The Embedded Muse 159

Editor: Jack Ganssle (jack@ganssle.com) April 7, 2008

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? Learn this and far more at my Better Firmware Faster class, presented at YOUR facility. See http://www.ganssle.com/classes.htm .

Are you in the Chicago or Denver area? I'll present a public version of the Better Firmware Faster class in Chicago and Denver, on April 23rd and 25th. Registration and other info here: http://www.ganssle.com/classes.htm . You'll earn 0.7 Continuing Education Units, learn a lot, and have more than a little fun.

I'll be teaching a public version of this class in London, UK, May 19. See http://www.ganssle.com/classes.htm .

Robert Michael won the Muse 158 contest by naming the first embedded C compiler (at least, this matches my memory). A recent taped interview with P.J. Plauger, the founder of Whitesmiths (taped for the upcoming Embedded Systems Conference) confirms the date. Robert wrote: "I have no way to substantiate the date, but at TRW/Vidar I used the Whitesmith C compiler to generate 8080 code to control the embedded processors on the Vidar ITS class 4/5 digital telephone switching system, in 1978. We used Unix version 6

on a PDP-11/60 (with the Rand "50 changes" enhancements) in cross compiler mode."

Shameless Promotion:

In 1971 Intel invented the microprocessor… but few knew what to use it for. Even Intel predicted a tiny market for the device. But they persisted and the following year came out with the 8008, the first 8 bitter. Companies quickly realized how adding a bit of intelligence to their products reduced costs and increased functionality. The embedded systems industry was born.

I was in college, working as an electronics tech at a small outfit named Neotec, but was the only employee at the company who knew anything about programming. (We had been using a consultant, who remains in occasional contact.) They tapped me to be their first in-house firmware developer. Though it was easy to master 8008 assembly language, my software engineering skills were null. This was typical of the industry. All early embedded people had been trained as EEs, not computer scientists. 30 years of lessons learned in the software business were ignored by this fledgling industry.

Programs were small and management clueless about software projects. But the 8008 had a max address space of 16k so hacking was reasonably effective.

Projects grew in pace with expanding address spaces offered by the 8080, 8088, 68k and more. Feature sets exploded. Complexity skyrocketed. Glass claims that for every 25% increase in features complexity grows by 100%, a number that's hard to dispute.

At Neotec we reacted in the usual manner. Hired more people. Worked longer hours. Made promises we never fulfilled. And, of course, shipped plenty of bug fixes to irate customers.

I eventually left and started a consultancy with a partner. What a ride we had! What fun it was to put our gear in, say, a steel mill with a hundred foot long piece of 2000 degree steel four inches thick moving at a breakneck pace. (That used a PDP-11 and a number of Z80s). Or systems that lived 10,000 feet deep in the ocean, running on a few AAs for a year (RCA 1802). Then there was the White House security system (over 100 8085s), which we designed and installed during the Reagan years.

After too many late nights we started to sense that there were better approaches than our usual heroics. But what was wrong with us? Why were we working so hard?

Then I started an in-circuit emulator company. Now providing tech support to embedded developers all over the world I discovered something interesting. ALL of us had the same problems. Little groups of engineers all used heroics, and nearly all delivered late. Ironically, we who invented the communications age didn't talk to each other. So we all

faced the same sorts of challenges, thinking that we were doing something wrong and that other teams had the secrets to success.

I studied software engineering. The field, all but ignored by firmware folks, had a great body of accumulated wisdom. Frustrated with the state of our practices, in 1988 I started writing for various electronics/embedded publications about technical issues and better ways to develop code.

Over the years I've had the honor of serving on a number of boards, including one technical college, and was even tapped to teach engineering. In those cases it was shocking to see that too many schools push the same old dysfunctional software methods: here's how a "for" loop works. Now write some code.

A dozen years ago I sold the emulator business and tried retirement. After one day it seemed a rather bleak and meaningless state, so looked for something interesting to do, something which could be of benefit to others. Turns out I'm clueless about most traditional philanthropic activities, so decided to try and help out in the one field I know and love. Today I continue to write, and continue to try to educate, berate, and encourage embedded developers and their managers to improve their techniques.

A couple of companies asked me to come in and provide training about firmware development. Well, high school speech class was a trauma. I was terrified of talking in front of anyone, a fear that persisted into my 30s. But as a personal challenge I had decided to get over this irrational dread and took some speaking classes, and then practiced. When the Embedded Systems Conference was born they asked me to give a talk… and I found it an exhilarating experience. What fun to interact with all of these very smart people! It seemed natural to develop a seminar that would help folks get better code faster.

Over the last decade I've given the seminar to hundreds of companies and thousands of developers.

