Getting Your Medical Devices to Market Faster, at Lower Cost, and on Schedule

Jerry Krasner, Ph.D., MBA

You only need to do a few things right in your life so long as you don't do too many things wrong – Warren Buffet

Introduction:

In 1931 Olaf Stapledon's book, *Last and First Men*, told the delightful story about a Martian invasion of Earth in which the Martians and denizens of Earth were oblivious to one another.

The invaders from Mars, looking for resources (in particular diamonds), consisted of a cloud of microorganisms possessing radio telepathy and a distributed consciousness. They possessed a sophisticated culture, but in their search for diamonds they looked right past the slow moving and slow communicating human inhabitants of Earth.

Being a old time warrior of the medical devices industry dating back more than 40 years and more than twenty 510k applications, Stapleton's story struck a familiar note when I look at today's medical devices industry.

Most medical device developments still use Microsoft OSes – notwithstanding the availability of faster, lower power OSes that have demonstrated a faster time to market and that developments using these non-MS OSes require fewer engineers and experience lower design costs. – not to mention the significant improvement in reliability provided by the non-MS OSes. We see the same blindness when it comes to commercial versus free Linux and with Open Source software. Medical device manufacturers are taking these "evangelists" at their word without any tangible proof. The theory is that commercial OS vendors are ripping off companies when "free" stuff is available. *We will provide data from actual developments to refute these claims.*

Yet, like Stapleton's slow moving and slow communicating humans, many medical device developers and their managers remain oblivious to proven technologies that can enhance their design experiences, create lower costs of development and provide for better audit trails should they be needed for CDRH review.

Wireless technologies have been adopted as a major part of patient monitoring, yet (as demonstrated in year-over-year EMF Surveys of Embedded Developers) they are seldom integrated into the development process in the most productive manner. The same can be shown for USB integration and deployment.

In a recently updated paper entitled *"Critical Issues Confronting Medical Device Manufacturers, Their CEOs, CFOs, Managers and Developers*", EMF took the reader through steps designed to "cover the bases" in order to minimize recalls and to have documentation to use to defend against a CDRH audit.

In this paper we wish to acquaint medical device manufacturers with currently available technologies that are used effectively in other vertical marketplaces to speed up the development process, lower the cost of development and better enable integration of wireless, USB and very efficient GUI development and deployment. The advantages of modeling will be discussed.

In this paper we will consider the following:

- Why choosing the OS most suitable to the application can produce significant savings
- What wireless technologies are being used by medical device developers – and what technologies are available that can enhance wireless and USB integration
- How to simplify GUI development and deployment.

Choosing the Most Suitable RTOS

Medical patient monitoring is *not* a mission critical design – the fastest response times are measured in milliseconds and critical alarm design can compensate for any systems failure. Nonetheless it was interesting to see:

1) The most widely used RTOSes did not exhibit the best results in time to market or design outcomes;
2) Certain RTOSes, e.g., ThreadX and Nucleus – interestingly created by the same individual - had better time to market, fewer designs completed behind schedule, and a lower cost of development than the more established RTOSes (e.g., Windows CE, VxWorks, Integrity, LynxOS) as well as Linux and Open Source
3) Commercial Linux outperformed Open Source
4) These comparisons have been calculated year-over-year (shared with subscribers to the EMF Market Intelligence Program - but not published openly) with the same outcomes, over the period 2007-2013

The unique EMF Executive Dashboard allows us to simultaneously look at developer responses to those that reported using Microsoft CE, ThreadX (chosen as a commercial RTOS example), Commercial Linux, Non- Commercial (free) Linux, and Open Source software.

From these filters we can interrogate the 2013 EMF Survey of Embedded Developers to determine:

- The number of software developers per project
- Average time from project start to shipment (time to market)
- Percent of developments completed behind schedule – and the delay in months
- Percent of developments cancelled – and time elapsed before cancellation

From these data we can calculate the average number of developers per project, the number of developer months lost to behind schedule and the number of developer months lost to cancellation. From these data we can calculate the total number of developer months per project. Now we can determine the comparative total cost of development between commercial RTOSes (Choosing ThreadX and Windows CE), commercial and free Linux and open source software developments.

Table I presents these data representing the broad industry (designs across all vertical markets).

