
Embedded Muse 182 Copyright 2009 TGG October 5, 2009
Permanent link to this issue: www.ganssle.com/tem/tem182.pdf

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:
- Editor's Notes
- Quotes and Thoughts
- Too Much Optimism
- Tools and Tips
- Naming Conventions
- Joke for the Week
- About The Embedded Muse

Editor's Notes

Are you happy with your bug rates? If not, what are you doing about it? Are you asked to do more with less? Deliver faster, with more features? What action are you taking to achieve those goals?

In his book "Quality Is Personal" Harry Roberts shows that "trying harder" won't change anything fundamental. More substantive changes must be made, and in fact it IS possible to accurately schedule a project, meet the deadline, and drastically reduce bugs. Learn how at my Better Firmware Faster class, presented at your facility. See http://www.ganssle.com/classes.htm .

For the first time in half a year, last week's Muse had quite a few job listings. One came in during the last two weeks. I wrote about the job market last week (http://embedded.com/columns/breakpoint/220300065) and am shocked at the poll results (http://embedded.com/pollArchives/showPoll.jhtml?surveyno=280301002) which suggest few of us work diligently at maintaining our resumes. A well-prepared resume is the key to getting an interview. It should be a living document that's constantly being tuned.

Quotes and Thoughts

Harold Kraus wrote: "I have added a Blaise Pascal quote to my collection of software engineering 'laws' that I don't think was not originally stated as "software" truism: "I have made this [letter] longer, because I have not had the time to make it shorter." Here, Blaise Pascal was apologizing for not having the extra time needed to express his thoughts more concisely. This applies to engineering design, including software. Obviously, one can design (i.e., code) much faster (read cheaper) when one is careless, but careless designs are much more expensive to reuse as well as to prove correct than careful designs. The investment in careful design can be returned in part or whole through reduced verification costs.

"It takes more time to decompose a large problem in to many carefully isolated small components. However, getting the code *right* and *proving* it is right happens faster if the software is composed of carefully isolated small components."

Too Much Optimism

"I know it's male, Dad", my daughter exclaimed, pointing at the viciously-snapping Blue Crab we had just landed in a bucket. I wanted to teach her how to tell the sex of this so-delicious product of the Chesapeake Bay by examining the shape of the creature's shell. "How do you know that, Kristy?", I wondered, my scientific description momentarily preempted. The crab reared back, snapping, angry, and aggressive. "By his attitude," she replied.

Later, pondering this conversation and her surprisingly deep insight into the male psyche, I thought about a chunk of code I'd recently read. (I know, I know, the peril of being a dweeb is we see everything in terms of our work). It was a pretty typical bit of firmware, peppered with both the usual flashes of brilliance and egregious flaws.

This firmware used quite a bit of dynamic memory management. malloc()s and free()s abounded. Some developers consider the use of anything other than statics a sin, but it is possible to write really great embedded code with either sort of memory management. But this particular code, like every bit of similar firmware exploiting dynamic memory allocation, was clearly written by a guy. No - there was no clue to the person's sex in the comments, the author's first name nothing but an utterly unrevealing initial. His testosterone came across in the same way exhibited by the crab. Whenever memory was needed, he issued a malloc(), and by God, that was a demand from this programmer to get some memory! Gimme some memory, NOW!

A gentler and more robust approach might be to recognize that malloc() has a return code. It might fail. Check the return code and take action if the system is low on resources. Demand memory? No, request it, with a sort of virtual "please", and recognize that it's possible the system will reply, "sorry, there's none available now".

I read a lot of code. A lot. And I've learned that we developers are an optimistic and an aggressive lot, blithely expecting that the computer will always respond in the way we

hope. The program will always run just as we decided it would. Nothing will fail (other than hardware, of course). Code, being deterministic, will always work properly once we pronounce it "done".

But it just ain't so! I collect embedded disaster stories, and the most common theme that runs through them all is poor or no exception handling. Everything runs fine until something rather simple goes wrong, the exception handler gets called, and (if it exists at all), it's poorly-thought-out, buggy, or totally inadequate, so the entire system crashes. Exception handlers are our last-ditch chance to save the system. Treated as afterthoughts they're sure to exacerbate the problems.

We've learned that datacomm is unreliable, so use TCP/IP which is robust. It'll transfer correct data despite missing and corrupt packets. Yet a friend's autopilot passes data between boxes using a proprietary comm protocol that tolerates no errors. Press a nearby radio's transmit button and the data link gets scrambled, crashing the code. We know that it's a disaster to pass a routine data which is out of range, yet rarely do I see debugging snippets (like assert macros) embedded to catch these common problems; when such constructs do appear they're usually slammed in as acts of panicked desperation. On processors with divide-by-zero traps I nearly never see a handler for this condition, despite the curse such divisions have had on the computer industry for 50 years.