No one is naïve enough to think there's a silver bullet that will solve all our problems. This field is intrinsically tough. Even if we knew how to generate 10 million lines of perfect code in a day, customers will scream to shorten the schedule to an hour.

But… it *is* possible to create better code faster. It *is* possible to drastically reduce bug rates and shorten schedules. Those two do go hand in hand – Capers Jones showed that bugs are number one cause of missed schedules.

For example, there's an approach that drives the usual 5-10% pre-test bug rate down by three orders of magnitude. Though I mention it in the seminars I don't teach the technique as few of us, for a variety of reasons, will ever embrace it. That seems odd,

perhaps, but my interest is showing methods that most of us can start using *immediately* to get dramatic improvements.

And that's the entire thrust of the seminar. What can we do *today*, in a stealth fashion, under the scope of management's radar, to get better code faster? What can we do without embracing some monster methodology that few will really use, that requires a revolution in the engineering department that probably won't happen? What can we individuals do now to improve our performance?

That's my passion. I'm frustrated with consumer products that don't work well, with safety-critical code that isn't, with a situation where every three-year-old knows if something electronical acts odd, cycle power. Can't we find ways to meet the schedule?

We can. I'll show you how.

If you want more details and the usual marketing hype, see http://www.ganssle.com/classes.htm . Most of the time I present the seminar on-site, when a company asks me to come in for a day. Occasionally I do a public seminar at a hotel. We have decided to do public seminars in Chicago and Denver this month, and London in May. About half the participants fly in; the others tend to be local. Please join us; it's a jam-packed but always fun day. I guarantee you'll learn a lot, all of which is backed up with hard numbers you can take back to the boss and your colleagues. But seats fill quickly.

If you're looking for more hands-on, in depth training, especially targeted at getting less experienced engineers up-to-speed in developing embedded apps, consider Netrino's five-day Embedded Software Boot Camp, which consists of a concentrated series of lectures and hands-on exercises. There's more info here: http://www.netrino.com/Embedded-Systems/Boot-Camp .

Tools and Tips

Dan Swiger wrote: In your (next to) latest Muse, a reader asked about forums. You provide a remarkable list in EM158, but I had one that I didn't see in there. If you are a user of the SlickEdit editor (AWE-SOME!), then the Slick Edit community forum is great! http://community.slickedit.com/

The developers are very active in the forums and you very often find ways to enhance your use of the product. There are so many features in the editor, it is simply unbelievable, and reading through the forums is a great way to learn new techniques.

Great Engineers Response

John Taylor had some good thoughts in response to this from Steve Litt in the last Muse:
I contend that the guru is using a troubleshooting process. Sure, it looks to *us* like he sniffs, licks his finger and touches a node, and immediately discovers the problem, but what he really did was use a process to troubleshoot, including sophisticated use of divide and conquer. Either that or he's seen the symptom before and remembers the corresponding fix.

He wrote: I've had people put off asking me for help until the panic sets in just to avoid me making them feel stupid when "I touch the node..." It really is a process, to be sure, and one that actually begins long before I get to the bench. When someone comes to my office with a problem I always start with two simple questions: What is it doing and what is it supposed to be doing. I usually have to squeeze the answers to those questions out of them but I'm not getting out of my chair until they do! On the walk to the bench or the drive to the office I'm already working out in my head what I plan to measure or which component I plan to smell and then what to do if my first guess comes up empty.

These are the two rules by which I trouble shoot and design:

1. There is no magic: Every problem has a cause and that cause can be discovered through analysis of the system, operating conditions and symptoms. The word "Mysterious" must never be used in the final analysis, this word describes what should be only an interim lack of understanding.

2. You can't solve the problem if you don't understand the problem: If you throw fixes at a problem until the problem goes away (aka "The Shotgun Method"), you haven't really solved or learned anything. Once you understand the cause, the solution should be immediately evident even if that means starting from scratch. If what you try does not succeed, rework your understanding until it is in sync with reality.

What makes the guru-process seem mysterious is that the "What to do?" dialog is running internally and silently; what's the point of getting advice from someone that's just spent a week debugging a circuit with the power supply unplugged?

I have also found that I get a perverse pleasure from estimating the number of seconds I'll spend figuring out their week-long puzzle, but that's just me!

Another reader, who wanted to keep his name private, wrote: While working on a rack mounted telephone system our company was building the technician on the production floor called me out because, in one of the system checks, the noise meter was reading was too high. At first we could not determine the reason for the noise reading until I asked what kind of noise was present. The technician was puzzled at the question. I plugged a test telephone into a jack on the noise meter designed for that purpose. Listening to the telephone I heard the distinctive sound of 60 Hz hum and found that the field of the power transformer was coupling into the system's wiring. Old guys in that business had previously advised me to listen to the line. And on that day, I became an old guy.