2013 EMF Survey Data All Respondents	**Ind ave**	**ThreadX**	**Open Source**	**Comm Linux**	**Non-Comm Linux**	**Microsoft CE**
Devel time Months - Start to Ship	13.9	12.9	13.2	12.9	15.1	14.3
% behind schedule	47.0%	36.9%	45.1%	47.5%	46.7%	38.1%
Months behind	3.8	3.0	3.6	3.1	4.1	4.0
% cancelled	11.2%	13.4%	10.9%	11.7%	11.2%	11.5%
Months before cancellation	4.4	4.6	4.4	3.6	4.4	4.4
SW Developers/project	14.7	6.8	19.0	17.4	22.3	13.3
Average Developer months/project	204.3	87.7	250.8	224.5	336.7	190.2
Developer months lost to schedule	26.3	7.5	30.8	25.6	42.7	20.3
Developer months lost to cancellation	7.2	4.2	9.1	7.3	11.0	6.7
Total developer months/ project	**237.8**	**99.4**	**290.8**	**257.4**	**390.4**	**217.2**
At $10,000/developer month						
Average developer cost/project	$2,043,300	$877,200	$2,508,000	$2,244,600	$3,367,300	$1,901,900
Average cost to delay	$262,542	$75,276	$308,484	$256,215	$426,978	$202,692
Average cost to cancellation	$72,442	$41,915	$91,124	$73,289	$109,894	$67,298
Total developer cost/project	**$2,378,284**	**$994,391**	**$2,907,608**	**$2,574,104**	**$3,904,173**	**$2,171,890**

Table I: Worldwide Comparative Costs of Development

It is clear from Table I that developers using commercial OSes (ThreadX and Windows CE) got to market faster, required fewer developers per project, had lower costs of development and lower costs associated with delays than what was experienced with Linux, or Open Source developments. Also it has been clear, year over year, that commercial Linux developments cost less than non-commercial Linux developments.

Table I makes it clear, nonetheless that developers using RTOSes other than Microsoft CE create additional savings – in this case the use of CE was twice as costly as ThreadX.

Since many medial device developments are using ARM processors, we looked at ARM developments as presented in Table II.

2013 EMF Survey Data ARM Developers	**ThreadX**	**Open Source Software**	**Embedded Linux**	**ARM All Users**	**Microsoft CE**
Devel time Months - Start to Ship	8.8	12.4	12.6	11.9	12.8
% behind schedule	23.5%	41.0%	37.8%	43.4%	37.2%
Months behind	2.0	3.5	3.4	3.7	4.0
% cancelled	7.4%	11.1%	6.0%	9.6%	12.2%
Months before cancellation	2.3	4.4	4.6	3.9	3.9
SW Developers/project	5.7	9.4	5.6	6.7	12.7
Average Developer months/project	50.2	116.6	70.6	79.7	162.6
Developer months lost to schedule	2.7	13.5	7.2	10.8	18.9
Developer months lost to cancellation	1.0	4.6	1.5	2.5	6.0
Total developer months/ project	**53.8**	**134.6**	**79.3**	**93.0**	**187.5**
At $10,000/developer month					
Average developer cost/project	$501,600	$1,165,600	$705,600	$797,300	$1,625,600
Average cost to delay	$26,790	$134,890	$71,971	$107,589	$188,976
Average cost to cancellation	$9,701	$45,910	$15,456	$25,085	$60,427
Total developer cost/project	**$538,091**	**$1,346,400**	**$793,027**	**$929,973**	**$1,875,003**

Table II: Worldwide Comparative Costs of ARM Developments

Comparing Tables I and II, it is interesting to note that ARM developments are less costly than those that are industry wide. Yet here we see that embedded Linux (combining commercial and non-commercial) performs significantly better than Windows CE ARM developments. ThreadX is still 50% less costly than embedded Linux.

These results may not sit well with some medical device developers (usually those who's living and future are seemingly not predicated on the commercial success of their products) who will argue to the end that "free" software is preferred route. It is the fundamental tenet of capitalistic endeavors (sort of a technology Darwinian clearing house) that such arguments are settled in the technology marketplace.

So the question arises - why choose an RTOS that offers less attractive TTM and Cost of development? Perhaps it is because that many developers have not had access to the data presented herein.

The cost savings in development costs and time-to-market are but the tip of the iceberg. Being late to market increases development cost, but MORE SIGNIFICANTLY, being late to market reduces market share, and can make the difference between a profitable, successful product and an also-ran that loses money. The cost of being late to market far outweighs the development cost, and any savings from use of a free RTOS solution. The RTOS may be free but the total cost of development certainly is not. Relying on the free RTOS can be “penny wise and pound foolish.”

