

Need help **debugging** the code in your ARM-based embedded system?



Break Points

Faster! Can we design embedded systems faster, cheaper, better?

Jack Ganssle

6/1/2008 11:30 AM EDT

Not long ago, my close friend Kirk, who spent his career managing real estate, read *The Soul of a New Machine*, Tracy Kidder's wonderful account of how engineers at Data General produced the *Eclipse minicomputer* in record time. Kirk found the book interesting and well written, but was dismayed at the high-pressure schedule and the people burnout. Then he made a comment that literally made me stop in my tracks: "I just couldn't believe that the picture Kidder paints of the high-pressure schedule is real, though; no one can work like that for long."

How could I explain to someone with no connection to the high-tech world how schedules are always the bane of our existence? That in my career, and in those of almost every engineer I know, every project we do is made to capricious and impossible deadlines? That in recent years timelines have shrunk even more, so Kidder's depiction seems almost benign by today's standards?

So I'm left wondering if perhaps anyone not involved in the technology business has a clue about how we're driven to madness by impossible schedules. Is our business unique? How many other businesses have such long-term and relentless pressure to get things done faster? Is constant, unpaid overtime a theme of any other segment of the economy?

Decent project management software appeared in the 1980s. Anyone can enter complex PERT and Gantt charts outlining every nuance of piecing together big endeavors. Who uses this stuff successfully? I've watched uncounted developers attempt to build a schedule around an arbitrary deadline set by marketing: they move triangles around like crazy, all except the final one, the one that has meaning, in an attempt to create a schedule that sounds believable, all the while knowing it's utter nonsense. When I was in high school, the Jesuits mailed us our report cards, which always seemed to turn up on a Friday afternoon. We would pluck our reports from the mailbox and reinsert them Monday, so the weekend wouldn't be ruined. This trick was just a childish way to postpone the inevitable, which is exactly what engineering groups do when they jiggle triangles like this.

Project-planning software is, of course, touted as an advance over the primitive, manual tools we used to create meaningless schedules in the old days. Now we can fabricate incorrect data even faster. That's one of the beauties of computers: once it took seconds, even minutes, to make a mistake. With computers you can make thousands of mistakes a second.

People have been writing software for over 50 years, and building embedded systems for 30. The one constant over all of that time is that features increase while schedules shrink.

We're trying to manage three conflicting things: an impossible schedule, an excess of desired features, and quality. Remove just one leg of the three, and the project becomes trivial. Can we ship with lots and lots of bugs? If so, getting it out on time is pretty easy. Can we neglect the ship date? With infinite time, we can get every feature working right.

This twisted triad dooms projects from the start when developers and management just don't recognize the truth buried in the conflict. The boss invariably wants all three legs: on-time delivery, perfect quality, and infinite features. He can't--and won't--get them.

It seems logical that we must manage features aggressively, since schedule and quality issues will always be non-negotiable. Use requirements scrubbing to identify and remove those features that really are not needed. Build the system in a logical manner so that even if you're late, you can still deliver a product that does the most important things well.

There is, of course, one other ingredient that forms the backdrop of the twisted triad, one that is more the fabric of the development environment: resources. Decent tools, an adequate supply of smart people, an enlightened management team, all form the infrastructure of what we need to get the project done.

In the 20th century, we learned to build embedded systems, but it seems management never figured out the appropriate role of resources in development projects. Somehow engineering projects are viewed much like building widgets on

Navigate to related information

Design: Memory overflow: Getting here from there

Discussion: Scheduling Momisms, or lessons your mom should have taught you

Discussion: Good contracts make good programs

Design: Employ the proper flash memory in your design

Design: DDS and the future of complex, distributed data-centric embedded systems



Most Popular

- 1 USB Explained: An Introduction to USB and Its Future
- 2 Digital Signal Processing: A Practical Guide (Part 1)
- 3 Fundamentals of Wireless
- 4 The Inefficiency of C++, Fact or Fiction?
- 5 Digital Signal Processing: A Practical Guide (Part 2)
- 6 Digital Signal Processing: A Practical Guide (Part 3)
- 7 Comment: Andy Grove, startups and job creation
- 8 Digital Signal Processing: A Practical Guide (Part 4)
- 9 BLDC motor for Electronic Bike Application
- 10 Digital Signal Processing: A Practical Guide (Part 5)

Product Parts Search

Enter part number or keyword

SEARCH

Get Involved

[Start a Forum](#)

[Subscribe to a Newsletter](#)

Are you connected?

There's a new place for designers of embedded internet

a production line. Need more widgets? Add more people, more machines. That just does not work in software engineering.

Fred Brooks, in his wonderful book *The Mythical Man-Month*, shows how adding people to a late software project invariably makes it even later. That two developers have only a single communications channel between them, but as we add engineers, the number of memo/meeting/e-mail links goes up with the number of people squared.

IBM found that as a project's scope increases, software productivity goes down--dramatically--for the same reason. Their surveys showed code production (in number of lines per day) fell by an order of magnitude as projects grew.

Barry Boehm's Constructive Cost Model is probably the most famous predictor of software schedules. It, too, shows that timelines grow much faster than firmware size. Double the number of lines of code, and the delivery date goes out by much more than a factor of two. Sometimes *much* more.

Yet "go hire some more people" seems the universal management mantra when a project plunges into trouble. It simply doesn't work.

Is there no hope? Will projects always be doomed to failure? Is the pressure so aptly illustrated in *The Soul of a New Machine* our destiny?

With project complexities exploding, it's clear that unless we dedicate ourselves to a new paradigm of development, using so much that has been learned about software engineering in the last half-century, we'll stagnate, wither, and fail. Those companies that accept new modes -- and old proven modes--of thinking will prosper. Two areas in particular are critical for new understanding, two areas that this volume deals with: reuse and tools.

« 1 2 »

print email rss SHARE post comment 0 Comments

Please sign in to post comment

Submit Comment

Follow Comments



More EE Times

- [Subscriptions](#)
- [Newsletters](#)
- [Reprints](#)
- [RSS Feeds](#)
- [Media Kit](#)
- [Sitemap](#)

- [About Us](#)
- [Privacy Policy](#)
- [EE Times Career Center](#)
- [Contact Us](#)
- [Email: feedback@eetimes.com](mailto:feedback@eetimes.com)

EE Times Network

- [EE Times Asia](#)
- [EE Times-China](#)
- [EE Times-India](#)
- [EE Times Europe](#)
- [EE Times Japan](#)
- [EE Times Korea](#)
- [EE Times Taiwan](#)
- [Electronic Supply & Manufacturing China](#)
- [Microwave Engineering Europe](#)
- [MCAD Online](#)
- [TechOnline India](#)
- [DeepChip.com](#)
- [Design & Reuse](#)
- [The RF Edge](#)

UBM
 All materials on this site
 copyright ©2010 EE Times Group,
 A UBM company
 All rights reserved

