Today's Software Bloat Part 1 – Hardware Abstraction

Why software development couldn't improve even with Java

Onkobu Tanaake

2018-09-20T10:00:00

I started coding software in 1993. It was QBasic on a KC85/3, top computer platform of GDR and available at my school. Until then I moved through various level of expertise and have always been involved with high scalability projects of BKA (German Federal Investigation Bureau), Deutsche Bahn/ Schenker (Railway/ Transportation) and energy markets as well as automotive backends. Non-IT People always ask me, why I don't make it much simpler and they even suggest a solution. This two-part-series explains, what separates experts from average people.

Write down current time with hours and minutes. I'll take a very simple machine from the real world, a toaster. It basically consists of a heating surface, a spring loaded lever system, a timer and some electrical control mechanism. If the lever is pulled down the heating surface is energized through the electrical control and the timer is started at the same time. When the timer is elapsed it somehow triggers the lever system to push out the bread.

Think about this a minute or two. Maybe even go to your kitchen and inspect your toaster. I really recommend to do so. On the one hand stepping back from a problem – in this case understanding my very abstract description of a toaster – gives your brain time to figure out the details in the background. Being in motion helps to really put the background tasks into the background. You could also multiply 2-digit numbers for 2 minutes, make 20 sit-ups or take a map of your surrounding and find a landmark like a single tree, a church, river crossing a street or similar.

Now be honest: could you construct a toaster from this description? A toaster one of your relatives, friends or collegues would use or even spend money for? Would it be safe and not burn down the house? I'll make it easier: explain the construction to a workshop, as one of the workers who surely never made a toaster before, to follow your instructions to built one. If your answer is yes, there are at least three options:

1. You're a product designer and toasters are your favorite design problem
2. You're fond of kitchen devices or a historian, especially for toasters
3. You're a true software developer/ home improvement expert

Let's stick with option #3: you're over-estimating your capabilities, under-estimate the effort, push theory aside and don't care about other's experience – like me. I'll outline this with a few questions:

- Does your device apply to laws and regulations for electrical devices of at least a single country?

- Did you take your labour into consideration, not only the bill of materials?
- Can you draw a detailed plan, ready for manufacturing, with correct scale?
- Why should someone buy your toaster and not one off the shelf?
- Does your design take excessive heat, flammable remains (small crumble) and potentially water into consideration?

I didn't even go into the details of setting the timer and whether the heating surface is one- or two sided or switchable between one- and two-sided toasting. There are (barbarian?) peoples in the world who »like [their] toast done on one side« (Sting).

A toaster is a rather compact machine with few so called states, controls and transitions. For example the starting condition could be not being plugged in and a transition the lever being pulled down. It is up to you to decide, whether the lever mechanism locks in or not even when not being powered. I prefer the safe way: without electricity no transition is possible, the lever comes up again. Let's go even further, with no bread added neither the lever locks nor does the heating start. This transition is coupled: Heating starts if the lever locked in. The lever locks in if the device is plugged in. Make it a chain:

1. Plug in
2. Lock in lever if power available and bread is inserted
3. Start timer
4. Switch on heating surface
5. Check if timer elapsed, if so, unlock lever and switch off heating

Perfect, we developed a sort of first control software. It is rather abstract but contains the basic elements like a start condition, transitions, user interactions, a control loop to check for the timer and some actions depending on conditions with logic operator *and*.

Write down current time with hour and minutes under the first timestamp when you started reading the text. Calculate the difference. Also have a look at the number of paragraphs. Even sit back for a couple of minutes and find at least two surprising changes of mind while you read this text. There was a view on the world and its toasters before you started to read this and now this world has changed. I am different from you, we have totally different concepts of the world. Surprise is an indicator for undermining your internal model of the outer world, of not meeting your expectations. Is it time yet to draw a plan with correct scale or is it still necessary to answer fundamental questions, examine existing toasters, the environment it is running in, constraints of its usage, your design budget and so on?

Now do you really think, something *easy* like a toaster can be extrapolated to something complex like a computer? Let's only take the first seconds into consideration: power is applied and somehow a first program needs to be started. It is not necessary to load an operating system, just something to give a clear signal to the user: no need to check the guts, computer is ok. Is there memory, a central processing unit, an output device, which device, how is it accessible? Imagine your computer being able to beep, boot from a USB disk or pen drive or over a network, even over the air.

A toaster serves a single purpose: heating things a certain amount of time in a safe way. In contradiction a computer serves no specific purpose. You can read messages, it can play music, you could design a toaster as a 3D model and turn it into a project of computer aided manufacturing. Browse the internet, upload content, crop images or »work« with it. Also take into consideration, that your mobile device is a computer.

This multitude of hardware, software, requirements, purposes and constraints requires a lot of layers of abstraction, one over the other and transitions and control mechanisms to bring up an environment a developer can deal with. I coded for HP-UX Super Domes as well as consumer PCs or mobile platforms – totally different languages but similar concepts. Luckily there is the powerful tool of abstraction. But most of the time this abstraction fails and a mobile device differs dramatically from a mainframe computer, when it comes to the non-function requirements. Maybe execution time (read a 1MByte file in under a second), response time (upon user interaction give a feedback within 50ms) or throughput (at least 1.000 requests per second) put a constraint on me as a developer, caused by the user's experience or expectations.

From the outside it might look like a metal box with a plug and some magic inside, similar to a toaster. You put energy in and somehow it is turned into heat and user experience. But it's been a long way from simple heating apparatus to highly integrated computation devices and the speed of this evolution is unprecedented. A developer's tools must keep pace with this speed up and accelerate it even more. But this belongs into part two. But to conclude my toaster example: While reading this and thinking about this you moved away from the average person towards a kitchen device expert. Hopefully you can see the world around you with more detail.