Unexpected stuff happens. Users do astonishingly foolish things. Complex interactions are the norm, not the exception; none of us are smart enough to predict all possible paths.

Why do we persist in writing such fragile systems? When do we accept the fact that failure is normal, weird things happen regularly, and it's our responsibility to catch, process, and safely dispose of such conditions?

Tools and Tips

Ed Sutter wrote: "One other tool, by the way, that I find quite useful for viewing and/or modifying binary files is "frhed" (free hex editor). It's available at: http://www.kibria.de/frhed.html .

"It does have the disadvantage of being Windows-only (as far as I know); but if you're in that space, then this is a nice one to add to the arsenal."

Carl VanWormer sent this: "Your mention of Snagit ($50) prompts me to suggest my new favorite, Picpick (http://picpick.wiziple.net/), which is free, and has more features than my previous favorite of the last 10 years. My preferred default is to capture a mouse defined rectangle after hitting the print-screen key. The resultant rectangle pops into an image editor for any pretty arrows and notes, with that result able to be printed or saved as a graphics file. This file was brought to my attention by Gizmo

(www.techsupportalert.com), where I have found most of my current favorite programs and utilities. Check out their reviews."

Stephen Pelc sent these comments: "I have to second the comments about UltraEdit and BeyondCompare. They do what they say they do. But the kicker is in the technical support - both companies provide good and fast support. Now that BC3 is available on Linux and UEX is in beta for Linux, I can actually use Linux for development these days. The comments about the Foxit PDF reader are also right. What they should also say is that it's really small and really fast - so much so that we use it for on-line help systems using our DocGen literate programming tool and LaTeX to generate the PDF and an easily parsed index file.

"However, the tool that saves me most time is paper and pencil. The good old double-ended word processor is still king! It's truly multi-user."

Then there's this from Christopher Svec: "For your list of tools, I suggest checking out cscope: it's like ctags on steroids. You can find where any C symbol is defined, called, used, called-by, etc. It works *pretty well* with C++, and passingly well with other C-like languages. It has its own curses based interface, plus it interfaces very well with vim and emacs. It's open source, and can be found here: http://cscope.sourceforge.net ."

Bryan Murdock wrote: "I am shocked and dismayed that your tools page doesn't mention Emacs. Normally I'd think that it goes without saying, but since vim is called out on there (one of the Two Great Editors, but don't tell either side that I have praise for them both), maybe Emacs should be mentioned too. It's trivial to get on a Linux workstation. For windows I suggest this distribution:
http://ourcomments.org/Emacs/EmacsW32.html .

"Like is pointed out on your tools page about Vim, Emacs has color coding and indentation support for a multitude of languages, it can do code browsing with ctags and cscope, it has a nice side-by-side diff/merge tool called ediff, powerful search and replace, macros, and it doesn't beep at you when you open a new text document and simply start typing (OK, I get some vim digs in occasionally)."

Naming Conventions

Vlad Zeylikman had an interesting thought about naming conventions: "On the subject of understanding naming here is a snippet from a brain perception study. Apparently, the significance of consonants was understood long ago as abjad alphabet was quite functional as early as some 3000 years ago. so, we might consider abbreviations of a third type as acceptable: drop the vowels, not that everybody would be happy about it but it's a choice."

Jack notes: I didn't know what abjad meant so looked it up. Here's Wikipedia's take: "An abjad is a type of writing system in which each symbol always or usually stands for a consonant; the reader must supply the appropriate vowel. In popular usage, abjads often contain the word "alphabet" in their names, such as "Phoenician alphabet" and "Arabic alphabet"." Later, the article mentions that some languages, like English, are still pretty readable if the vowels are removed, and the process of removing them is delightfully called "disemvoweling!"

Vlad continues: "Brain Stuff from Cambridge University:

"Olny srmat poelpe can raed tihs.
"I cdnuolt blveiee taht I cluod aulaclty uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuan mnid, aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Amzanig huh? yaeh and I awlyas tghuhot slpeling was ipmorantt! if you can raed tihs psas it on!!"

Joke for the Week

Unix Man (set to "Nowhere Man")

He's a real UNIX Man
Sitting in his UNIX LAN
Making all his UNIX plans
For nobody.

Knows the blocksize from du(1)
Cares not where /dev/null goes to
Isn't he a bit like you
And me?

UNIX Man, please listen(2)
My lpd(8) is missin'
UNIX Man
The wo-o-o-orld is at(1) your command.

He's as wise as he can be
Uses lex and yacc and C
UNIX Man, can you help me at all?

UNIX Man, don't worry
Test with time(1), don't hurry

UNIX Man
The new kernel boots, just like you had planned.

He's a real UNIX Man
Sitting in his UNIX LAN
Making all his UNIX plans For nobody ...
Making all his UNIX plans For nobody.

About The Embedded Muse

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

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.