Busting bugs from birth to death of an embedded system running an RTOS

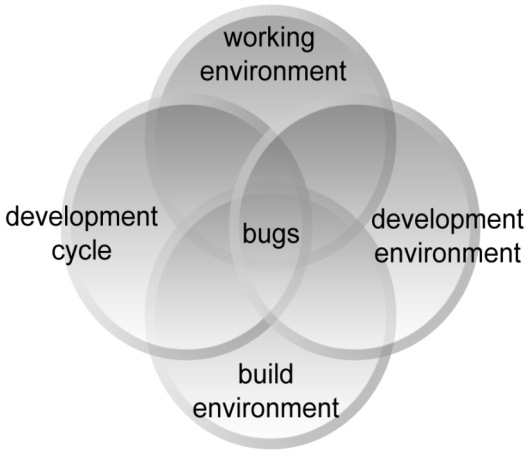
Robert Berger
Embedded Software Specialist
Stratigou Rogakou 24, GR-15125 Polydrosso/Maroussi, Athens, Greece
Phone: (+30) 692 593 3428, Fax: (+30) 210 684 7881
email: robert.berger@reliableembeddedsystems.com

Abstract

We are all trying to reduce the amount of bugs in our products, but as Frederik P. Brooks, Jr. wrote in 1975 [1] “There's no silver bullet”. ...and thirty three years later there's still no silver bullet. The Standish Group [2] analyzed U.S. companies developing application software and found, that only 16% of all projects were successful in 1994 while ten years later still only 34% were on time, on budget and not canceled. Taking into account that embedded software is by nature an order of magnitude more complex than application software, one can assume that between 60% and 80% of embedded projects will fail. What follows is a brief overview how to debug an embedded system. We'll begin before “classic troubleshooting” would typically start and finish after it would typically end. Keep in mind that the later in the development cycle an error is found, the more costly it gets [3]. Of course, we will also discuss what I consider “classic debugging”, which is utilized during the implementation, integration and testing phases.

1 Introduction

“We can't solve problems by using the same kind of thinking we used when we created them.”
- Albert Einstein



Bugs are introduced in the following areas:

1.1 The working environment

1.1.1 Immersion

The research of DeMarco and Lister [4] showed that all tasks require a period of immersion depending on their difficulty/complexity. This means, for example, that if you are interrupted writing some firmware, you will need at least 15 minutes to return to the point you left before being interrupted. A working environment, where you can work uninterrupted over a period of time will enable you to produce fewer bugs.

1.1.2 Overtime

If you are constantly under pressure and working overtime, you will be stressed and end up burning out with health problems. A working environment where you are encouraged to work smart instead of working hard will decrease your bug rate.

1.2 The development cycle

The classic linear software development cycle as taught by the PMI [5] might look like this:

- Requirements
- Design/Planning
- Implementation
- Integration
- Testing
- Release
- After sales support

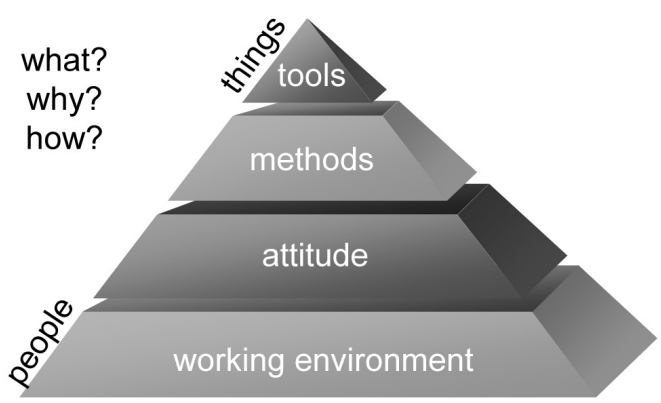
In the waterfall model it can take a couple of years from requirements to release. Developers are going to procrastinate until shortly before the deadline and then there will be long working hours, quick fixes and lots of bugs. It's certainly better to apply an iterative waterfall model [6], where the whole development cycle is shortened to a couple of weeks and repeated again and again. In this model, changing requirements can be dealt with, prototypes can be given to customers and developers will work more productively.

1.3 The development environment [7]

For the overall development framework I propose the integration of an issue tracking system [8] with a version control system [9,10] and a documentation system [11]. It is then possible to have links between issues/requirements and version control. The version control system enables you to retrieve the sources for each release.

1.4 The build environment

The most used tool of a software developer is his/her editor plus the operating system on which this software runs. Every developer shall be able to edit/compare/search in his/her preferred environment, but the compilation has to happen on some central server (cluster), where the build environment is identical for everyone and there is no way for the developer to change it. This setup enables you to make a complete snapshot of the build environment. After implementing a development and a build environment as described above you will be able to fully reproduce the host and the target side of the software image.



