Requirements *ad Verso*

By Sergey Kucherov
(May 19, 2006)

Long, long time ago there were people called programmers. These people used computers to write programs and they used these programs to make the computer do what they wanted it to do. The software they developed was usually pretty good, because programmers knew exactly what they wanted, they tested their programs themselves, and they already knew how to use it.

There were other people called users. They had computers too, but did not have time or skills to write their own programs. So, they borrowed software from programmers and learned how to use it. Some programs were easier to use than other, but users never complained, because "the software was provided 'as is' with no warranties whatsoever and the entire risk as to its quality and performance was borne by the user." In time, the user-friendly programs became more popular and the idea of selling programs became more appealing to the programmers. This is how the era of commercial software started.

After programmers sold all the programs they wrote for themselves, they needed to develop more programs for sale. This time, however, the task was more challenging. First of all, when you make something for other people, you need to know what they want – in other words, you have to know the requirements. Second, if you are not planning to use your own product, then you need another way to debug it – in other words, you need testing. This is how the era of software development methodologies started.

There are many methodologies for software development and design and *all of them* suggest to start the process by collecting and documenting user requirements. It makes a lot of sense: if you want to create software, then you need to find out what the software should do.

Different methodologies suggest different ways for analyzing, documenting, and tracing the requirements, but until now, nobody dared to suggest that you can develop software without requirements.

What are the software requirements? A set of verifiable statements about features of the software, ways of using the software, and the expected result of such usage.

The most important word in the definition above is *verifiable*. If you cannot verify the statement then it is not a requirements but just wishful thinking. For example, the phrase "The program should work on a PC running Windows XP" can be considered a requirement, but the phrase "The program should have a sleek and sexy user interface" can not.

If you start a software development project and if you already have all requirements documented and approved, then it will make your life, as a developer, much easier if you use this valuable piece of information in your development process. Moreover, even if you do not have the requirements yet, then you should ask for them. Maybe you will get lucky and your customers – users of your software – know exactly what they need and will share this knowledge with you. However, if it did not happened and your attempt to get requirements from users has failed, I very much doubt that any further attempts will succeed.

I mean, of course, you can beat it out of the user, but the requirements obtained this way will be worthless. Users will give you a pile of garbage just to get you out of their faces. You do not want those requirements, believe me. After all, evidences obtained through torture are not admissible in court.

You can also take the high road – you come up with your own vision of the requirements and then you ask users to make necessary changes. This approach works very well in situations when a signature under the document means more to you than a working software and customers satisfaction.

Finally, you can have a workshop. God, I like this word. It usually works this way: you lock your customers together with members of your team and start asking them questions. You would believe, that this is the way to find consensus and identify the real and complete requirements for the software. I would not count on it.

Your customers will not tell you the requirements, nor will approve your version of it, nor will disclose it to the interrogation team during requirements workshop. They will never do this, but not because

they enjoy seeing the suffering of software developers and their managers. No, they will not give you the requirements because they *do not know* it!

Why should they? How can they?

If it is new software, then the users have never seen it, they never tried it, and they cannot imagine neither pain nor joy of such experience until you install such software on their computers and let them try it.

And, as you know, the primary goal of all those software development methodologies is to avoid just that – spending time and money developing the software until you know for sure what exactly users want. Congratulations, my friend, you are stuck!

Now it is the time, when you expect me to tell my plan. I hate to disappoint my readers, but I do not have one. I am afraid that this is just one of those problems, that cannot be solved. But I believe I can solve another one.

Do you remember the old good software that programmers used to develop by themselves for themselves? The ones they developed without any requirements nor formal testing? Programmers did not need to document requirements because they, being also the users of the software, knew them already. But what about the others who bought this software from the programmer? They tried it, and they *learned* how to use it. This is the answer – I cannot solve the problem of a programmer who needs to write programs without completed requirements, but I have a solution for the users. If you need software, then let programmers do their job and develop software the way they feel is right. Then you install the program, try it, and learn how to use it. If the software is too hard to learn, then look for another vendor.

It sounds good, but probably will not work. Not for business users anyway. They do not have time to try and learn software, and more importantly, they cannot afford it. For example, business users cannot just try to submit an invoice using a new application, because if it fails, then it will cost them much more than fifty bucks and half of hour of wasted time. It may cost them a customer and it may cost them a job.

This fear of trial is the biggest problem for commercial software, and it needs to be solved. This is what I suggest: instead of asking users for the requirements, let them list all prohibitions. Instead of documenting what the application should do, users will write down the list of things that should never happen. Let users tell you all their fears and frustrations – this will ignite their interest.

I know what you are thinking: it probably means just more work for you. Think again. By starting the project with asking for anti-requirements you have a chance to get more accurate information, because users know their fears and frustrations very well. You also will be able to create more meaningful test cases and plans, because you know exactly what to test for. And, what is more important to you as a developer, you will finally get the freedom to design your software the way you like it. Think about it – every existing software development methodology pushes you to document requirements, and then it forces you to derive the software architecture and design from these requirements. Instead, if you deal with well defined limits, then you have a *carte blanche* to do everything you want as long as your software does not extends these limits.

The business user, who is afraid to test new invoice applications, will tell you that he does not want your program to loose the invoice. He also does not want to sit in front of the monitor wondering whether his customer has received the invoice or not. And he wants to be able to send the invoice from the old system if anything goes wrong. Now you can design the software any way you want as long as the user will not experience the problems listed above.

Of course, this approach has its down sides. First of all, it does not eliminate the problem of users who change their minds in the middle of the development cycle. You still need to have change control, version management, and the signature of your customer, and you will need to test the software. Finally, probably the biggest problem of all, you will not be able to take advantage of most of the existing software development methodologies, because they all rely on the requirements that you will not have.

If you decide to test my theory, then make sure your users are aware that this is a new approach. Make it clear to your customers that as long as the software does not cause any problems listed in the document they have signed, you are free to design and develop the program any way you want. Users should also understand and agree that they will have to spend some time learning your software before they can enjoy its benefits.

Let me know how it went.