



Note: This article is scheduled to be featured in the August/September 2009 issue of MultiLingual Computing Magazine, in Adam Asnes' Business Side column.

Internationalization ROI

It's easy to get agreement that revenues beyond a company's home country market are important. If you look at some of the great global US brands, you'll find that global revenues are 50% or even greater than 65% of their gross. While much has been made of measuring the return on investment for localizing software, what about measuring the very process of making software which is internationalized so that it can be localized and supported worldwide?

There are lots of issues to measure, and they vary in emphasis for the company which is making its first efforts outside its home market, to companies that have highly evolved processes for global releases.

First we must consider opportunity costs, backing up marketing and sales efforts, competitive pressures and right down to cost of engineering. Now typically ROI calculations get down to hours saved at a particular rate, which is certainly valuable information and usually those numbers are paramount to analyzing any kind of process changes. But if a company is making new efforts or experiencing painful delays in global releases, opportunity costs and major market factors are deal makers and the stuff that executive level directives are made of.

Internationalize or Die

This heading may sound dramatic, but it's quite the case for some of our clients. For instance, we have a client whose software platform is used by third parties in e-commerce efforts. Many of their accounts are well recognized names in retail and merchandising, who are beginning to look at markets outside the US as important to their brands. While our client is not interested in purchasing localization themselves, if they can't make their product support data management and presentation in multiple languages and locale sensitive formats, they will lose their customers to competitors. I asked their senior management what was at stake, and they replied nothing less than their company's future growth and survival. Given that this a billion dollar company, I'd say that's a pretty big opportunity cost ROI on an internationalization effort.

Opportunity Costs

Internationalization happens because it's first and foremost a business driver. I have yet to meet the development team that decides to internationalize just because it would be an interesting task. So I think it's appropriate to first consider business drivers outside of the development process itself.

Perhaps global sales efforts have been taking place with a US English product. Outside of development, there are costs of sales, marketing personnel, supporting distributors, legal and administrative costs to name a few. These all have expensive price tags, which are independent of having an internationalized

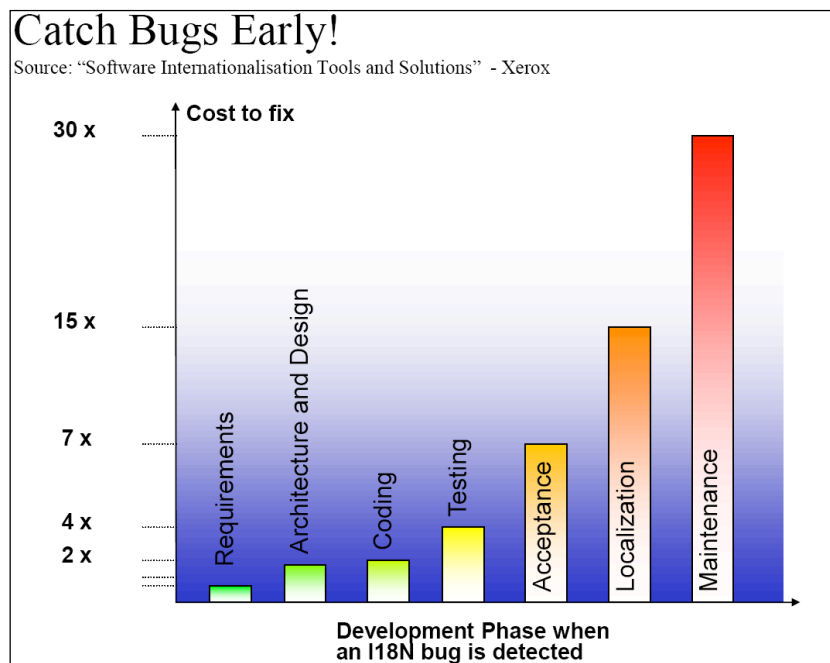
and localized product. And an internationalized and localized product has been shown to make those representative costs far more effective at producing revenue.

Cost of Delays

In an earlier article and subsequent whitepaper on our site, I outlined the cost of being late. The quick summary is that the marketing team will typically have projected revenues for each market, but dependent upon release criteria. If a product is a single quarter late, which is not bad for a large project for some software development teams, they just lost a quarter of their year for the sales teams to meet those projections. What's the value of one quarter of sales effort? If those sales efforts are expected to produce increased results over time, how does that roll out and effect market penetration in future years? While these are broadly variable scenarios, I always like to consider the "top end" revenue implications before beginning to count development hour savings. The top end always has far broader consequences and those opportunity costs get very real with numbers followed by many zeros in a competitive world.

Cutting Development Costs

My company, Lingoport, has just released Globalyzer 3.0 which is aimed squarely at supporting entire development organizations. It's actually the only commercial system of its nature, purpose built to support a very broad list of programming languages, measuring, filtering, reporting, tracking and even fixing internationalization issues over the development processes via its client, server and database components. Companies have products to measure coding quality, security issues, memory management and more. Now we are adding static analysis of internationalization to the source code development process. Remember, if so much revenue is riding on global markets, doesn't



It's far more efficient to find and fix i18n issues at the source level, rather than depending upon testing and localization iterations.

it make sense to actively measure and aid software globalization issues, just as much as software

security issues? Why not check source for embedded strings, locale-limiting methods/functions and classes, Unicode compliance, Font issues, i18n limiting programming patterns and the like at regular automated intervals rather than waiting until QA or localization? Remember the management principal

that if you want to improve anything, measure, track and report it as close to its creation as possible. What gets measured gets done.