Before we discuss methods and tools, I would like to point out that it's all about people and their attitude towards those things. If someone does not understand why methods are used the way they are used, or has a better idea he/she should be encouraged to point this out and get a chance to discuss it. Life long learning and proactive behavior are obligatory for a team, which helps to produce high quality code.

2 Methods and Tools

2.1 Requirements/Architecture - Design/Planning

2.1.1 FRAT [12]

FRAT is based on the hypothesis that four views of a system are needed to completely define it. These views are: what it does (functions), how well the functions are performing (requirements), a physical or actual description of the system (answers), and verification and validation when completed (tests).

2.1.2 Risk Matrix [13]

		probability		
	risk	low	medium	high
consequences	high	medium	***high***	***high***
	medium	low	medium	***high***
	low	low	low	medium

2.1.3 Partitioning [14]

The number of lines of code produced per year decreases drastically with the amount of lines of the whole project. Partition your project into sub projects and automatically more lines of code will be generated per year (multiprocessor systems).

2.1.4 Prioritize (80/20)

From Boehm's Top Ten List On Software Improvement [15]:

- 80% of avoidable rework comes from 20% of the defects
- 80% of the defects come from 20% of the modules
- 90% of the downtime comes from 10% of the defects

2.1.5 Wideband Delphi [16]

- give requirements to the team
- the team decides on metrics (LOC, hours)
- assume all tasks will be done in sequence and everyone is 100% devoted to each task
- everyone proposes his/her estimates
- plot estimates
- re-iterate until estimates converge

To keep this paper resonably short we'll skip a detailed discussion on tools. Short lists are provided instead. References are given where appropriate. The symbols used are as follows: ▼ low cost, ▲ high cost, ▼▲ huge variation in price, ◄► pays off.

2.2 Implementation (host side)

	Cost	RTOS	comment
Firmware Standard	▼		See also [17]
Code review	◄►		Needs time to be done, but pays off quickly [18]
Coding style	▼		code beautifier [19] part of build environment
Static code check	▼▲		[20, 21, 22] part of build environment
Metrics	▼		code metrics [23] part of build environment
Documentation	◄►		doxygen [24] part of build environment
Reentrant Code	▼	✓	necessary for RTOS and interrupts [25, 26]
Programming Lang.	◄►		see also [27]
colorgcc	▼		colorgcc [28] part of build process
Reuse	◄►		initially bigger effort, pays off [29] with variations
Makefile debugger	▼		see also [30]
Editor	▼▲		the tool used most by developers [31, 32]

2.3 System/Task stopped (target side)

	Cost	RTOS	comment
simulator (host side)	▼▲		can be slow in execution [33, 34]
eval board	◄►		eval board+BSP+RTOS = quick sol. [35, 36, 37]
monitor program	◄►		bootloader, some debug, SW breakp. [38, 39]
remote debugger	▼▲		e.g. gdbserver [40]
RTOS aware debug.	▲	✓	knows about RTOS kernel objects [41, 42]
ROM emulator	▼▲		quick download of code - historical
ICE/JTAG/BDM	▼▲		breakpoints single-step view/modify values/registers memory dump see [43, 44, 45]

2.4 Running System (target side)

	Cost	RTOS	comment
stack check	▼	✓	usually part of RTOS [46]
heap check	▼		modified versions of malloc/free [47]
profiler	◄►		see [48, 49]
tracer	▼▲		[52] or extension to JTAG/BDM/ICE [43, 44, 45]
SW controlled LEDs	▼		display boolean values
Scope/LA	▲		toggle pins and measure - least intrusive
7 segment display	▼		extra code needed - in early stages for debug
char. based LCD	▼		extra code needed - in early stages for debug
MERT	▼		fancy printf
graphic display	▲		extra code needed - in early stages for debug
system probe	◄►	✓	little overhead - heavy work done on host [50]

2.5 Integration

As already mentioned above it's crucial to have both a build and a development framework to be able to reproduce the image, which goes first to the test team and later on to the customer. By embedding a time stamp and a version tag in the image, it's possible to uniquely identify the software at any time.

2.6 After sales support

In case you do have some connectivity to the devices in the field the following features can and should be provided:

2.6.1 Remote software upgrade

A bootloader is typically involved here with some application triggering the software upgrade. With sufficient support from the hardware, this can be made quite secure utilizing for example, a multi stage bootloader.

2.6.2 Remote debugging

All of the target side tools can be utilized here, but usually it's about identifying the cause of the problem using MERT or a system probe, and possibly providing a quick patch for it.

3 Discussion

Although there is no generic solution, it is possible to be on time and on budget [51]. Some methods/tools are great, but there seems to be some difficulty for us using them properly and in the right context. This is due to not having the "big picture" and our unwillingness to improve our habits. To solve the problem with our habits we would need to debug the human mind, which is out of the scope of this paper and conference. If you stayed with me until now, you are on the right track, since life long learning has the potential to change your habits. I hope you were able to pick up some new ideas and are willing to try them out. Ben Franklin says: "*The definition of insanity is doing the same thing over and over and expecting different results.*", so if you want to change your bug rate, you should try something different.