Model Driven Development (MDD)

Model Driven Development (MDD) is the latest improvement in the level of abstraction in writing software applications. Software development has moved from machine language to FORTRAN to C and C++ languages with the use of compilers, as well as to Java and to the SQL database. As we examine each step along the development levels of abstraction, we see that the higher levels of abstraction have offered significantly improved productivity and ease-of-writing applications. In such, developers can handle increasingly complex developments without increasing the development work load thereby enabling applications to be developed faster and less costly than with previous techniques. MDD is also capable of powerful simulation modeling and trade studies to compare design choice.

MDD separates the model from the code enabling the developers to work on a platform independent model. The auto-code generation capability writes the code according to the underlying OS and processor used which enables rapid prototyping of product subsystems.

EMF data has clearly shown that the total cost of development can be orders of magnitude *less* than acquisition costs in appropriate circumstances when MDD is employed:

- On time shipment of product – MDD developments ship faster than comparable developments not using MDD.
- MDD enables effective code reuse.
- Maintaining the expected performance, systems functionality and features and schedule of the development – Final design results are closer to pre-design expectations for MDD developments.

- Achieving market windows of opportunity – MDD developments not only get to market faster, but they can be easily upgraded to meet new market opportunities by integrating legacy software into new designs and automatically generating and deploying new code.
- Cost of re-design necessitated by changes in available hardware is minimal with MDD.
- Reduced project costs by finding problems earlier, when they are cheaper to fix. Previous research (Defense Systems Management College – 1993) has shown that after having spent just 15% of the development budget, 85% of the costs are already baked in. What costs X to correct in the early stages can cost 10X or more to correct later.
- Cost of in-field support is more effective because support personnel can more clearly understand design models and code versus just source code. MDD minimizes the possibility of product recalls.
- Documentation is automatic. The original developers may have retired but with MDD all of the documentation and interface information is retained. Imagine the cost of upgrade if all of the software apps had to be redone.
- CDRH audits are performed more effectively as the auditor can visually see the code executing and be able to ascertain that corrections have been completed. In cases where there has been a recall,
- MDD offers an effective and rapid method of correcting software errors. In such manufacturers can apply for a new and expedited 510k rather than having to wait up to 24 months for a new 510k.

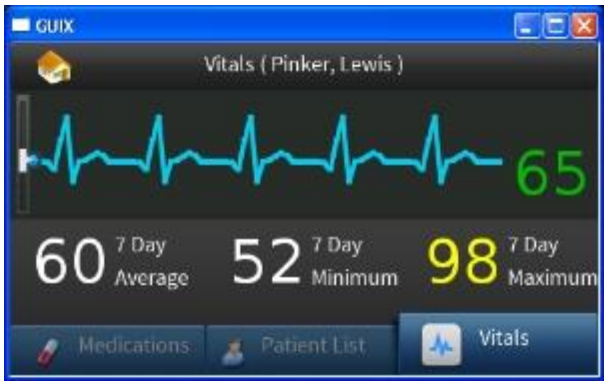
Enhancing the Design Process – looking at developmental efficiencies using wireless protocols, USB enablement and GUI development and integration

Integrated GUI Development

Thanks to the explosion of opportunities that started in smartphones, GUIs have become commonplace in medical, consumer, and industrial applications, prompting the need for advanced tools to simplify their development. Most embedded system programmers are not LCD specialists and do not want to program these displays at the lowest level, which involves constructing individual graphical shapes and objects ("widgets"). Instead, to speed time to market, most developers use a library of routines that manage the GUI details. Express Logic's QUIX development framework allows developers to describe the widgets at a high level of abstraction. Such libraries help developers to enhance programming productivity and avoid many errors. GUIX offers developers an advanced UI framework and rich library of unique widgets tailored to help them construct whatever GUI they envision. Programmers can call GUIX functions from their C application programs, and GUIX performs all the necessary drawing functions to produce a clear, interactive GUI on LCD screens of various sizes and resolutions.

GUIX is a small-footprint, low-overhead runtime engine and development tool featuring automatic code generation for embedded systems capable of graphical display. GUIX simplifies GUI development and targets the ARM 32-bit MCU and MPU architectures, including Cortex-M3, M4, A8, and A9, in medical devices, consumer electronics, and industrial control equipment.

Interestingly, GUI designers can create GUIs using GUIX Studio™, the companion PC-based application that enables WYSIWYG rapid prototyping of GUI designs (as shown in the Figure). With GUIX Studio, the designer can select, drag-and-drop, and resize images, backgrounds, widgets, and other elements of a powerful GUI without having to write a single line of code. GUIX Studio generates the code necessary to implement the exact GUI design constructed on the PC. The generated code can be dropped into the application and executed on the target system.



