Some personal notes on multidisciplinary development

Ton Kostelijk

This text contains notes on my experience with some disciplines: software and mathematics, embedded software and hardware, proceeded by a more personal introduction.

Several months ago, I was invited to put some words on paper about my prejudices of different disciplines. Thinking about it, and extending it to the cultural issue as well, I realize that I have crossed a lot of "cultural" boundaries in my profession and because of my personal background.

I was born in 1961 as the 11th child of an agricultural farmer, a religious family, in a small village in a polder (North-Holland). In particular here in Brabant, I still miss the wide outlook over the land 20 km far, the waters and wind that surrounded me there. With the sixties and growing welfare, a lot changed in my first 20 years. My eight brothers in particular were quite outgoing and paved my path to be allowed a lot of freedom in behavior. Already with my scooter, I visited my eldest married sister at the age of 4, a trip of 5 km away (I was very well taught to duck to the side when a car passes). As one of the very few of my town, I was educated at the Athenaeum, in the nearby (12 km) city. I realize now that it was the first time of many that I joined a new group, where many of my old group did not follow that step. In the city, things go very differently. My circle of movement kept on growing, and I decided to go to the Free University in Amsterdam, to become a scientist.

In different ways, most of us cross cultural boundaries, even when you are Dutch, and you live in The Netherlands. Differences between groups are an interesting subject to discuss, provided that it is done with the willingness to understand. When done in a reproaching or even accusing way, there are only losses. However, in the end the differences between individuals are larger than the differences between groups. So I propose you read the remainder with some relativism.

My education (experimental solid state physics) combined a strong mathematical foundation with pragmatism and a sharp eye for deviations in expected behavior. In short, physics is all about constructing, rejecting, validating and extending models of the experimental reality. Mathematics is about models of the imagination. When an experimental model is equivalent to an imaginary model, knowledge of mathematicians can be used.

Software and mathematics

Some people think that programming is a form of applied mathematics. In my view software originates more from engineering than from mathematics. I respect Dijkstra for his contributions but I disagree with his rigid view on software development. Even Gödel's incompleteness theorem has proven that proofs and completeness are not fully united. Still there are people that keep on searching for the ultimate software development method or tool that will enhance productivity enormously, and prevent any mistake that happened in the past. Instead of posing the question "what is the best method for all systems?" one should pose the question "what is the best approach for my system?" In this way, mean and lean solutions most likely occur. First of all, one can benefit of the intrinsic structure of the problem at hand. Secondly, the notion "approach" instead of "method" indicates which route to take without claiming to know the solution. The typical problem is too difficult to tackle in one go, leading to another element: usage of an incremental approach, or in other words, a spiral development model. And last but not least, the job is performed by a (set of) group(s) of people, where experience, communication and other cultural items play an important role.

Nevertheless, one can benefit a lot from mathematical insights, just like physics benefits from it, without implying that physics is a sub-domain of mathematics. Since problem / system modeling is apparently so important, it may no longer be a surprise that so many software architects originally studied physics.

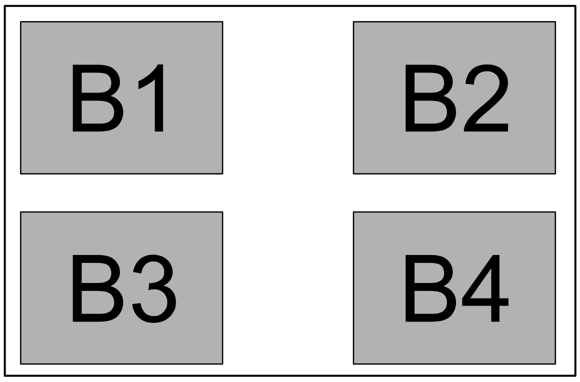

Figure 1: Typical hardware view of a system

Embedded Software and Hardware Development

There are several differences between hardware and software.

First of all, the definition of a system is quite different for hardware or software. For a hardware designer, a system is a collection of hardware blocks that are interconnected. For a software designer, a system consists of several layers; an application (including services) runs on drivers where drivers are “almost hardware.” Whereas for hardware designers interrupt routines are considered out of their scope. This implies that a gap in responsibility exists, the hardware software interface.

Secondly, embedded software is mostly engaged in handling use-case transitions, whereas hardware designers are focused to make a block process well in a steady-state use case.

Thirdly, hardware typically is designed bottom-up (“re-use”) whereas software is typically designed top-down. This latter statement may no longer be completely true: because of high development cost, software re-use is growing rapidly, resulting in more and more glue code in systems. One may wonder whether the amount of glue code is in balance with the shielded amount of core code.

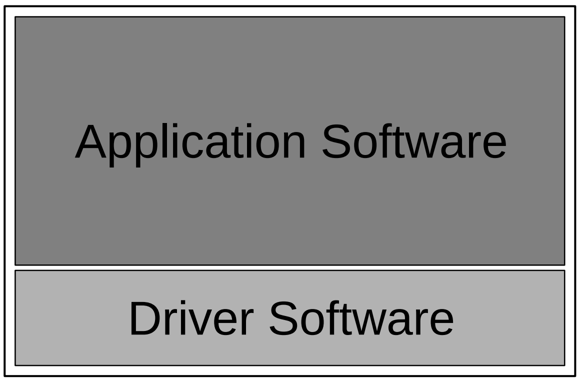

Figure 2: Typical software view of a system

Fourthly, hardware has real concurrently running functions. Software only has timesharing, where concurrency is faked by the operating system. The software performance is deteriorated many factors by limited caches, uncached access, context-switches (interrupts in particular and task-switches) and busload. As a result, hardware designers typically overestimate the software performance of a system. The average software designer is not at all focused on performance whatsoever. This means that performance set-backs are common when execution architecture view is not elaborated in a system.

Fifthly, one of the major separating issues is that software development lags behind one or two generations of the hardware development. In other words, the hardware and software focus differs.

Finally, software suffers from hardware bugs, not vice versa. On average, 30-50% of the effort to make drivers is related to bug fixing, unclear specs, system errors, etc. One of the opportunities is to keep the overall balance sound: whether to save effort in hardware design by spending effort at the software side, or vice versa.

Final remarks

The cliché that technical software people talk to computers only and work in isolation is so beside reality. This paper could have grown many pages more because making products involves so many different disciplines. In the global world we live in, it involves many different cultures and nationalities as well. In our work, this gives extra color to our profession. It is also the reason why the educational focus of an architect shifts from technical towards social during his or her professional life. Since the differences between individuals are larger than the differences between groups, it helps to forget the group your colleague might be part of, and appreciate the unique person you are in contact with. In case of misunderstanding, the group habits may be a cause though, but firstly focusing on the group clichés ignores most value of the person. Multi-disciplinary work can thus become an adventure of synergy and appreciation.

About the author

Ton Kostelijk Born in 1961, Married. From 1979 to 1985 Ton Kostelijk studied physics, Experimental solid state physics with IT, at the VU Amsterdam. From 1985 to 1995 he worked at Philips Natlab on CAD for VLSI Design, for which he received his Ph.D. in 1994. Thereafter, he was Chief software archtict Digital Receivers program (G+4 settopboxes) at Philips' ADC/ASA until 1999. Currently, Ton Kostelijk works as system architect at the Philips Digital Systems Lab in Eindhoven.

Currently engaged in

- System Performance Feasibility
- Member of Core Architecture Team Disk Systems product family.
- Coaching of architects
- Chairman of "QITARCH" of PDSL
- Teaching courses for CTT (3), ESI, OOTI.