lotus SPR database is actually one of the most successful and useful applications of Lotus Notes technology ever, and, in a similar manner, the Bugzilla project is one of the best things to have sprung out of the Mozilla effort. Certainly it was more useful than the Mozilla Gecko browser suite (mother to the Firefox) until the M7 milestone, the Mozilla 0.7 release, was reached years ago. This suggests that a new kind of software law is at work here, one analogous to Zawinski's Law* on "software expanding until it can support email" (and now feed technology). According to this law, which I'll call Koranteng's first law of software systems:

---

* Jamie Zawinski coined his law of software envelopment as follows: "Every program attempts to expand until it can read mail. Those programs which cannot so expand are replaced by ones which can."

> A software platform reaches its tipping point once it can serve as a bug reporting system.

Intuitively this makes sense: Once developers can view, file, and retrieve bug reports using their own products, they will be more likely to use them, and their confidence in their mission will improve dramatically.

But back now to my original topic. I'm sure that the artists who created the clip art in that design firm had no subversive intent; they probably just used outdated stock art as their source material. Still, you sometimes wonder at these things, and, in this vein, I'm reminded of the Nobel Prize–winning Portuguese author Jose Saramago's wonderful novel, *The History of the Siege of Lisbon,* in which a copy editor proofreading a historical novel defiantly changes a crucial word in the text and in this way also changes the outcome of a famous historical battle from a miserable defeat to victory. A masterful alternate history of Portugal then develops, and its culture is fully reimagined. I'm sure that bored developers sometimes slip similar things into their products, and not just serendipitous "Easter Eggs." In the same release of Freelance, we put together a hidden game and a screen show with the photos of all the developers that could only be reached by activating an unusual key sequence (I won't kiss and tell here). These days, there are Web sites that catalog the hidden secrets (and occasional hidden features) that can be found in many software products.

More seriously though, the Freelance incident (or similar incidents like those involving the release of new dictionaries and thesauruses from Microsoft) goes beyond the cultural sensitivity with which this post is concerned into the rarefied realm of political sensitivity. For example, if you typed the phrase "a fool," the Microsoft Word 2000 thesaurus would offer "trick" as a replacement. The opposite

was the case when typing "zzzz" in Word 97, in which case the thesaurus suggested "sex." Later service packs had to be issued to "fix" these and other glitches. This is also the case with product names: You don't want the mere mention of your product to cause snickering or give offense. A misconceived name or culturally insensitive feature can be a black eye that will embarrass and divert revenue from even the highest-flying company. In 1998, Lotus SmartSuite was a $400 million a year business; that month's lost revenue was no small thing.

FORENSIC INVESTIGATIONS

Although obviously important, the task of internationalization, which aims to make technology available in different languages and scripts, is a mostly thankless, and oft-neglected, one. From a developer's standpoint, you think about it mostly in terms of dealing with "resource bundles." Wherever you find onscreen text, you have to replace it with a key word and put the text in a file, along with the key word to reference it. This file is then sent off to translators for each of the different languages you support. The resulting bundle of translated files is packaged in the released program. When the software runs, it figures out the user's language, looks up the correct language file, and uses the given key word to obtain the resource string for display.

Designing software in this way allows you to support new languages by simply dropping a text file for a new language in the right location. This is a pain because you probably started with a little prototype of a user interface and now you're being asked to find all those hard-coded strings in the user interface and "do the right thing." It is also confusing because, from then on, you see key words instead of phrases in the dialogues in your development environment.

Adding to this irritation is the fact that internationalization typically only becomes an issue late in the game, when the really interesting work is over and you're ready to think about the next big thing. Instead, you're grudgingly forced to worry about handling different writing systems; dealing with bidirectional text; or (that old favorite) wrestling with the "special characters" in the wide variety of programming languages, file formats, protocols or operating systems in your software. In computer science there is a general category of problems that have to do with delimiter characters (underscores, hyphens, colons, semicolons, angle brackets, and other special characters). Software is very concerned about structure, and the parsers of our Tower of Babel file formats need to know where records begin and end. Every tribe in the software world has its idiosyncratic ideas about language, and which characters to use to separate records, as it seeks to impose structure on the world. There are ever-shifting alliances, and fashions, and it is a heady proposition to make sense of things as the tribes interact. Those who excel at these tasks are akin to forensic investigators: patient, precise, and possessed of a keen eye for detail.

