The Embedded Muse 96

Editor: Jack Ganssle (jack@ganssle.com) April 21, 2004

You may redistribute this newsletter for noncommercial purposes. For commercial use contact info@ganssle.com.

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

Editor's Notes

Want to learn to design better firmware faster? Join me for a one-day course in Chicago on May 17. This is the only non-vendor class that shows practical, hard-hitting ways to get your products out much faster with fewer bugs. See http://www.ganssle.com/classes.htm for more details. There's also cheap fly-in options listed on the web site for folks coming from out-of-town.

I often do this seminar on-site, for companies with a dozen or more embedded folks who'd like to learn more efficient ways to build firmware. See http://www.ganssle.com/onsite.htm.

I've created a video titled "Develop Firmware in Half the Time" that distills the basic ideas and processes needed to efficiently crank out great firmware. There's more information available at http://www.ganssle.com/video.htm.

Mike Whitcombe passed along the URL to a dirt cheap project management tool named ProjeX. It's an add-in for MS Excel. Find it at www.waa-inc.com.

Metrics tools can be very useful to access the quality of your code. I've used some freebie versions (see www.chris-lott.org/resources/cmetrics/) and Programming Research's nice but expensive QA-C (www.programmingresearch.com). John Johnson sent along a link to a $195 tool called RSM that provides all sorts of useful metrics: http://msquaredtechnologies.com/index.html.

Copyright 2003 by The Ganssle Group. All Rights Reserved. You may distribute this for non-commercial purposes. Contact us at info@ganssle.com for more information.

One of the interesting figures these tools spit out is a complexity metric, generally using the McCabe Cyclomatic algorithm. This is the number of independent paths, and therefore the minimum number of paths that should be tested. If your code exceeds some threshold it's time to refactor and simplify.

As Brian Kernighan aptly noted: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Keep the code simple. Use metrics tools to flag that which needs recoding.

Facts and Fallacies of Software Engineering

Robert Glass's new book, Facts and Fallacies of Software Engineering (2003, Addison-Wesley, ISBN 0-321-11742-5) discusses 55 "facts" and 10 fallacies that permeate this industry. It's a great title and an interesting concept, but I felt the book didn't live up to its promise.

Some of the facts are correct but so well-known as to be trite. Fact 6, for instance, states that learning a new tool or technique lowers productivity. Well, duh.

Others are interesting. Fact 5 tells us that most software tools and techniques will yield a 5 to 35% increase in productivity and quality... not the orders of magnitude so often promised. That jibes pretty well with my own experience. My rule of thumb is that tool vendors over-promise and under-deliver. A corollary is that consumers of these tools often expect miracles. "This time we won't run into trouble because we're buying (trumpets sound) A New Tool!" Glum faces weeks later testify to the collapse of their hopes.

Fact 7 is amusing, and much too true: developers evaluate lots of tools, buy many, and use practically none.

A number of facts address scheduling and estimation. The short version: estimates are done at the wrong time (at the outset of the project, before the problem is completely understood), by the wrong people (upper management and marketing), and are usually not corrected as better data becomes apparent. All true. He gives no solutions.

An fascinating study quoted in the book reveals that the projects with the highest productivities were those for which no estimates were done at all!

Copyright 2003 by The Ganssle Group. All Rights Reserved. You may distribute this for non-commercial purposes. Contact us at info@ganssle.com for more information.

Maybe the most interesting numbers are in Fact 21: for every 25% increase in the complexity of a problem the software complexity doubles. This one bit of wisdom explains many of the problems we run into when building large systems. Requirements explode, estimation is tough, optimization often doesn't – or can't happen.

Fact 32 isn't a surprise to those who study the literature of software engineering, but it often shocks developers. When a programmer says the project is "done", generally 40 to 45% of the logic paths have never been tested. No wonder firmware is so buggy! And Fact 33 shows that even 100% testing isn't enough. 35% of all bugs arise from missing logic paths; another 40% from the execution of a unique combination of logic paths. 100% testing just ain't enough.

The book is a very fast and entertaining read. It's humorous and some of the information will make you stop and think.

Perhaps my biggest objection to the work is the lack of solutions. Glass inundates us with problems, and then often says "that's just the way software is; you can't expect it to be any different." He's right, and he's wrong. Software IS hard. But we can do a better job of testing... and we must. We can get more out of tools... and we must. And it is possible to do better estimates, given support from management and courageous developers willing to convey what might be bad news.

