The Embedded Muse 165

Editor: Jack Ganssle (jack@ganssle.com) September 15, 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 San Jose, Phoenix or Austin areas? I'll present a public version of the Better Firmware Faster class in San Jose on December 8, Phoenix on December 10 and Austin on December 12. 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. Sign up before November 8 and receive a $50.00 discount.

The Renesas Developer Conference will be in San Diego October 13-15. It's a heavily technical, hands-on, conference with sessions on CAN/LIN, Zigbee, RTOS, Ethernet, USB, Flash Technology, etc. They're also going to have the Mythbusters there for a special session. See http://devconrenesas.com .

Venture Development Corporation (VDC) has completed its 2008 embedded systems engineering survey. To download a copy of the highlights, see: http://www.vdc-corp.com/misc/08_esdt_eu_survey_highlights.pdf .

Inheritance

What started out as a joke a couple of Muses ago about how hardware people can talk to software engineers has turned into an interesting discussion. James Thayer had this to say: Inheritance vs. containment is one those things that people seem to screw up all too easily. I've always found that the phrases "is a" (inheritance) and "has a" (containment) to be useful ways to distinguish the two...

Bird "is a" Animal -- Inheritance
Bird "has a" Rock -- Containment
Bird "has a" Beak -- Containment

Circuit Board "is a" Insulating Material
Circuit Board "has a" Resistor
Circuit Board "has a" Battery"

But there are other things that people tend to screw up beyond containment/inheritance. It's possible to get that part right and still have bad decomposition of objects. The second set of examples is deliberately constructed to have some additional potential problems

For one thing, what is a Circuit Board? Is it the blank PWB or is it the stuffed board? If it's poorly named, we might make bad assumptions (and worse we might mix our

assumptions without realizing it). If I assume that Circuit Board is a blank PWB then inheriting from Insulating Material might make sense but containment of the resistor doesn't make sense. But if you assume that it's the stuffed board, then inheriting from Insulating Material wouldn't be right but the fact that it contains a resistor would make sense. (It might be better to say that Stuffed Board "has a" Blank PWB which "is a" Insulating Material.)

Another pitfall is trying to inherit from things that have attributes you don't care about in your operating context. Blank PWB might, in fact, be a good insulating material but if you don't really care about the insulating characteristics of the PWB then it really doesn't make sense for me to inherit these attributes. This example may seem obvious but there seem to be plenty of examples where someone decides that it made sense to inherit because class A really "is a" kind of class B, but the attributes of B are never referenced. There is no value in inheriting just because you can.

James Grenning wrote: The notion that OO's goal is to model the real world is not really up to date with current OO thinking. Some do use that way, and it does not usually lead to better designed software. The main goal of OO is dependency management, information hiding through encapsulation and the ability to define and program to interfaces. The typical problem of long dependency chains can be very effectively broken by inserting an interface. In C++ an interface is a base class with all pure virtual implementation. Essentially you get to define what an object is supposed to do, without revealing how it will do it. When a compile time dependency is broken by inserting an interface, essentially the top down dependency from client to server is inverted, making both the client and the server depend on the interface, without knowledge of the other.

Being able to model things form the problem domain is a bonus on top of dependency management.

Funny Datasheets

Stefan Wimmer likes the first sentence on page 7 of this datasheet:
http://focus.ti.com/lit/ds/symlink/ucc37321.pdf

Code Inspections

The recent discussions here about code inspections brought in an interesting email from Asbjorn Sabo: Nordic Semiconductor specializes in solutions for wireless communication. As a number of our chip designs contain microprocessors, embedded work and firmware development is a large part of the activity at our software group. For

internal purposes we compiled an overview of the code reviews done in a part of one of our development projects. It then occurred to us that this might be of interest for you, maybe for presentation in your newsletter, The Embedded Muse. (While this is a small and not very rigorous collection of data, it is real life data none the less.)

The code reviews cover a period of one and a half year, ending mid august 2008. The code in question is C code for an embedded 8051, controlling the radio part of the chip and implementing the lower layers of the protocol stack, developed by a team of four to five persons.

(Jack notes: It's hard to put the data into this text-formatted newsletter, so it's on-line at http://ganssle.com/misc/nordic_reviews.xls . There were 23 reviews of a total of 66 files which uncovered 399 errors, or about 6 per file on average.)