The consequences of overlooking these seemingly small details are, again, larger than one might expect. If, for example, your software mangles names with accents, you'll have real trouble selling in France. The same applies to ampersands and apostrophes—which have significance in SGML (standard generalized markup language) and its derivatives (HTML and XML). And don't get me started on wider character set and encoding issues. As an ironic case in point, while developing Lotus K-station, IBM's first attempt at a portal, one of our best business partners was somewhat stymied in his development and extraordinary evangelism of our product because, in its earliest release, Lotus K-station couldn't handle the ampersand in his company's name.

Luckily for us, he temporarily renamed his organization in his corporate LDAP directory; Sun & Son became Sun and Son until we worked out the quirks. To this day, I always make sure to test whatever product I work on with its organization name. It's surprising though, how often this kind of problem recurs.

THE SCENE AT HOME

In the Internet era, most technology, and certainly all software development, has to have global concerns in mind. It is said that the sun doesn't set on an IBM project, and it is true that I work with a very diverse set of colleagues the world over. Presumably, one of the benefits of such a widely dispersed and diverse workforce would be to mitigate the likelihood of issues in this area. This benefit, however, is only realized if everyone gets the opportunity to give their input. As my current project has been going through translation and localization testing of late, I've been thinking a lot about the different strategies for handling internationalization.

The "old Lotus" process, for example, was fairly decentralized: Each product group would be assigned an internationalization team. In addition to being localization gurus, members of such teams were domain experts and knew the product inside out. Having the team involved from the very beginning in the product development cycle had many benefits since it enabled them to give crucial design feedback very early and iteratively. Your first prototype was immediately critiqued from their standpoint. The obvious downside of decentralization was the lack of uniform standards, chaos that our translation teams couldn't bear. Some products were very easy to localize; others were, to put it mildly, far less so. This was in keeping with the culture of Lotus, which was historically full of artists, writers, historians, soci-

ologists, physicists, and folks like me: electrical engineers who picked up software engineering almost as a hobby.

As we transitioned to the more centralized IBM globalization process, we got the benefits of uniform standards and greater resources. More languages could be supported in the initial release, and you could at least point developers to documentation about the processes they should follow and in that way avoid the usual ad hoc stumbling about. The IBM style is all about corporate professionalism; it is more ponderous and process minded: hence the "Big Blue" nickname. There are many benefits to organized processes, but they can also come at the expense of domain expertise, sensitivity, and a much faster feedback loop. Members of the test teams are generalists and, on any given week, will be testing a wide variety of products. They often aren't aware of the nuances of your particular product and are often barely getting up to speed with it by the end of the testing phase. Hence localization issues frequently surface too late in the game and cause unnecessary reworking and product delays. These processes are of course constantly being tweaked, as all processes are, and in recent years, I've seen a renewed emphasis on domain expertise in the internationalization teams that has improved our product development considerably.

One of the lessons here may be that software engineering is very different from computer science, precisely because of its greater preoccupation with what Graham Greene called "the human factor." For once you bring people into the picture, you bring culture: conversations, marketplaces, attitudes, details, conventional wisdom, and sometimes blind spots. Working on the Web exposes one to a multitude of such voices, and, for that reason, I view it as a conversational puzzle. The clues are sometimes irreverent and cacophonous, but the solutions are always interesting and, once found, ultimately rewarding.

Here is a brief example, which should help to tease out the kind of technical, design, and business dilemmas that can arise if you aren't attentive to the cultural issues in your product development.

I embarked over the past couple of weekends on a mass digitization project and scanned, retouched, and uploaded 2,000 or so old photos from shoe boxes under my bed. The technology involved in this exercise was scanner hardware and image acquisition software, the bundled Adobe Photoshop Elements for color adjustment and red-eye correction, and a couple of online photo-sharing services: Yahoo Photos and Flickr (coincidentally, I started the day Yahoo's acquisition of Flickr was announced).