Testing

A new magazine, Software Test and Performance, arrived in my snail-mailbox this week. One short opinion piece got my attention. Parasoft's CEO Adam Kowala wonders why we spend so much of a project on test and debug. He notes that every mature industry has learned to anticipate defects and prevent them from ever entering their products.

It's an interesting thought. I'm amazed at how few problems creep into automobiles. A modern car is an incredibly complex piece of equipment that probably has hundreds of thousands of parts. Yet autos generally meet a very high quality bar. Kowala uses the automotive industry as an example; however, the comparison really isn't fair. Cars are mass produced, identical copies stamped out in great volumes. Every software application is a hand-made, unique product. There's no mass production of programs that's analogous to manufacturing.

He complains that we developers rely on testing to insert quality into the product, while we know that testing, while critically important, isn't enough to prove correctness (see Fact 32 above). He says: "We don't think of the whole process of building and deploying

Copyright 2003 by The Ganssle Group. All Rights Reserved. You may distribute this for non-commercial purposes. Contact us at info@ganssle.com for more information.

software in a way that would *prevent* errors, because we don't believe it can actually be done." He goes on to say that not only is defect prevention possible, but every mature industry has stopped relying on testing as a way to make their products correct.

Kowala proposes that wee need a process for integrating error prevention in the software lifecycle. Kowala's Parasoft, it should be noted, sells products designed to reduce defects so he's hardly a disinterested third party. But that doesn't reduce the importance of his message. I'd argue that the adoption of other processes – standards, code inspections and the like – are a critical part of eliminating errors before they surface in the code. But he is absolutely right in complaining about our reliance on testing alone, when we know that it just does not work.

(Unfortunately the magazine isn't available on-line, but the web site, www.stpmag.com, gives subscription information).

Electronic Voting

With a presidential election in the offing the furor over electronic voting machines continues to escalate.

Votehere provides a technology that confirms the accuracy of these machines. They recently released the source code (www.votehere.com), not as open source, freeware or for distribution purposes, but to show correctness through transparency. The company knows that security through obscurity is no security at all. I commend them on taking this initiative.

This week's Risks Digest (http://catless.ncl.ac.uk/Risks/23.32.html) has a long (for Risks at least) and unfortunately rather partisan commentary about the state of the industry. But it is interesting reading.

A recent MIT study rates hand-counted paper ballots as the best for eliminating overvotes and undervotes. "The difference between the best performing and worst performing technologies is as much as 2 percent of ballots cast. Surprisingly, (hand-counted) paper ballots -- the oldest technology -- show the best performance."

I voted using Diebold electronic equipment in Maryland's primary in March. Despite my concerns for the integrity of the software, touch screen displays made the entire process very intuitive. What happened to my vote? I can only hope and presume it made its electronic way into a secure database and was counted accurately.

Jobs!

Copyright 2003 by The Ganssle Group. All Rights Reserved. You may distribute this for non-commercial purposes. Contact us at info@ganssle.com for more information.

Let me know if you're hiring firmware or embedded designers. I'll continue to run notices for embedded developers as long as the job situation stays in the dumper.

Z-World needs embedded people: http://www.zworld.com/company/careers/. You may know these folks from their very popular Rabbit microprocessor.

Torkin Lian wrote: Check out "Join us" at http://www.axxessit.no/ for embedded software development position. This vacancy is situated in Norway.

An Annapolis, Maryland outfit needs Systems Engineers (2) programming for multiprocessing in a UNIX environment, and Signal Processing Guru's (2) for sonar and radar applications. Hands-on experience in designing, programming and testing high performance DSP systems important. In addition, they will have experience in designing and developing software for multi-processor, multi-threaded architectures in a UNIX environment. Email dannice@shorehire.net.

Joke for the Week

Josef Roehrl wrote:

To the optimist, the glass is half full.
To the pessimist, the glass is half empty.
To the engineer, the glass is twice as big as it needs to be.

So, if you let the engineer at it (fix the glass), the optimist, while pretty happy, becomes happier and the pessimist becomes ecstatic.

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*".

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.

Copyright 2003 by The Ganssle Group. All Rights Reserved. You may distribute this for non-commercial purposes. Contact us at info@ganssle.com for more information.