Standards

My comments on standards provoked a lot of replies. Christoph Schmidt wrote: If you want to enforce common indentation, spacing, ... in C code, try GNU indent (http://www.gnu.org/software/indent/).

To comply with coding standards like MISRA C, I would recommend PCLint / FlexeLint (http://www.gimpel.com/), which can check for a wide range of rules as well as common pitfalls. Unfortunately, I haven't found an open source alternative yet. As it is a commercial program, it comes with some license cost. splint (http://www.splint.org/) is an open source LINT, but I does not like it that much. Do you know any other open source LINT?

For commenting, I use doxygen (http://www.stack.nl/~dimitri/doxygen/).Not just for creating some nice formatted documentation of my code, but also to ensure that everything is commented and the identifiers in the code match the ones in the comments.

Using these three tools, I can focus on writing correct code and meaningful comments, without need to study the coding guidelines every few minutes (except for naming conventions). Using these tools in groups produces a quite similar coding style, especially with new programmers, which are not used to the coding guidelines yet.

Pete Klammer said: "[...] only a few percent of companies enforce any such standards. Though many do have a standard, few conform to it. Why such resistance?"

This reminds me a little of my impression of ISO 9000: it doesn't require any particular method or process, but rather requires that your particular method or process be well-specified and documented, and adhered to with consistency.

Since there is no widely-shared consensus on any single coding standard, there will

inevitably be a variety of favorites amongst the staff as employees come and go. Since there is usually no definitive rationale for one style preference over another, much time, energy, and ego can be expended trying to resolve disputed preferences. While this examination might indeed be edifying, and the unexamined life is indeed not much worth living, the company is not paying for philosophy, and this project has to ship by Friday or else. Meanwhile, the losers of such contests are more likely to feel abused than converted, and less likely to feel comfortable sticking around.

It's easier to contemplate a coding standard per product, or per project, and indeed if your company is essentially one flagship program (an OS, say), then you might more easily have a "company-wide" coding standard. But for shops that ship some number of unrelated independent firmware projects, whose clients or even whose programmers may never see each other's work, arguments about degrees of refinement in coding style can seem like costly indulgence.

Of course, I do have a highly evolved coding style, still evolving in fact after 30+ years with many opportunities to revisit my earlier selves, and I believe it contributes to my productivity by framing my intentions -- higher-level constructs than the native programming language contains – in ways that are captured and manifest in the source text. I am just now trying to foist these ideas upon my son now taking his first programming class in engineering school, and his final grade -- as well as future use of the skills -- will tell me how right I am.

From Down Under, Geoff Patch wrote: Firstly, I agree about Mr Labrosse and his code. The source code to uC/OS is magnificent, and has been an inspiration to me for many years.

When I joined CEA, every engineer wrote code using his (we were all guys then!) own style. I could open up a file in Emacs, and immediately know who had written the code, and I always thought that this was A Very Bad Thing. Since taking over as the head of the Software Engineering group I've implemented and vigorously enforced a standard coding style, and my philosophy is understood and accepted by all the software engineers in the company. The younger people actually expect this, based on what they're being taught in their university courses. They would be astonished if we didn't have such a standard.

To ensure compliance with the standard, our code inspections have a defect category for "non-conformance to coding standards". A missing white space character will be tagged as a code defect in this category, and this defect will have to be corrected before the code may be released.

Engineers who believe that coding standards crush creativity are completely wrong. I

want my staff to be creative in ways that matter, such as their choice of algorithm. I don't want them to waste time being uselessly creative in trivial matters, such as where to place a curly bracket or how many characters to indent a line of code.

> One wag turned the tables on me. He made an analogy between
> coding and writing, and asked if authors (meaning me) should
> write equally "sterile" prose.

I've heard this before, and it's bogus. Much of "style" is about consistency of layout. English language style guides specify that we should all start sentences with a capital letter, use paragraphs, use common punctuation etc. Within those style guidelines you can write everything from a user manual to a poem without having your creativity hindered. Using standard style enhances both ease of expression and ease of comprehension of source code. If every engineer used a different style, it would simply hinder comprehension by other engineers.

James Thayer also had something to say about styles in writing prose: As it happens, many employers of authors and writers do impose a style guide for work that they publish. The reasons stated for doing so sound very similar to the reasons that it makes sense to have style guidelines for software. One resource for style guidelines for authors is: http://www.library.fordham.edu/res_guid/StyleGuide/StyleGuide.html

In short, perhaps coding and writing are not so different after all. I would also argue that in neither case does a style guide make the work "sterile".

Seriously, if anyone thinks that their creativity is stifled by how they place brackets or how they name their variables, they are probably in the wrong line of work. I'd much rather have someone working for me whose creativity is expressed in the way that they solve problems than in the way that they type their code.