I noticed very quickly that all the photos that I uploaded to Yahoo Photos had somehow turned out darker than on Flickr. Both services resize uploaded photos; when you reduce the size of images, you have to select the pixels and colors you are going to use, but the photo-resizing algorithm used by Yahoo Photos was giving worse results. This was noticeable to me because a large number of photos featured darker-skinned people like me. The originals looked fine on the screen and wherever there were lighter skin tones. In the case of darker skin tones, however, the resized photos were not so good. This meant that if I didn't believe in the virtues of Save Lots of Copies Everywhere (SLOCE), I would have leaned toward Flickr and stopped using Yahoo Photos.

I also noticed that my experience of the Flickr Web site was very different from that of my family and friends who used Internet Explorer. Almost all of them complained immediately about the first bunch of photos I presented to them. My initial thought was that the problems stemmed from the different browsers we were using (Internet

Explorer in their case, Mozilla in mine). After a little investigation, however, I found that the real reason for the complaints was the Flash plug-in. If you had Flash installed, Flickr was coded to use it to display images. On the other hand, if you didn't have Flash, the browser's native rendering took over the display task. It turns out that images that are rendered in the Flash plug-in have a slightly darker tinge than the images rendered directly by the browser itself. This is not normally noticeable unless darker skin tones are involved, as was the case here. This problem became even worse when they tried the screen show feature. The black background of a Flickr screen show (also implemented in Flash) impaired the contrast still further.

Finally, it became clear while I was retouching the photos, and constantly forced to tweak them manually, that the Quick Fix and Auto Correct options in Photoshop were also better suited for lighter skin tones. Now, this tweaking is not a big deal in the case of a few photos—indeed it's fun to fiddle with photos. But after a couple of hundred images, it gets tiresome. I found myself longing for "smarter" recognition by the software or, at least, for a nice "dark skin" option that I could set in a preferences dialogue. In short, I started to think about abandoning Photoshop for a different program.

I mention these nitpicks with otherwise excellent and useful products because of the larger design issues they raise. Technology is simply a tool to serve people, and, obviously, people live in significantly different societies and cultures. We all know that different cultures adapt technologies in different ways to suit their local preoccupations and concerns. And I have certainly had my own localized concerns these past weeks.

Even when the technical fixes are easy, there are design dilemmas and economic trade-offs that arise. True, Macromedia could implement better JPEG rendering in Flash, but

that comes at a certain expense: A good renderer is a hard thing to write (even if they could license the photo-rendering code from the Mozilla folks). Also, what they have appears to be good enough for most people (except for me obviously). So, when do you decide that your product is good enough that you can stop pandering to the Long Tail? Can you *ever* afford to do that? Aren't you in danger of missing out on a vast market opportunity?

Yahoo Photos could certainly implement a better photo-resizing algorithm—although presumably there's a performance penalty to be paid if you use a more color-accurate algorithm (or perhaps a larger resultant image size). Since the Yahoo service operates with tens of millions of users and photos, this could potentially limit the scalability of their platform in a serious way. Conversely, if all Yahoo users switched en masse to Flickr, which uses more expensive algorithms, would their platform be able to handle it? Or would it generate a case of teething problems and turn users off because of poor response times, and so on?

Flickr could very easily provide a JavaScript and native HTML browser screen show alternative to their Flash-driven version (as all the other photo services do). The fundamental reason for a screen show is to display a set of images in sequence. It takes perhaps ten lines of JavaScript code to implement something that will work in virtually every browser that exists. The only benefit of Flash is to provide transition effects between images—to be sure, transitions are flashy and liven up screen shows, but that is a matter of style rather than substance. A major downside of Flash is that it isn't included in a default installation of most browsers; one has to go and actively download and install it. The endemic problems users encounter with the installation of software on local machines are ironically part of the reason many have moved to using the simplified interfaces of