4 References

[01] Frederick P. Brooks: The Mythical Man-Month 2nd ed. (20 year anniversary edition), ISBN: 978-0201835953, Addison-Wesley Professional, 1995
[02] The Standish Group, http://www.standishgroup.com/
[03] Alan M. Davis: Software Requirements: Objects, Functions and States 2nd ed., ISBN:978-0138057633, Prentice Hall PTR, 1993
[04] Tom DeMarco & Timothy Lister: Peopleware: Productive Projects and Teams 2nd ed., ISBN: 978-0932633439, Dorset House Publishing Company, Incorporated, 1999
[05] Project Management Institute, http://www.pmi.org/
[06] Niels Malotaux: Evolutionary PM methods, http://www.malotaux.nl/nrm/Evo/
[07] Bugzilla/CVS/Mediawiki integration, http://oss.segetech.com/integration.html
[08] Bugzilla, http://www.bugzilla.org/
[09] Concurrent Versions System, http://www.nongnu.org/cvs/
[10] Subversion, http://subversion.tigris.org/
[11] Mediawiki, http://www.mediawiki.org/
[12] 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
[13] Risk Matrix, http://en.wikipedia.org/wiki/Risk_Matrix
[14] Steve McConnell: Software Estimation: Demystifying the Black Art, ISBN: 978-0735605350, Microsoft Press, 2006
[15] Barry Boehm & Victor R. Basili, SW Defect Reduction Top 10, IEEE Computer, 2001
[16] Andrew Stellman & Jennifer Greene: Applied Software Project Management, ISBN: 978-0596009489, O'Reilly Media, Inc., 2005
[17] Jack G. Ganssle: Firmware Development Standard, http://www.ganssle.com/misc/fsm.doc
[18] Karl E. Wiegers: Improving Quality Through Software Inspections, http://www.processimpact.com/articles/inspects.html
[19] GC GreatCode, http://sourceforge.net/projects/gcgreatcode/
[20] Splint, http://www.splint.org/
[21] Coverity Prevent, http://www.coverity.com/
[22] Klocwork, http://www.klocwork.com/
[23] Tim Littlefair: An investigation into the role of Software Metrics in software quality improvement, 1999, http://cccc.sourceforge.net/
[24] doxygen, http://www.stack.nl/~dimitri/doxygen/

[25] Jack G. Ganssle: The Firmware Handbook (Embedded Technology), ISBN: 978-0750676069, Newnes, 2004
[26] Phillip A. Laplante: Real-time systems design and analysis: an engineer's handbook 3rd ed., ISBN 0-471-22855-9, IEEE press, 2004
[27] Lutz Prechelt: An Empirical Comparison of Seven Programming Languages, IEEE Computer, 2000
[28] colorgcc, http://www.schlueters.de/colorgcc.html
[29] Victor R. Basili & Marvin V. Zelkowitz: Empirical Studies to build a science of computer science, Communications of the ACM, Nov. 2007
[30] GNU Make Debugger, http://gmd.sourceforge.net/
[31] Xemacs, http://www.xemacs.org/
[32] Eclipse, http://www.eclipse.org/
[33] Simics, http://www.virtutech.com/
[34] Rudolf Graf: Simula5x, ISBN: 3-928434-14-4, Feger&Co., 1992
[35] IAR Development kits, http://www.iar.com/
[36] Wind River BSP, http://windriver.com/products/bsp_web/
[37] Micrium ports, http://www.micrium.com/products/rtos/kernel/ports.html
[38] Ed Sutter: Embedded Systems Firmware Demystified, ISBN: 978-1578200993, CMP Books, 2002
[39] u-boot, http://www.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary
[40] gdb manal, http://sourceware.org/gdb/current/onlinedocs/gdb_18.html#SEC161
[41] Micrium/IAR C-Spy, http://www.micrium.com/products/rtos/kernel_awareness/kernel_aware_iar_c-spy.html
[42] Wind River Device Debuging, http://www.eclipse.org/dsdp/dd/
[43] ICE & BDM ,http://www.lauterbach.com/
[44] ICE & PROBE & TRACE, http://windriver.com/products/OCD/
[45] JTAG-Trace & JTAG-Link, http://www.iar.com
[46] Jean J. Labrosse, MicroC OS II: The Real Time Kernel 2nd ed., ISBN: 978-1578201037, CMP Books, 2002
[47] Dmalloc - Debug Malloc Library, http://dmalloc.com/
[48] gprof, http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html
[49] oprofile, http://oprofile.sourceforge.net
[50] uC/Probe, http://www.micrium.com/products/probe/probe.html
[51] T.Gilb: Principles of Software Engineering Management, ISBN: 0201192462, Addison-Wesley Pub Co, 1988
[52] Linux Trace Toolkit next generation: http://ltt.polymtl.ca/

5 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 U.S.A for their communications solution. 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 and IEEE lecturer for professional activities in the EMEA Region.