There is no longer a need for developers to waste time on implementing GUIs when an effective and easy to integrate means is available. It makes sense that such a tool can enhance a development’s time-to-market.

Medical Device Developer’s Use of Wireless Protocols

An analysis of data taken from the 2013 EMF Annual Survey of Embedded Developers enables us to compare wireless choices of medical device developers with wireless use across the embedded industry.

Tables III and IV present developer responses indicating a comparison between medical and other embedded developments for “currently designed-in” and for “plan to use in 2014” developments.

EMF cautions the reader that what developers indicate they “plan to use” frequently does not match what they actually use in the following year. Nonetheless, it is interesting to report on what they anticipate using in 2014.

Currently Designed In	**Industry**	**Medical**
Bluetooth Classic V2.1	17.6%	35.4%
RFID	13.4%	29.3%
802.11g	23.7%	28.0%
Zigbee	21.8%	24.4%
802.11b	17.4%	19.5%
HTTP	17.2%	19.5%
Bluetooth Low Energy V4.0	6.5%	18.3%
802.11n	14.5%	17.1%
GSM	13.8%	15.9%
802.11a	14.7%	14.6%
XML	11.1%	14.6%
IrDA	8.4%	13.4%
Proprietary	11.7%	13.4%
3G	18.2%	12.2%
Bluetooth High Speed V3.0	8.0%	12.2%
NFC	6.3%	9.8%
WPA2	9.2%	8.5%
WPA	7.5%	7.3%
WEP	6.7%	6.1%
802.11i	5.9%	4.9%
4G	6.9%	3.7%
CDMA	8.6%	3.7%

Table III: Comparative Current Wireless Use

A side by side inspection of Table II shows the preponderance of Bluetooth use (all versions), WiFi, and Zigbee. We report WiFi separately by protocol, and Bluetooth by version, rather than cumulative, whereas the ZigBee is reported as cumulative. ~~since developers seldom use more than one version (e.g., 802.11g developers wouldn't also use 802.11n~~). RFID use was interesting to observe. However, RFID ha~~i~~s been used predominantly to control inventory and surgical use rather than for patient monitoring applications.

Of interest is the use of Zigbee by developers. The frequency of use is comparable between embedded industry use and medical device developments. ~~Yet t~~The 24.4% reported use in the last year is significant however the plan to use shows that ZigBee use is dropping off and ~~. We say that notwithstanding feedback from a prominent supplier of~~ both Bluetooth and~~,~~ WiFi ~~and Zigbee stacks (Clarinox) that Zigbee sales are dropping while Bluetooth sales~~ continue to bloom.

Plan to use in next 12 months	Industry	Medical
Bluetooth Classic V2.1	14.2%	28.8%
Bluetooth Low Energy V4.0	14.2%	28.8%
802.11g	18.7%	24.7%
802.11n	15.9%	20.5%
802.11b	10.7%	19.2%
RFID	9.2%	17.8%
Zigbee	14.7%	17.8%
802.11a	8.2%	13.7%
HTTP	13.2%	12.3%
Proprietary	9.2%	12.3%
Bluetooth High Speed V3.0	7.0%	11.0%
IrDA	4.5%	9.6%
NFC	5.2%	9.6%
3G	11.4%	8.2%
WPA2	6.5%	8.2%
GSM	7.7%	6.8%
WEP	3.2%	6.8%
XML	7.2%	6.8%
4G	8.0%	4.1%
802.11i	3.7%	2.7%
LTE	4.2%	2.7%
CDMA	4.0%	1.4%

Table IV: Comparative Anticipated (2014) Wireless Use

Examining Table IV we can see from developer responses the drop in anticipated Zigbee use which is consistent with what Clarinox has reported to us.

It is interesting to see that Bluetooth use (and anticipated use), as reported in Figures III and IV seem to be consistent. The EMF takeaway is that Bluetooth users (for Classic, Low Energy and High Speed) are satisfied with the performance, cost and associated implementations and will be unlikely to change to another protocol.

RTOS and other vendors be aware – it is a good investment to integrating and support Bluetooth as part of your OS offering as a means to enhance the design experience of your customers and prospects. Bluetooth use among embedded developers is high and enabling developers to easily integrate it into current designs. Clarinox, by offering Bluetooth and~~,~~ WiFi~~ and Zigbee capabilities~~, has established partnerships with several prominent RTOS vendors.

