# The Embedded Muse 136

Editor: Jack Ganssle (jack@ganssle.com) November 9th, 2006

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

## Editor's Notes

Did you know it IS possible to create accurate schedules? Or that most projects consume 50% of the development time in debug and test, and that it's not hard to slash that number drastically? Or that we know how to manage the quantitative relationship between complexity and bugs?

I'll cover all this and much more December 1 in Santa Clara, California, in my "Better Firmware Faster" seminar. Check out http://www.ganssle.com/classes.htm for details. The site is just a few miles from San Jose airport, which is inexpensively served by Southwest Airlines and others. Come for the seminar on Friday and explore San Francisco over the weekend! Better yet, check out the Computer History Museum in Mountain View.

Do you have a dozen or more developers who could benefit from this firmware training? I can present my Better Firmware Faster class at your facility. See http://www.ganssle.com/classes.htm.

Embedded Systems Design has an interesting article about using ints safely in C/C++. That would seem a minor subject but the piece is insightful and worth reading. See http://embedded.com/showArticle.jhtml?articleID=193501786&pgno=2.

There's more on CERT's Secure Coding page (http://www.cert.org/secure-coding/ ). If you follow the links you'll find vulnerabilities in a number of commercial compilers and other tools. I found none listed for embedded tools, suggesting the vendors are doing a fantastic job… or that, as usual, embedded is a stealth industry.

Scott Rosenthal sent along this link (http://qshc.bmj.com/cgi/content/full/14/3/212 ) about audible alarms in health-care systems. Some of the advice is applicable to anyone working with those all-too-often annoying bleepers.

## Reading Code and Beautiful C++

I asked if anyone had links to stunning C++ code. Bandit wrote:

```
“void main(void)
{
     cout << "hello world";
}
```

I think it goes downhill from there."

That could have been the Joke for the Week!

Wayne Johnston submitted: "Regarding reading code, I think Martin Fowler said it best. 'Any fool can write code that a computer can understand. Good programmers write code that humans can understand.' The quote is from one of his books – I can't remember which one just now."

Peter Miller responded to this:
> When you write a program, what you're really doing is communicating
> to another human what it is you want the computer to do.  Programming
> is about communication, just like any other form of writing.

"I agree with your reader about the readability of code, but I have a small quibble: if the human writer manages to successfully convey to the human reader what he *wanted* the code to do, but the compiler actually did something else, you have a bug.  This is why reading our own code to find bugs is so hard - we see what we wanted, not what *is*. The purpose of code readability (driven largely by the white space the compiler throws away, and the choice of symbol names which the compiler abstracts away) is to tell the reader what the compiler *will* do to the code. Code formatting decisions must be ruthlessly and consistently driven by language structure and semantics if they are to accurately convey the effect of the code to a human.

> Several readers wondered if there was a good place to find truly
> beautiful C++ source to read. I'm stumped. Any ideas?

"I have a 25 year repository of just about all the code I have ever written (sadly, my SC/MP code was all hand written, and hand assembled, and is now long gone). Not all of my archive can be shared with you as some of it belongs to my employers, but a very large amount of it is open source. See the URL in the sig block, below, for links.

"Not all of that code archive is what I would today call beautiful, and some is presently in transition from pretty good C into will-be-better C++, but with over 1 million lines of my code out here, it takes a long time. However, I am quite happy with the USCD Pascal p-Machine cross compiler I am writing. It's written in C++. (It's a long story, a trip down memory lane (in aforementioned archive) turned into some serious retro-computing.) For the source code see http://ucsd-psystem-xc.sourceforge.net. Code reviews are always welcome."

## Writing the Manual First

Last issue Xtian Xultz wrote about how he has learned to write the user's manual before generating the code. Justin Phillips sent this: "The Mythical Man Month deals with this kind issue in detail. What Xtian was alluding too was that the product needed an architectural specification at the outset to give 'conceptual integrity'. I worked on a project in my mainframe days where the user manual eventually became the system specification!"

David M said: "I had the same experience... After a disastrously complex GUI product laden with "Feature Creep", I asked the program manager to write the manual first on the next product with a complicated GUI. We wrote the manual first (or at least keeping one feature ahead of the SW development) complete with screenshots and everything. The SW development was a dream, the testers loved it because they could test to the users manual (enhanced in house with more in depth explanations), and the tech writers loved the fact that their job (writing the manual) was 90% done, and we didn't have to slip production to wait for the manual.  Plus during development the Marketing and Sales guys had access to (but not control of) the manual for their input.  The end product, if not everything they wanted, was exactly what they expected, and they were happy."

