Re-use & Componentization of software development

Arjun Venkataraj
Smriti Metikurke

The idea of software reuse is rapidly catching up in the software industry. With more complex software being built and deployed, the degree of unknowns would be the key in building effective bug free software's and reduce the lead time to the customers. This brings a relatively old idea of software componentization into the forefront in today's world.

So what exactly is this Software reuse?

Often people relate this term to the usage of source code. This is not the case. Software reuse can be to do with any asset of a company starting from the requirements, design models, architecture, code documentation, user manuals, test plans, test scenarios etc that could be effectively utilized.

If one imagines software to be made up of several layers and all layers communicate among itself and to the shell. Thus having a basic framework will help in identifying a common building block .Even if one of the layers is built on a common framework, which can be used further without much rework, it will reduce the unknowns and thus help increasing the predictability of the software.

So the goal of a software reuse here is to use as much data as possible and reduce the time, cost and risks associated with re-development.

No doubt that it helps in the entire software cycle, but there are certain areas where the accrual of benefits is more than the others and the major areas being Development, Testing and Automation. Not to mention the ultimate aim always to have lesser bugs and a satisfied customer.

So how does this all fit in what we do? Well the bigger picture is that all these are spokes of the same giant wheel called "software". So when we talk about the bigger picture let us see in what spheres of software cycle it makes a difference.

Development:

Developers will have a common framework which can be reused atleast as a basic building block. The concept of "Low Coupling and High cohesion" comes into play. With low coupling, a change in one module will not require a change in the implementation of another module. This would also reduce/eliminate the domino effect of having to rewrite modules leading to better readability and easy maintainability.

Thus developers see a "Reverse domino effect " where they can reuse more and more since inter modular communication becomes more and more generic and in turn reduces anomalies of interaction between modules and increases visibility and fuels the bigger picture faster.

Testing:

The idea of component based software turns the focus more on integration testing. With the intermodular communication being simpler, thus reducing issues of interface level defects. Even if the defects are found, a fix in one area would help fix multiple areas.

Benefits:

- Reliability of the software as a whole greatly increases
- Only the files and functionality that are needed will go in. In effect it will be a "PULL" process; only what is needed will go in.
- Reuse becomes easy as dismantling and rebuilding the components gets easier since boundaries are well defined.

Automation:

Since the components have clear cut boundaries with "Low Coupling and High cohesion", it would become easier to automate the integration testing. Once automation takes over the basic component based end user scenarios are covered earlier in the development cycle and reduces the defect turnaround time.

Conclusion

Instead of re-inventing a wheel, using the same wheel with more innovative ideas can help do wonders. Benefits of this might not be visible during the early stages of implementation or at an individual level. But for an organization as a whole, this step can lead to better productivity, quality and reliability of the software's used. It can even result in faster delivery of products resulting in customer satisfaction.

All said and done, it's not an easy task to start with. It requires a lot of planning, investments and commitment. Above all people need to be educated about the benefits of reusing the software which could be a first step to successfully developing and implementing "software componentization".