Electronic Engineering JOURNAL

Feature Article

March 31, 2016

We are Teaching the Wrong Lessons

Giving a Computer to All 11-12 Year Olds is Not the Answer

by Dick Selwood

I had planned to write about new advanced technologies (quantum computing and the like) and to report on SEMI's Industry Strategy Symposium Europe, but instead, I have been distracted by the announcement that all British children in school year 7 – that covers children between 11 and 12 years old - are being given a computer. The aim is to re-enforce the government diktat that "all children should learn to code". All the talking heads that are being rolled out to support this initiative state, as an unchallenged fact, that coding is as important as English for someone in the early 21st century. The justification is that coding is the way to gain understanding of the digital worlds, including, inevitably, the Internet of Things.

What the children are getting is the BBC micro:bit, a 4 x 5 cm card (half the size of a credit card) with a Nordic Semiconductor microcontroller powered by an ARM Cortex-M0. As the micro:bit is an implementation of the ARM mbed platform (ARM being one of the partners in the project), there is already a pile of system software and middleware available. An NXP Kinetis with an ARM Cortex-M0+ controls the USB interface. (There is also a Bluetooth interface). NXP has provided a three-axis accelerometer and a three-axis magnetometer. The card has 256KB of Flash and what at first sight looks like a measly 16KB of RAM. There is a five-by-five grid of LEDS and two control buttons. Supporting it is a huge BBC website with material from a wide range of sources, multiple programming languages, lesson plans for teachers, and more.

The project was driven by the BBC (the British national TV and radio service), ARM, Microsoft, NXP, Barclays Bank (which already has a network of "Digital Eagles" across the country to help people use computers and the Internet), Lancaster University, Samsung, and a bunch more organisations. element14, already an active player in the RaspberryPi arena, has pulled the project logistics together and is manufacturing the boards.

To use the micro:bit, you need access to the Internet through a PC or smart phone. Programs are developed on the BBC micro:bit web site and then downloaded to the board through USB or Bluetooth.

Children will start to learn to code by altering parameters in existing programs and seeing how this changes output. For example, they can instruct an on-screen dog to move through a grid. They will also work through tutorials and eventually will be "messing about with code".

The roll-out has created a lot of happy people. It was accompanied by the obligatory positive noises by the senior people from all the organisations involved. The BBC ran it on radio news and current affairs programmes, and you could barely turn on the television without seeing some presenter saying, "Wow – I've just written some code", or some luminary in the entertainment world saying how important coding is. If it weren't for the Belgian bombings, there would have been even more coverage. The British government is feeling smug that it is bringing coding to schools without the taxpayer funding it.

While Britain is the first country where school children are getting a computer, the pressure to get everyone coding is worldwide. A few weeks ago, President Obama announced an initiative to get all American children to learn coding. Across Europe, other countries are following the trend.

Yet I am not happy.

Why?

Firstly, all the material I have seen is about writing code. There is nothing that I have found that deals with any other aspect of the development cycle. Specification, design, formalised ways of testing - none of this is present. This means we are going to have yet another generation of coders (Where did the word "programmer" go?) who believe that writing code is all that matters.

Last year, there was a long and fairly objective article on this topic (http://www.theguardian.com/news/2015/dec/03/should-kids-learn-code).

But what caught my attention was a description of how a self-proclaimed coding boot camp worked. After a selection process,

The survivors sit at long banks of screens, where they are supposed to spend nine hours a day learning to program computers, although most choose to stay long into the night, practising what they have learned. They work in pairs to bounce ideas off each other and, although there is a helpdesk for those stuck on complex coding problems, there is no teacher to give instructions. The idea is to encourage self-sufficiency and the ability to muddle through with peers, as they will have to do when they reach the workplace.

There is no suggestion that there could be existing tools and techniques other than *muddle though with peers*. A look at the boot camp web site suggests they are developing web coders, since they teach Ruby, JavaScript, and HTML 5.

How does learning to code make you better able to survive in the technological world? I happen to know, in too much detail, how the internal combustion engine works. But it doesn't make me a better driver. And of course my driving skills may, in a few years, be redundant if the autonomous car fulfils the promises its backers claim.