the browser and the content of the Web. Flash has historically also been problematic when it comes to accessibility, which is the term that software developers use to describe a program's usefulness to people with disabilities. The use of Flash in the browser raises issues with keyboard navigation or high contrast schemes, which can cause difficulties for the sight or hearing impaired. In short, browser content that is rendered in Flash is opaque to many classes of users, making me wonder how many people they have turned away by not providing a native browser screen show. Flickr uses Flash extensively in the rest of their product, and, in the case of the screen show, its usage serves to discourage people from downloading images. Is this pseudo-digital rights management (DRM) an essential feature of their service? A photo-sharing site that makes it difficult to share photos doesn't sound right to my ears. A possible alternative would be to keep Flash but offer differently colored backgrounds for screen shows, in order to avoid the kinds of contrast issues I encountered. But where would that option show up in the user interface?

Similarly, Photoshop could implement a slight variant of their various Quick Fix and Auto Correct features that would be more attuned to my kind of skin color (indeed I assume that photographers in Africa have written macros or filters that do such a thing). How best then to phrase a global preference in an options dialogue in Photoshop? "Adjust for darker skin tones"? Documentation writers would have a field day finding the right verbiage for such an option. Also what about usability? If you add all these preferences to your product what would your user interface look like? Try typing the "about:config" URL in a Mozilla browser to get a sense of the complexity that modern software developers face.

If there were a huge market for these products and services in Africa (which is unlikely given the low Internet

penetration rates and presumably widespread instances of software piracy), the issues I faced would of course be a real problem for the companies in question. There would be demand not just for local language versions (say, a Swahili language version in Kenya) but also for tweaks that would make these services more closely attuned to the prevailing culture and, in this case, ethnic backgrounds.

Over the past 150 years, as photography has evolved into the digital realm, photographers in Africa have had to deal with brighter sunshine and higher contrast, as well as darker skin tones, when processing their photos. The people who install photo laboratory hardware in Ghana, where I come from, always have to recalibrate their equipment to deal with the kind of skin tones that dominate the local market. The factory defaults simply won't do. I've had better results developing film in Ghana than in the United States because I often forget to tell the labs here that they should "watch for skin tones."

I'd expect then that software that was truly local (by which I mean, sensitive to local concerns) might sometimes need not just run-of-the-mill language changes, or even writing system changes, but also, as seems to be the case here, algorithmic adaptations.

As software designers, we try to engineer simplicity and refrain from overwhelming users in their interaction with our services and products. Our main focus is usability—for the individual users, for the business community, and for society as a whole. Yet there are very real and often competing concerns about the application of technology in different cultures. So the next time you see a vaguely worded so-called Turkish option somewhere in your application's configuration dialogues, know that someone somewhere was likely adapting their product for a local market. Join me, though, in saluting the developers, testers, product

managers, and designers who collectively worked together to come to that solution. I'd hazard that the tweaking of the product was to fix a deal breaker in some market.

Finally, and for what it's worth, I find endlessly fascinating this notion that cultural sensitivity in technology sometimes necessitates algorithmic adaptation. Maybe though, iterative adaptation in response to local environments—evolution, in short—is the name of the game. Perhaps that's simply the way things should be.

POSTSCRIPT (A YEAR LATER)

I reported my issues to Flickr and later prodded them with some other folktales about their excessive use of Flash. I know I wasn't the only one with complaints, but I'd like to think that my slightly different framing of the issue helped tip the balance: a month later, they stopped using Flash to display images and moved to a much lighter weight HTML user interface. They continue to use Flash to drive screen shows and organize albums, but, to their credit, they expose enough of their internals to allow third parties to step into the breach to provide alternate interfaces if needed. There is still work to do to "fix" Flash rendering of images, but that doesn't concern me much since I don't use it. Adobe Photoshop will be a challenge however. I'll simply note that about 5 to 10 searchers now come across this article every day trying to solve the mystery of "darker skin tones Adobe Photoshop," pointing to an unfulfilled need. If that number continues to rise, I'll hazard it won't be long before we see a "dark skin tone" option appearing in the Photoshop preferences. I look forward to that day.