More busting bugs from birth to death of an embedded system running an RTOS

Robert Berger
Embedded Software Specialist
Stratigou Rogakou 24, GR-15125 Polydrosso/Maroussi, Athens, Greece
Phone : (+ 30) 697 593 3428, Fax: (+ 30) 210 684 7881
E-mail: robert.berger@ReliableEmbeddedSystems.com

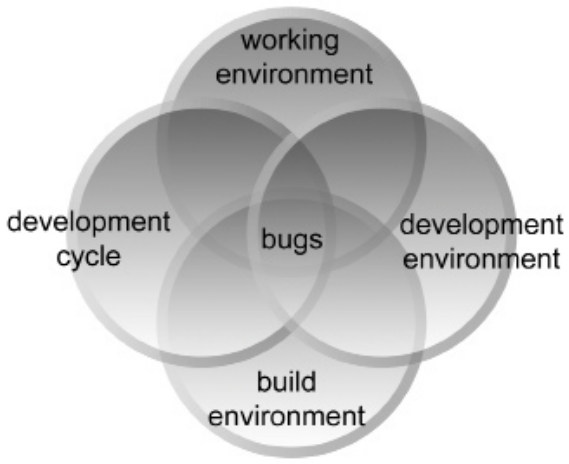
Abstract

After the positive feedback from last year's talk[1] I decided to extend my speech about saving time, money and nerves by using the right "down to the earth debugging philosophy". How do you avoid bugs? How do you find and fix those bugs you could not avoid before hand? What options do you have to tackle problems when running your application on top of an RTOS and even before that? In my mission to help people create better embedded software let me try to strengthen your fundamental understanding of how embedded software teams should work and how embedded software operates.

1 Introduction

A bug is a malfunction which needs to be characterized and reproduced in the absence of any debug tool in order to avoid a Heisenbug. If you don't clearly understand the problem you'll have small chances to properly fix it. Time to market pressure as well as a continuous increase in complexity make it pretty tough to reach zero defects so bugs will sneak into your system and the sooner you eliminate them the cheaper it gets.

Let's see where bugs are introduced before looking how to prevent or find them:



1.1 Working Environment

1.1.1 Office Space

The excuse that "great office space" is too expensive has been proven wrong by the work of DeMarco and Lister back in 1999[2].

1.1.2 Time Management

Interruptions require a period of immersion to reenter the mental state of "flow" which is in the order of 15 minutes[2]. Avoid overtime by setting your priorities right since working constantly under pressure and with unclear/unrealistic goals can easily lead to burnout and high bug rates.

1.2 Development Cycle

The linear development cycle as taught by the project management institute PMI[3] might look like this:

- Requirements
- Design/Planning/Architecture (FRAT[4], Wideband Delphi[5], Cone of Uncertainty[6],...)
- Implementation
- Integration (avoid this by continuous integration[7])
- Testing (unit testing[8])
- Release
- After Sales Support

1.3 Development Environment[9]

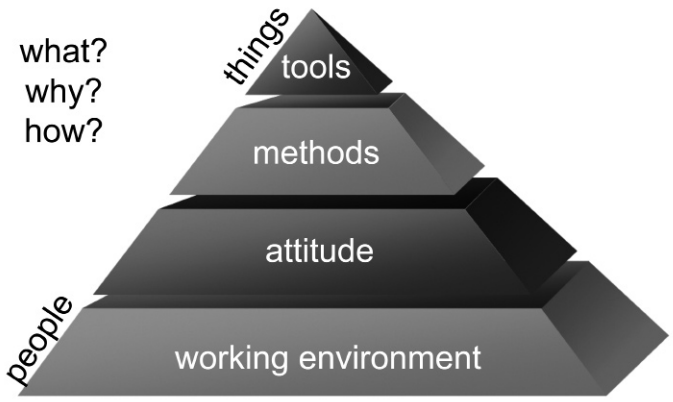
As a development environment I propose a combination of an issue tracking system [10] with a version control system (VCS)[11, 12] and a documentation system [13]. Like this you can link code from the VCS with requirements from the documentation system and issues from the issue tracking system. The VCS helps you to reproduce released code.

1.4 Build Environment

There shall be a distinction between developers and framework maintainers. The tool most used by a developer is his/her editor. Every developer shall be able to edit/compare/search in his/her preferred environment, but the compilation has to happen on some centralized server (cluster), where the framework maintainer makes sure that the build environment is identical for everyone and that it's not easy for a developer to change it. This setup allows you to make a complete snapshot of the build environment which in combination with the VCS allows you to fully reproduce the host and target side of the software image.

2 It's all up to the people

I would like to point out that it's all about people and their attitude towards methods and tools. If someone does not understand what/why/how methods/tools are used he/she will never use them properly. Life long learning and proactive behavior are obligatory for a team, which helps to produce high quality code.