## New Kinds of Debugging Tools

Last issue I wrote a bit about static analyzers, and asked for your input on these tools. A couple of readers sent in the comments below. To be fair I invited Steve Barriault of PolySpace to respond to the responses; his comments follow as well.

Vlad Zeylikman wrote: "I had experience with PolySpace about 4 years ago.  The system promised a lot but delivery was another story.  The code it was given was pretty bad: convoluted functions, some 5-6 screenloads long, half a dozen tasks interacting with each other via messages (mostly).  In some cases, they communicated via globals protected by disable/enable interrupt pairs.  The code needed analysis and refactoring as it suffered from unexplained random hangs.  The hope was that PolySpace would pinpoint the problems, we'd fix them, and this would buy us time to do a real refactoring and optimization.

"Well, PolySpace tried.  They gave us a PC on which we ran the analysis and it did not go well.  So, they offered to send our code to their Grenoble headquarters where they have a server farm - it was a matter of pride and they wanted to show that they are up to the task.  Our code sample hung their servers.

"PolySpace rep was a nice and a diligent man and I worked with him and the tech support guy and in the end we decided that it was taking too much time.

"I refactored the code, removed the globals, etc., etc.  The system got better and faster. And the code became readable.  A year hence, we got a call - they ran our old code through their servers!  The rep realized that it did not matter to us but wanted to show that they are getting better and maybe they should be considered after all.

"My impression after all this is that it can be a useful tool but when the code is bad - high cyclomatic complexity, lots of variables in the scope, and complex interactions - it becomes too taxing for PolySpace.  It might be much better now, I hope it is: they are improving and the hardware is faster now.  But when the major surgery is in order - do it and forget the analyzers."

From Justin Phillips: "I've used Parasoft's Insure++ to perform static and dynamic analysis of embedded code in a host environment. It's a good tool which uncovers all of the usual "features" and more besides. The runtime analysis feature also provides code coverage figures which can be helpful(!).

“My reservations about it were that it had setup costs (time), also code instrumentation degraded runtime performance (expected) and was resource hungry, which affects usability. What would have been useful was a runtime tool usage detection function in the tool’s API, which is a feature Rational's Purify plus. Which is useful for determining which memory allocator to use.

“As far as lint goes, flexelint is lint on steroids.

“With all of the above tools, the biggest pain is ensuring that that they are fully integrated into the software development process that's hopefully followed. Also their continued usage needs evaluation and review i.e. sharper tools are always required.”

A reader who wishes to remain anonymous wrote: “We are using PolySpace here with good end results. However, it is a beast to run reliably on Windows, and the results are often difficult to interpret. Also the analysis is very slow (longer than the time between releases for us), and incredibly the server product cannot be sped up by adding processors. The tool is only able to verify subsections of code, not the whole system (which is apparently too complex for PolySpace to handle). It does not cope well with autogenerated code.

“The key to success is automating the submission of code into the tool, and investing time implementing the many tweaks that PolySpace recommend to improve accuracy. Out of the box, the current version (3.3) produces too much "noise" for many developers to take it seriously. After the tweaks and automation, it all becomes a lot easier to deal with, and it does find real and potential run time bugs!

“So: not happy with the setup pain and quality of the GUI which is too much and too flaky on windows at least. All the obvious errors are found, and so far (2 months, 4 releases) no errors that "should have been found" have crept through.

“Any bug is potentially catastrophic. So I guess they are worth the price if you have the money.”

Tormod Tjaberg commented: “We had a meeting with PolySpace a few months ago where they were showcasing their product.

“The presenters were not very knowledgeable about the product. When it came to the actual licensing and pricing we could not understand why they needed a special server where they uploaded the code to and it performed the analysis.

“We gave them some code and after some hassle they managed to start the analysis. They discovered one trivial bug. But at that point I telnetted into the work environment and ran flexlint which in < 10 seconds found the same bug.

“The product had a hard time coping with a mix of C & C++.

“And we could not get it to analyse anything on our current Linux project. There were bugs in PolySpace itself which choked on the kernel include files.

“I would not say they are selling snake oil but their emphasis is convincing non IT savvy managers that this is "the silver bullet".

“I would use Flexlint, it takes some time to set up but when it's running it's fast. It also does multiple passes and code analysis.”

Matthew MacClary uses these checkers:
“1) gcc -Wall -O3 (sometimes with a -pedantic)

“2) splint (http://www.splint.org/) is designed to be a better lint
with a security focus, the theory being that broken code is
insecure code.