Cost per I18n Bug – Case study with mature localization practices

In working with a new client, which is already quite mature in their localization and internationalization efforts, we had the opportunity to get actual ROI data, based on real internationalization bug fixing costs they had measured over 60 localized products. After cleansing that information of confidential data, they gave me permission to share it though limiting the data to results from 17 products.

Traditionally, they have been finding internationalization bugs during internal and external localization QA testing efforts, including both Psuedo-Localization (creating fake translations for testing purposes) and actual localization testing performed by both their organization and vendors. They counted five organizations touched by internationalization errors: Localization Vendor QA, Localization Project Management, Internal Localization QA, Product Development QA and Core Engineering. The process goes something like this:

1. Internationalization bug is discovered and reported during Localization
2. Project Manager tracks the bug, may enter or flag it in a bug tracking system
3. Core Engineering, which likely has moved on to other efforts by now, must assign and fix the bug
4. Product Development QA must verify the fix and any other issues the fix may have affected
5. Additional Localization efforts may need to be made for the same issue

This iterative process gets pretty expensive. Remember that a maxim for software development is that the earlier you find and fix bugs, the less expensive. Fix a bug before a QA cycle, and you save multiple people having to process that bug in some way, and retest the solution. Need to fix a bug after release? Costs get much worse. This principal is a major contributor to the popularity of moving to agile development cycles, so that you enhancing and verifying software in smaller, successful, less expensive cycles.

Our client figured on an average of 25 internationalization issue bugs per release, an average of 10 hours spent cumulatively by the five groups per bug , with an average of 60 releases per year over these 17 products. Some products had zero i18n bugs reported, others had over 100. The business case for finding internationalization issues in source code as part of regular automated processes integrated into their build cycle gets very clear at this level. They estimated savings of \$420,000 per year, just on reducing localization QA costs. By finding the issues early, total product development savings were calculated to be over \$760,000 per year.

-Please see the table on the next page-

Resource	Hours per i18N bug	Avg # bugs per release	Avg product releases/yr	Avg Cost per hour	Cost
L10N Vendor QA	1	25	60	55	82,500
L10N PM	1	25	60	45	67,500
L10N QA	2	25	60	45	135,000
Core QA	2	25	60	45	135,000
Core Eng	4	25	60	57	342,000
Total Cost/Yr					\$762,000

Remember that even maturely localized products, still have regular new release cycles, which in turn create the potential for new internationalization issues. Product Development never really stops, and teams tend to be more broadly geographically distributed than ever before. That makes measurement tools all the more valuable for localization savvy companies.

Cost per i18n Bug – Case study, product has never been localized

When you consider companies engaging in early globalization efforts, the payback simply multiplies per product as you can expect the i18n bug count to go way up. Without a tools-based approach to finding and fixing issues, internationalization will be very heavily trial and error iterative. One can write a few scripts which will take considerable time, research and effort, and still likely produce unreliable results. Then you can pseudo-localize display strings after you’ve found as many as you can and externalized, or populate the database with target encoding data. You would then test, test and test again while you had to hunt down the issues one by one in the source. This only multiplies the cost per i18n bug. By finding issues first at the source level, you can actually begin to orchestrate their correction, tying directly to that issues precise location within hundreds of thousands, or even millions of lines of code. And that’s an intelligent way to find and remove a needle in a haystack.

The table below illustrates the costs of i18n bug iterations for a single product of about 500,000 lines of code during the first internationalization effort. This table doesn’t include additional costs of researching and implementing various scripts and homemade utilities to help the work get done. It also doesn’t take into account that a tool like ours actually isolates i18n issues, pinpointing them in source, while also

facilitating batch externalization of strings – both very tedious and time consuming activities. Consider that even a simple error message that gets missed using traditional scripts and trial and error, may not show up at best during late QA efforts that force the error to appear, or worse, after product release. We commonly hear that it takes three or four localization releases to weed out those sorts of issues that get missed so easily. That is why this table lists a higher i18n bug rate for 2 subsequent releases than the table used for the localization mature company earlier in this article.

Resource	Hours per i18N Bug	No. of bugs	Cost per hour	i18n Bug Fixing - 1st release cost	i18n bugs in new releases	Add'l releases per year	Release i18n bug costs	Annualized i18n Bug Costs
L10N Vendor								
QA	1	300	55	16500	45	2	4950	
L10N PM	1	300	45	13500	45	2	4050	
L10N QA	2	300	45	27000	45	2	8100	
Core QA	2	300	45	27000	45	2	8100	
Core Eng	4	300	57	68400	45	2	20520	
Total Costs				\$152,400			\$45,720.	\$ 198,120

Assumes 500,000 lines of code for one product, traditional iterative internationalization efforts on legacy code with in-house tools: e.g. Scripts, GREP, rather than a commercial system that finds, fixes and audits code for i18n issues. Figures based on real experience working with companies on i18n projects.

Pitfalls and Adjustments

I think it’s fair to say that no tool offers a panacea. The strike against coding quality checkers in general has been complaints about over reporting errors, often referred to as false positives. It’s true that if you overload a developer on data that is only partially relevant, that data risks being ignored. That is why any enterprise scalable solution must include dynamic ways to filter results, share those filter controls and track them over time. You also must have flexible detection, so that you can add unique parameters that invariably crop up and can be quite particular to a specific code base.

New processes may not be greeted with enthusiasm by development teams which are typically already over tasked and under-resourced, so it’s important to help them understand the meaningfulness of getting global releases out faster and with higher quality. Automating code checking and reporting during a regular process like a periodic build is an excellent way to track and highlight progress.