3 Find and fix bugs

Although an RTOS usually helps partitioning your system it also potentially introduces shared data problems. The questions are wether you actually need an RTOS and what debugging capabilities it offers to you. Does your boot loader allow in the field software upgrade? How about remote debugging and post mortem analysis?
As mentioned above one of the phases of software development is the implementation phase with the typical edit-compile-debug cycle. You already know my opinion about editors that every developer should use his/her preferred one. The compiler can be your friend. By using the highest warning level, fixing the warnings and making your code compile with different compilers/compiler options and compiler versions you can eliminate “suspicious” and even buggy code. Make sure to compile the code you test and the code you ship with the same options. The third component is the debug phase. Here we need to distinguish between debugging by stopping the whole system/task (monitor program, SW debugger, ICE/JTAG/BDM,...) and debugging a running system (memory debugging, profiling, tracing, wiggle GPIO pins, watchdog, exceptions,...) as well as debugging in the lab and remote debugging in the field.

4 Prevent bugs

4.1 A better edit-compile-debug cycle

After a clean compile and a static code check[14,15] but before the above mentioned debug phase a code review[16] should happen (find functional bugs, code reading, coding style, metrics, documentation[17],...) based on a firmware standard[18].

4.2 Productivity collapse

The productivity collapses non linearly with the number of lines of code(LOC)[19].

Countermeasures would be

- Partitioning
- Code reuse (architecture needs to be compatible, write reusable code, use 3rd party code - code spelunking[20], support different operating systems[21],...)
- Rise level of abstraction
- Training[22]

5 Special considerations

It’s considered good practice to periodically read and verify mission critical registers (like mapping of peripherals to a specific address space) for the system to be able to recover. Other registers, which are semi-critical (UART, SPI,...) might be refreshed before every transaction and not only once during system startup.

6 Discussion

In times of worldwide crises those who can save money and create embedded software development teams with higher productivity are the winners. I hope you were able to pick up some new ideas and are eager to try them out on your road to be such a winner. Companies are under serious pressure and can not keep on doing things like they used to but need to create a people-centric culture with high quality standards. It's all about improving our habits and trying new things. If you stayed with me so far, you are on the right track, since life long learning has the potential to change your habits and thus to decrease your number of bugs.

7 References

[01] Robert Berger: Busting bugs from birth to death of an embedded system running an RTOS ISBN: 978-3-7723-3961-5, Proceedings & Conference materials embedded world 2008
[02] Tom DeMarco & Timothy Lister: Peopleware: Productive Projects and Teams 2nd ed., ISBN: 9 78-09 32633439 , Dorset House Publishing Company, Incorporated, 1999
[03] Project Management Institute, http://www.pmi.org/
[04] Brian W. Mar & Bernard G. Morais: FRAT – A basic framework for systems engineering, Proceedings of the 11th Annual Int. Symposium of the Int. Council on Systems Eng., 2002
[05] Andrew Stellman & Jennifer Greene: Applied Software Project Management, ISBN: 9 78-059 6009 489 , O'Re illy Me dia, Inc., 2005
[06] Steve McConnell: Software Estimation: Demystifying the Black Art, ISBN: 9 78-0735605350, Microsoft Press, 2006
[07] Niels Malotaux: Evolutionary PM methods , http://www.malotaux.nl/nrm/Evo/
[08] http://unittest-cpp.sourceforge.net/
[09] Bugzilla/CVS/Mediawiki integration, http://oss.segetech.com/integration.html
[10] Bugzilla, http://www.bugzilla.org/
[11] Concurrent Versions System, http://www.nongnu.org/cvs/
[12] Subversion, http://subversion.tigris.org/
[13] Mediawiki, http://www.mediawiki.org/
[14] Splint, http://www.splint.org/
[15] Coverity Prevent, http://www.coverity.com/
[16] Karl E. Wiegers: Improving Quality Through Software Inspections, http://www.processimpact.com/articles/inspects.html
[17] doxygen, http://www.stack.nl/~dimitri/doxygen/
[18] Jack G. Ganssle: Firmware Development Standard, http://www.ganssle.com/misc/fsm.doc
[19] Barry W. Boehm: Software Cost Estimation with Cocomo II, ISBN: 978-0130266927, Prentice Hall PTR, 2000
[20] Codespelunking, http://www.codespelunking.org/
[21] Mapusoft, http://mapusoft.com/
[22] shameless self-promotion, http://www.reliableembeddedsystems.com/training.html

8 Author

After working for multinational companies in Middle Europe, Robert Berger is currently an embedded systems software manager for one of the largest multinational groups in Southeast Europe. He is part of the team that received from TMC the 2006 product of the year award in the USA for their communications solution and deployed 2007 the first commercial HD (high definition) H.264 IPTV solution in the USA and 2008 the first commercial HD H.264 IPTV solution in Canada. He has always seen himself as self employed and consults, trains and engineers projects all over the globe. His true passion is to mentor and coach, so he works also as a teaching assistant for the embedded course of a private university in partnership with Carnegie Mellon. He enjoys helping companies improve their development process and writing embedded software. He is a project manager for the Linux Driver Project, IEEE lecturer for professional activities in the EMEA Region, speaker at the Embedded World Conference and moderates various Embedded Systems Professionals forums all over the world wide web.