“3) valgrind is excellent http://valgrind.org/, it is actually a
dynamic checker though. I implement the core of my functionality
for embedded systems as regular Linux programs, and then just pass
a define variable to the compiler when I want to to pull in the
microcontroller specific functionality. This systems lets me
use every tool that is available for Linux development.

“I am happy with these tools. I use gcc with lots of checking turned on all the time, the other tools I just use on hairy problems - though making them part of my standard flow would be great too.

“The key errors that aren't checked are asynchronous / interrupt handler type problems. I rely on coding best practices like reliable debouncing routines, ring buffers, and double buffers as appropriate to design this type of asynchronous code correctly from the start.”

Steve Barriault from PolySpace had this response: “There are a couple of things that I would like to address:

“1- The main difference between our tool and others is that our analysis technique is exhaustive (we actually analyze the whole range of values different variables can take at different points in the program). It can be difficult to believe, but PolySpace can actually point out in your code which operations will, may or will never return a run-time error. That is clearly unique, because traditionally, static tools can only tell you where errors may lie, but they can't certify which ones are 100% safe.

“2- Obviously, there is a trade-off here. Since our products are exhaustive, they involve a lot of calculations and eat some serious memory space. So we analyze less lines of code and take more time to do it. Also, for very complex constructions or messy code, our algorithms might not be able to make heads or tails of it. Because we want our "green" diagnostics (safe operations) to mean something, we have to flag these operations as possible faults.

“Our own experience recoups a lot your first reader's - adding processor power brings a marginal performance improvement, but adding memory makes A LOT of difference. We currently recommend to install our server on Linux (clients may be on Windows), but we are also investigating how to make the product work faster on Windows.

“3- Is the analysis worth the wait? Our experience shows it depends on the user's need. If you are in a field where a high level of reliability and robustness is needed, PolySpace is generally a very good fit, because our tools tell you where errors can hide, and where they won't. In these industries, if you just obtain a list of bugs without guaranteeing the rest is safe (which is the case with standard static analyzers), then you have to treat all the lines of code as suspect - ultimately, that means more code review, test cases, or (unused?) defensive code.

“This is why PolySpace has been used extensively in fields where run-time errors are a major hurdle - Auto, Defense, Aerospace, Medical Devices, etc. But again, it really depends on the users' goals and objectives.

“4- Analyzing smaller chunks of code at a time does not need to be a bummer. An increasing number of clients use our products during development (by the developers themselves, or by a testing team at unit level). Their logic is simple: find errors as soon as you make them, so you can correct them right away, when the code is not too complex. PolySpace also provides these clients with feedback about which types of code structures and styles are more prone to errors than others, so developers use that feedback to improve both the reliability and the robustness of their code. This is exactly the phenomenon that has been described by one of your readers. Once the code is very robust, it can be analyzed one last time at an integration level to point out integration bugs (for example, concurrent access to shared data).

“We do also have clients that use PolySpace to improve the reliability of existing systems - our clients used our products to analyze hundred of thousands of lines of code, even if they sometimes have to split the analysis into two or three subsystems.

“5- We are investing a lot in our R&D effort, and I guess this is reflected in our pricing structure, as pointed out by your readers. This is what enables us to keep innovating and improving the product. For example, we were able to optimize our latest C++ release so as to speed up analysis time considerably. Likewise, your first reader might be interested to know, in an "avant premiere", that our 4.1 release (scheduled around April 2007) will include improved usability and tools to help users review different categories of checks. There are a number of other innovations in the pipeline.

“Ultimately, pricing must also be compared to what you stand to gain. A PolySpace client-server configuration is able to service many engineers, and the purchase price is the cost of ownership (while some other companies license their products on a yearly, number-of-lines basis). And what is the price of bugs? What is the price of doing more traditional test cases or more lab testing when run-time errors are detected through these means? And, what is the price of a bug in a release? Automotive organizations know this problem all too well (recalls)... So again, this is something that depends on your readers' situation.

“Finally, as you mentioned before on Embedded.com, organizations have traditionally invested a lot of money on hardware testing. Software is increasingly complex and vital to a number of systems, and software bugs are just as critical and as complex to find than hardware bugs. Just as in hardware, developers and testers need to have access to a variety of industry-grade tools that will make their work both efficient and high-quality.”

## Tools

Editors are a hot button for all of us. Four folks wrote in with their preferences:

Justin Phillips seconds many who have lauded Ultraedit: www.ultraedit.com - “A fully featured editor with built diff tool and loads of other neat features. And crisp (www.crisp.demon.co.uk) - A sophisticated editor for both Linux and Win32. Too complex for my taste but take a look.”