Table V presents a listing of criteria deemed most important for selecting Bluetooth protocol stack.

Most Important Selection Criteria	**Industry**	**Medical**
Microprocessor support	42.5%	42.5%
Reliability	31.5%	40.0%
Compatibility with our development tools	37.0%	37.5%
Availability of source code	37.8%	35.0%
Real time performance	27.6%	35.0%
Acquisition cost	44.1%	32.5%
Quality of support	18.1%	32.5%
Host platform support	22.8%	30.0%
Includes good development tools	21.3%	27.5%
Supports easy porting of existing software	8.7%	20.0%
Memory constraints	13.4%	17.5%
Compatibility with suppliers and vendors	13.4%	15.0%
Must be free (for both development and production)	26.8%	12.5%
Performance on internal evaluation	11.0%	12.5%
Provides device drivers and/or Board Support Packages	15.0%	12.5%
Royalty cost (production licenses)	12.6%	12.5%
Security	10.2%	12.5%
Customer approved or specified	9.4%	10.0%
Must be open source	12.6%	10.0%
Safety certifiable (DO-178B/C, IEC 61508, FDA, etc.)	5.5%	7.5%
Availability of perpetual license	22.0%	5.0%
Availability of professional services (porting, integration, or qualification)	13.4%	5.0%

Table V: Reasons for Selecting a Bluetooth Stack

From Table V we can see that the Bluetooth acquisition cost is less important for medical developers and that they don’t expect any freebies. So look out chip providers that include a Bluetooth stack with the processor. Developers want more flexibility and are willing to pay for it. Also, the availability of a perpetual license is not an issue for medical applications (only 5% report this to be a concern, compared with 20% for the embedded industry).

Issues of importance to medical device developers are realtime performance, quality of support~~ (this is why RTOS vendors are better suited to offer an integrated Bluetooth package)~~, reliability and host platform support. On these points the robust stack offering from Clarinox with support for multiple RTOS vendor products is well aligned to the needs of the medical industry.

EMF asked medical device developers what aspects of their protocol software selection process turned out to be the most disappointing.

Table VI presents their responses which are compared to respondents from the broad embedded industry.

Most Disappointing Aspects of Protocol Software Selection

	Industry	Medical
Expectation of better support	45.8%	50.5%
Better technical solution	45.8%	47.3%
Product integration	40.6%	47.3%
Better price	26.3%	23.1%
Open Source	14.4%	16.5%
Compatibility with customers and suppliers	20.0%	14.3%
Vendor reputation	5.2%	4.4%
Ease of purchasing	7.7%	3.3%
Corporate standardization	8.5%	3.3%

Table VI : Most Disappointing Aspects of Protocol Selection

Wireless - EMF's Takeaway:

Of the short range wireless technologies Bluetooth and WiFi are the most adopted by medical device developers. This gives them both the opportunity to interact with a large number of existing devices and infrastructure.

WiFi is used for higher data rates and longer distance but is more susceptible to co-existence issues. Bluetooth caters for lower data rates/shorter distance (than WiFi, higher data rate/longer distance than RFID) but the technology is less affected by large amounts of RF transmission within close proximity and the battery lasts longer.

A key requirement is reliability – and some protocol stack software is more robust than others. Clarinox is the partner of choice of all the leading RTOS vendors due to the superior robustness and reliability of the Clarinox Bluetooth and WiFi stacks. *By supporting Clarinox stacks into the specific RTOS, the design experience is enhanced for medical device developers and it is easier to achieve better time to market*.

Bluetooth from the module or chip vendor will provide minimal standard functionality – but for anyone that seeks faster pairing, maximized performance and faster data rates, plus a larger range of applications – then a dedicated stack vendor is typically preferred~~required~~.

What medical device developers need to consider when choosing a wireless technology

- Range – different technology have different range of transmission
- Data rate – how much data needs to be transferred?
- Battery life – is the device battery operated? How long does the battery need to last
- Latency – how quick does the connection need to be
- Robustness – medical devices cannot afford to require frequent reboot
- Wireless technology standards – is it important to interact with other devices such as phones or tablets? If so then a standard technology that is included on those devices should be used
- Regulations – regulations around the use of RF within some medical environments will be controlled, this level of control may vary with between different governing bodies (jurisdictions)
- Coexistence – increased radio frequency “traffic” increases the chance of the RF equivalent of “grid lock” – some technologies are more effected than other
- Security – privacy and security of data concerns
- Electromagnetic Compatibility (EMC) evaluation requirements for a number of devices such as active implantable cardiovascular devices that provide one or more therapies for bradycardia, tachycardia and cardiac resynchronization.