The rule is that before a review, the code shall compile cleanly and give no Lint warnings. The issues and errors found during the review are classified as major, minor, code that is correct but still should be improved, specification violations and code standard violations. In addition, some issues have not been classified. Both the number of issues and their classifications should be taken with a grain or two of salt, though. (Reviews have been done at different times, with different reviewers, some issues are difficult to classify, ...) Some issues are given multiple classifications, but they are only counted once in the "Total" column.

A few more things:
The code reviews are most often done by four persons (moderator, recorder, reader, author) who meet after preparations and read through the code, paraphrasing it. Another variant used include two reviewers reading the code thoroughly on their own, and then a short meeting with the author where the reviewers' give their comments to the code.

On average, most of three to four man days are spent on a review. For the reviews in this collection, a grand total of around eighty man days is a quite good estimate of the time used.

In an email exchange Asbjorn answered some questions for me:

> - About how many lines of code were reviewed?

That we have not tracked, actually. (But I can see that it might be a good idea to do so.) A rough estimate would be some hundred lines of code (not including comments and white space), maybe a bit more, per review on average.

> - Do you know how many bugs remained and were fixed after the reviews?

Not per reviewed module. But we have found around 45 bugs in total on this part of the project, that is on all the reviewed code together. These 45 bugs are typically functional errors (system does not work, does not work as intended, does not follow specification), often related to the interworkings of various software and hardware modules.

> - Is the team satisfied with the results?

Opinions are a bit divided, I think. Some feel that reviews are taking much time, others seem quite satisfied. This overview of the reviews was intended as input into this discussion, to see how many bugs we actually did catch and how useful the reviews were.

We have been trying various approaches to the reviews, from the full reviews (four persons in meeting, paraphrase all of code) via somewhat lighter reviews (two other persons read the code on their own, making comments on errors and things they do not understand) to a few very informal ones on smaller pieces ("Hey, could you have a look at this?"). I think the discussion on, and search for, the "best" way of doing reviews for us will continue for some time yet.

Comm Monitors

Yet more suggestions for tools to monitor communications protocols have come in. Ajay Wazir suggested YAT (Yet Another Terminal) from http://www.lvr.com/serport.htm .

Tjark van Dijk wrote: I followed the subject about protocol analyzing tools and maybe this product (indeed, one of my own) may be of interest to your audience. http://www.tildesign.nl/rs232_ttl_converter-20107101P2N .It's an RS232 to TTL converter, just Rx and Tx. The reason to develop a product like this was that I became soo tired making again an RS232-TTL converter on a breadboard because the controller board I had to work with just output TTL. And off the shelve solutions cost around $ 50 or more and were large.

If a Muse reader wants to order and tells me MUSEDISCOUNT in the order they can buy the unit for 11 Euros plus shipping in any quantity.

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.

Netrino has multiple open positions for embedded software developers and FPGA and board-level electronics designers, in Maryland and California. Join the Embedded Systems Experts(tm) in a fun work environment that offers challenge and variety. Learn more at http://www.netrino.com/Embedded-Systems/Embedded-Software-Jobs

Ready to take your career to the next level with an innovative, growing company in West Michigan? Discover what's waiting for you as a Customer Software Development Engineer at Gentex Corporation. We're looking for people with a four year degree in Electrical Engineering that have a high proficiency in C and C++ languages, assembly languages, and real time operating systems. In this role, you will interface with customer software requirements engineering, software test, electrical development, and electrical systems in the creating and debug of embedded software. Learn more and apply online today at www.gentex.com.

Embedded Software Developer required to join an established Irish company working on the next generation in automotive electronics. Requirements:
- Expert C/C++ coding and debugging in an agile environment.
- Minimum 5 years experience with low-power, real-time systems.
- Track record of designing high-availability products using 16/32-bit microcontrollers with virtual memory.
- Experience with CAN, embedded Linux, and GUI toolkits would be an advantage.

We are located in Irelands fastest-growing city, and one of the sexiest cities in the world: http://en.wikipedia.org/wiki/Galway . Contact: careers@bluetree.ie .

Joke for the Week

We in the US are preparing to switch to digital TV in 2009. Many might not know of the looming switchover of the power grid from AC to DC. For more on this see http://www.csl.sri.com/users/neumann/insiderisks08.html#214 .

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.

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.