Roland Bennett likes PSPad: “I really appreciate the feedback from users regarding their favorite editors. Having used Visual SlickEdit for many years, I was looking for a free editor when I left the company where I had used it.”

Vlad Zeylikman is using Epsilon from Lugaru, Inc (http://www.lugaru.com ). “It’s not free but whenever someone names a new cool feature, Epsilon had it for years. I find that it's well worth the price: loaded with features, Windows/Linux/DOS in one package.”

Jon 'Far' McKamey writes: “Another good choice is MadEdit. (http://madedit.sourceforge.net ). I'm a bit of a nut about finding the right editor, and I'm toggleing between this and Notepad++ at the moment. “

Gergely Budai wrote: “There is yet another tool that needs to be mentioned in my opinion. (Source navigator / Source Documenting).

“In our company we need to maintain pretty much Assembler sources, so we started to look for something that fits for our needs. I stumbled over the tool called ROBODoc (http://robodoc.sf.net) which we started to use and to extend.

“Since it does not know anything about the language it is documenting, it's pretty useful for less structured ones (like assembler) or to document whole projects (various languages and makefiles, etc).

“Some Pros:
- Can be used with any language that supports remarks
- Can be used to document whole projects
- Since one header can have multiple names it is great to document assembler code (subroutines with multiple jump-in points)
- Absolutely free
- Source highlighting
- Multiple outputs (html/xml/latex/rtf/ASCII)
- User defined objects (like functions, variables, etc)

“Some Cons:
- No call graphs or anything like that (since it does not know anything about the language)
- False links if someone uses "bad" names like ‘use’, ‘do’, ‘copy’”

David J. Scott likes WinMerge: “WinMerge is simply awesome. It lets you compare two files side-by-side and do simple editing. I don't know what I'd do without it. It’s free at http://winmerge.org/

"From the website: Visual differencing and merging of text files Flexible editor with syntax highlighting, line numbers and word-wrap Handles DOS, UNIX and MAC text file formats Unicode support Difference pane shows current difference in two vertical panes Location pane shows map of files compared Highlights differences inside the lines File filters in directory pane Moved lines detection in file compare pane Shell Integration Rudimentary Visual SourceSafe and Rational ClearCase integration Archive file support using 7-zip Plugin support Localizable interface via resource DLL HTML-based Manual."

## Jobs!

Let me know if you're hiring firmware or embedded designers. No recruiters please, and I reserve the right to edit ads to fit the format and intents of this newsletter.

Draper Laboratory located in Cambridge is an independent research institution that incorporated out of MIT. Draper has the freedom to work on problems that are too risky or too early to attract commercial industry. After demonstrating that a system solution is feasible, we complete our mission by transitioning the technology to industry for volume production. www.draper.com There are openings for Embedded Software Engineers.

Candidates will be expected to work on varied applications which include micro-electronics systems wearable by soldiers and integrated on unmanned air and ground vehicles. Other applications include avionics for guided airdrop systems. The candidate is expected to have strong software development skills and familiarity with embedded processors. A B.S. or M.S. in Computer Science, Electrical, Mechanical, or Aerospace Engineering is required. Demonstrated embedded software development skills through hands-on projects using the C language is desired.

To apply send a resume to nbeaumont@draper.com

## Joke for the Week

.
From Catherine French:

Whose lines these are I think I know.
He's on another project though;
He should not mind my pausing here
To clarify the logic flow.

... though Management does NOT appear
To sanction major changes here.

"Fix it not 'fore it shall break"
And Beta Test is looming near.

Still, best case good ideas enmeshed
With side effects will fail the test
Of year two thousand (coming soon).
Change just two lines, and then I'll rest.

Half vast possibilities run deep
But I have promises to keep
And lines to code before I sleep,
And lines to code before I sleep.

(With apologies to Robert Frost)

## About The Embedded Muse

The Embedded Muse is an occasional newsletter sent via email by Jack Ganssle. Send complaints, comments, and contributions to him at jack@ganssle.com.

To subscribe, send a message to majordomo@ganssle.com, with the words "subscribe embedded *your-email-address*" in the body. To unsubscribe, change the message to "unsubscribe embedded *your-email-address*". ". BUT - please use YOUR email address in place of "email-address".

The Embedded Muse is supported by The Ganssle Group, whose mission is to help embedded folks get better products to market faster. We offer seminars at your site offering hard-hitting ideas - and action - you can take now to ***improve firmware quality and decrease development time***. Contact us at info@ganssle.com for more information.