Medical Device Developer’s Use of USB

Table VII presents, based on the results of the 2013 EMF Survey of Embedded Developers (667 respondents), the driver interfaces used by medical device developers compared with the overall embedded industry.

Driver Interfaces	Medical	Industry
USB	63.2%	50.1%
I2C	62.3%	52.3%
RS232 (serial)	53.8%	55.2%
TCP/IP	48.1%	51.6%
SPI	45.3%	37.9%
100Mbps Ethernet	44.3%	47.5%
IPv4	36.8%	37.3%
CAN	34.0%	33.8%
10Mbps Ethernet	31.1%	30.9%
HTTP	31.1%	29.2%
UDP	24.5%	30.9%
DHCP	22.6%	20.0%
DMA	22.6%	16.9%

Gigabyte Ethernet	21.7%	24.4%

Table VI I: Driver Interfaces Used

In 2013, USB remained the leading choice for driver interface use among medical device developers, substantially higher than TCP/IP and RS 232 (serial). The question arises as to whether USB is a better choice from a cost containment viewpoint. Such factors as time-to-market, number of developers required per design, cost of delays (behind schedule and cancellations) contribute to the cost of development. Table VIII presents a development cost comparison between USB, TCP/IP RS232 and Ethernet.

	Ind ave	USB	TCP/IP	RS232	Ethernet
Devel time Months	13.8	13.6	13.2	14.1	13.7
% behind schedule	35.2%	38.1%	38.5%	37.7%	36.9%
Months behind	4.4	3.2	3.9	4.7	3.8
% Cancelled	10.0%	9.5%	9.5%	9.7%	9.3%
Months before cancellation	5.1	4.6	4.6	4.8	4.9
SW Developers/project	11.4	6	11.6	9.1	12.4
HW Developers/project	7.4	3.7	5.1	5.7	5.5
Total Developers/Project	18.8	9.7	16.7	14.8	17.9
Average Developer months/project	259.4	131.9	220.4	208.7	245.2
Developer months lost to schedule	29.1	11.8	25.1	26.2	25.1
Developer months lost to cancellation	9.6	4.2	7.3	6.9	8.2
Total Developer months lost cancel/delay	38.7	16.1	32.4	33.1	33.3
Total developer months/ project	298.1	148	252.8	241.8	278.5
At $10,000/developer month					
Average developer cost/project	$2,594,400	$1,319,200	$2,204,400	$2,086,800	$2,452,300
Average cost to delay	$387,054	$160,651	$323,730	$331,150	$332,564
Total developer cost/project	**$2,981,454**	**$1,479,851**	**$2,528,130**	**$2,417,950**	**$2,784,864**

Table VIII: Worldwide Comparisons –USB, TCP/IP, RS232, and Ethernet

USB developments have a cost of development advantage of nearly 67%.

EMF asked responding developers to report on the factor that most impacted their decision to buy embedded products and tools.

USB developers are significantly more sensitive to value of tools, to ease of use, and to speed performance. Interestingly, they’re not as interested in “compatibility”.

Again, intuitively, this makes sense. The cost of tools is a larger fraction of the project cost, because labor input is lower. Ease of use is important, because there are half as many developers on the project, compared to industry average. Compatibility is not as important, because the project tools do not to be amortized over multiple products. Speed/performance is important, because (during development) this means less time spent by developers waiting for the tools.

By providing a range of choices and limiting responses to choosing only four characteristics, we are able to create a listing of the factors most important in their decision making process. The result is presented in Table IX.

In general, what FOUR characteristics are the most important to you in buying embedded products and tools?

	Industry	**USB**
Price/cost/value of product	66.3%	72.0%
Ease of use of product	59.2%	65.8%
Quality and reliability of products	48.5%	51.9%
Technical support	41.3%	44.0%
Compatibility of products	43.3%	38.3%
Speed/performance of products	31.0%	35.4%
Reputation of supplier/vendor	16.3%	18.9%
Leading edge technology	16.7%	13.6%
Ease of dealing with vendors& processes	8.3%	9.5%
Personal trusted relationship to rep or support people	8.1%	6.6%
Sales service and support	6.5%	5.3%
Other (please specify)	2.5%	1.6%

Table IX: Most Important Characteristics used in Buying Decision Making.