Many of the flag bearers for coding stress that there is a strong need for coders. There is a widely viewed video where Silicon Valley luminaries say how exciting coding is and what a great future lies ahead for coders, accompanied by photos of Google's and other companies' workplaces with catered free food and ping-pong tables.

But this is seriously misleading. A friend who is a software engineer once said, "Coders are the bricklayers of the system development team". This is a bit harsh, but the analogy is not totally without merit. Bricklayers are skilled craftsmen doing a great job, and there is a wide variety of expertise. But while they may work on their own initiative when building a garden wall, if they are building a house they are working to plans drawn up by an architect, just as coders do in a well-managed project process.

Sadly, the coding ethos, as currently used, generally also seems to preclude code re-use. Chip design makes great use of IP, either bought in, such as an ARM processor core, or re-use of internal material. Yet, **as Bryon Moyer recently pointed out (http://www.eejournal.com/archives/articles/20151123-synopsyshacked/#rant)**, web developers seem unable to re-use even simple pieces of code such as checking the format for phone numbers.

Another major issue is that, even if large numbers of these children do get programming jobs, then how long will they hold them? India and other lower-wage economies are also teaching their children to code, and off-shoring will become increasingly attractive at the lower end of system development. At the top end, the use of modelling and automatic code generation, creating code that is correct by construction and needs no debugging, is growing steadily. These two trends are going to squeeze out many programming/coding jobs.

It is interesting that no one is suggesting that people should all be taught circuit design, even though silicon is at the heart of the new technologies. Is this because it is accepted that creating hardware is an engineering discipline and recognised as such?

Yet in 1968, nearly 50 years ago and before most active programmers were born, software engineering was the subject of a major conference as people became more and more concerned about how many projects were running late and over budget or failing completely.

The proceedings of the conference are available online, and, with a certain amount of change (the only computers then were main frames), much of what was said then is relevant to software development today.

Two quotes:

"Software production today appears in the scale of industrialization somewhere below the more backward construction industries."

"Programming is still too much of an artistic endeavour."

The papers discussed things that, in today's jargon, we would call formal methods, test driven development, iterative development, and other "modern" approaches – 50 years ago, they were all there. Since then, while there are examples of good engineering practice in isolated areas, software has generally continued to make the same mistakes over and over again.

The IT world that the 1968 conference addressed is still full of examples of projects that have failed completely or only delivered late, over budget, and incomplete. As I was writing this, there were reports that the Metropolitan Police, the police force for London, has abandoned 21 projects worth more than £100 million ($140 million), with many more in trouble. Central Government computer projects are notorious for failure. In Britain, a major change in the way welfare payments are handled is running late because of poor IT. When these projects are analysed (which doesn't happen), the causes of failure are the same as those identified in 1968.

The recent **report by The Barr Group on their second Embedded Systems Safety and Security Survey (https://vimeo.com/158433552)**, reinforces this. There were results from almost 2,500 embedded engineers. Of these, 543 respondents said that the worst possible outcome of their product having problems was that people would be seriously injured or die; often there was the potential for multiple deaths. Of these, 67% said that their product met relevant safety standards, 22% said that their product did not meet safety standards and perhaps even more worrying was that the remaining 11% didn't know. For the "did-nots", there may have been a reason for not following the standard (although it is difficult to see what it might be), but what were the "don't knows" thinking? Now if you were writing software for applications that could kill people, you would think you would follow the best possible engineering approaches and use everything available to make sure that the code was of the highest quality. But 30% didn't use static code analysis, 16% didn't have coding standards, and around 40% either didn't or only "maybe" undertook code reviews.

And this is the industry into which we are hoping to attract young people by teaching them coding.

One argument is that coding teaches problem-solving and logical skills. There is already a mainstream subject in every school that can do just that. It is called mathematics. Let's leave aside what you and I would think of as mathematics and concentrate instead on basic numeracy. If a fraction of the effort being thrown at coding were to be devoted to making sure every kid left school with basic numeracy skills and sufficient understanding of basic statistics to examine critically statements such as "when given this medicine the number of those who recovered doubled", the money would be well spent.

Channels

Career (/design/career). **Computers (/markets-industries/computer)**. **Maker & Hobby (/markets-industries/maker)**. **Software (/design/software)**.