A reader who wished to remain anonymous sent this: You said "Later I realized one difference between English prose and firmware is the longevity of the code." I agree and would add that while notable to many academics I would say that trying to read Chaucer's olde english is just that: trying.
For example:
For oute of olde feldys, as men sey,
Comyth al this newe corn from yere to yere;
And out of old bokis, in good fey,
Comyth al this newe science that men lere.

On coding standards and code reviews: You omitted to mention the MISRA software

guidelines which I think are an excellent example of guidelines developed specifically for embedded systems. As you know, MISRA - The Motor Industry Software Reliability Association published the "Development Guidelines for Vehicle Based Software", a remarkably compact collection of software development rules hat aid in the development of reliable, and safe, software for vehicle based software.

I tried establishing coding standards in a former workplace and was met with resistance from my coworkers and friends when I was their supervisor. The parties involved could not agree on conventions, rules and standards, even when some good third party standards and guidelines such as the MISRA guidelines mentioned above were available. I did not want to be authoritative so I backed off for the "greater good" as seen by the company, the output of code, rather embedded code which in turn went into a product that was itself shipped. The company's focus on code today and legacy support later led to archives of undocumented code, in varying styles, that had to supported, possibly for decades, by people other that the original designer. Your example standard outlines this last concern is where it says that software should: "clearly communicate how it works to a future programmer or the future version of yourself".

In the same working environment, I also tried establishing code reviews. These were also met with resistance from my coworkers for the reason that when one critiques another's work it is seen as criticism and is not received well. I tried to remove this interpersonal dynamic through the use of third party code tools, similar to pc-lint. But again, there were objections to the rules used by the code checkers and some of the rules were disabled making the tools ineffective. And again these efforts were set aside as they were seen as interfering with the output of code.

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.

GoAhead Software, located in a Seattle suburb, is changing the face of high availability and system management for the telecom, military and aerospace industries. Currently, GoAhead is seeking a Software Developer to join its technical team. This position designs, develops, documents, and tests software for a variety of operating systems. Qualifications include a BS degree and 4 or more years experience designing and implementing sophisticated software systems; excellent proficiency in C language with exposure to C++ and/or Java language; a thorough working knowledge of Linux operating systems; and experience developing and debugging with the GNU tool chain.

If you are a bright, career-minded individual with a passion for technology, we want to hear from you! Please email your resume and cover letter to careers@goahead.com. All qualified applicants are encouraged to apply.

Joke for the Week

.
No joke this week. Instead here is a compendium of funny instructions and other tidbits folks sent in response to last issue's link to amusing mnemonics.

From Tom Paden: But your humor spot has a bit of truth in it. The early 6800 had an undocumented test op-code called HCF; Halt and Catch Fire. It would fetch from consecutive memory locations to test out the address bus. I forget how I found out about it, but it's documented on Wikipedia: http://en.wikipedia.org/wiki/Halt_and_Catch_Fire

Chris Nelson wrote: An early mainframe OS at RPI had a joke language called Screwbol (if I recall correctly) which actually implemented HEO (Halt and Eat Operator). When executed, this instruction caused the big line printer to open the access door then slam the write head back and forth to cause the printer to walk across the floor toward the operator console.

Then there are these from Stephen Irons: The ones I always tried to avoid were from Motorola's 6803 microcontrollers:

BHT: Branch if High Tide
BLT: Branch if Low Tide

These were real instructions for at least the 6803, and I think for the whole 6800-series. The real meaning if I remember correctly were Branch if Higher Than and Branch if Lower Than, using an unsigned comparison, as opposed to BGT and BLT (greater than and less than) which did a signed comparison.

From Gary Gaebel: Your Joke brought to mind some similar hardware devices:

The permistable multivibrator -- great when you need a constant output signal. Saves on pin count too - no input pins needed.

The 9046 x N, random access, write-only memory chip. I actually have a spec sheet for this one.

And lastly, the Quad Two Input Maybe Gate. Apply any combination of 0's and 1's to the two input pins, and maybe you'll get a 0 on the output, or maybe you'll get a 1. Think

random number generator...

From Peter Klammer: One of my favorite opcodes was always:

LDCR -- line of dashes, cut ribbon

(For the younger generation: computers (and "tabulating" machines before them), especially the big one-per-campus installations in restricted-access raised-floor air-conditioned rooms, used to feature "line printers" with 132 hammers impacting upon fan-folded paper tractor-fed past a foot-wide ribbon and a spinning drum of type. The whole thing was bigger than a jukebox. You know what a jukebox is, right? It played 45's; you know, vinyl records, with grooves? Never mind.)

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.