Using code of others

Peter Thoemmes, 2004-02-27

Very often programmers are asked following kind of question by the manager: „Hey Bob, why don't you use the code of Rob? He has done the same kind of thing and it is working fine.“

Finding an answer to such kind of questions is often not easy for programmers, because the question brings them into a hopeless situation:

1.) Software engineers can't understand how somebody can raise such a question.
2.) Managers can't understand why there is no simple answer to that question.

I try to support both the programmers and managers by this paper and I hope I can.

A programming language is just like a toolbox with many different tools, like a mechanical toolbox: there are wrenches, screwdrivers, drills, files, hammers, and and and. What you do with it can reach from the making of a small metall box up to the making of a modern car and much more. To build a modern car there is other tools used, that have been made before (even complicated robots). The variety of things you can do on top of the basic toolbox is so wide that the resulting products do have nothing in common, like a toaster and a Porsche.

Software engineering is like car manufacturing. Some of the programming languages („the basic toolboxes“) are so universal and so basic that they are used in almost all branches of software engineering, like real-time-operating-system development using 8-bit-micro-controllers on the one hand and web-server development on top of scalable computer-clusters on the other hand. Most famous examples are the programming languages C and C++.

Now, if you look into one development department („one car manufacturer“), then there have been established many ideas over the time and many rules to follow when using the basic programming language („the toolbox“). Sometimes there is a powerful server that runs multithreaded („the car engine“) and only a few engineers are intelligent enough to work on it, and only some engineers are able to tune it in a way that is is powerful and will run for a long time. These things are kept secret by the development departments, as they cost lifelong research and practice and they build the fundamental difference between the company and the competitors on the market. Very often the work is based on just one mind (see Linux or the Qt libraries) and if management understands that mind, they give the guy (or lady) the freedom to build many great things on top. Also they give him (or her) the time to train others to use his ideas the right way. Ideally the guy should also have the time to document the whole lot of ideas and rules, which sometimes costs years.

Everybody understands that in car industry not even simple things like car seats or steering wheels can be taken from one car (say a VW Golf) into another (say a Porsche), not even when the company belongs to the same manufacturer group. Although sometimes parts might look quite equal, we all know that they will never fit. And we are **not** talking about the seats of a tractor and a formula one car here.

In software engineering this is different. For non-developers the code looks the same, if it is lousy code from a newbie or high quality code from a perfect expert: I see you do C++ („you use torque wrenches“), I also know somebody doing this – maybe you should talk to each other...

Non-developers don't see if there is a set of hundreds of rules to be in one's mind while writing the next line of code, of if somebody simply only adopts an hello-world example to his needs.

Teams like *peter-thoemmes.org* established big sets of such rules, for example: **High stability** leads to rules like 'never use the heap by *new* directly but by smart-pointers or STL containers'. **High performance multithreading** leads to rules like 'never use static memory by threads before the main thread initialized the lock objects'. **High effective maintenance** leads to rules like 'always divide projects in layers' and 'never program any algo twice, but put it onto the deepest layer possible' and 'never make a layer visible to the same or lower layers'. **High portability** leads to rules like 'never use any filesystem or network access directly, but through the lowest layer code'.

Big and well maintained code bases are full of such rules. Although sometimes the rules are written down, it takes years for a programmer to become able to write safe, performant and stable code. This is directly comparable to the learning of a human language: the fact that there is a dictionary on your desk will never make you speaking a new language. It takes a trainer and a lot of practice.

Once a programmer is able to use such a big code base safe and well, he faces another problem: he can write complex and stable applications in just a few days, but management thinks that is an easy job: inner complexity, high stability, low maintenance and the required skills of the programmer are things not visible to externals. Only the full understanding of the product and objective tests can show what the product is like. And never compare apples with pears here!

And now to the answer of the question: „Hey Bob, why don't you use the code of Rob? He has done the same kind of thing and it is working fine.“

Rob has not done the same kind of thing. It just looks like. He is using the same basic toolbox, but different rules, as he is working in another development department. His code can never be taken as is, just the idea behind can be taken and we can write new code to make it fit into our project. BUT: we might have much better ideas and it might cost less time to follow those rather then follow Rob's approach. From time to time we have a look at what others do, but we focus on our own creativity rather then copying other ideas. Copying is never a deep-copy here, as the creative brain behind is not in our team. We better use our own ideas, because we fully understand them.

REMARK:
Sometimes managers do force their developers to open there ideas to externals to reach a short termed personal goal. That's one of the biggest mistakes a manager can do. He not just demotivates his main creative people by the outmost extend, but he gives away the core competence and secrets of the company. Most of the time this is not legal and against the code of conduct.