CrossTalk would like to thank NAVAIR for sponsoring this issue.

The Pros and Cons of Code Re-use

Software represents a significant investment for any organization and the quest to lower that upfront cost to field a capability and make use of that investment as technology evolves, delivery mechanisms change and requirements "morph" is an ongoing effort for all organizations.

Abstraction layers, containers and wrappers, middleware, and application frameworks are some of the approaches currently being used that allow the re-use of software. The interface and boundary layers in these systems and the degree to which the design is modular, not tightly coupled and uses non proprietary components and interfaces leads to the open architecture label and impacts total cost over the lifecycle.

The decision to re-use including COTS, GOTS and "open source" versus developing new software is an important consideration in the estimation and planning process as an efficient design that meets only your requirements can have a lower cost to integrate and maintain over a module or application that does multiple and potentially unknown functions or is "tightly coupled" with other software. With an efficient approach to the use of existing code it is worth considering if the code is modified versus re-used.

A common assumption for re-use is that the software is untouched at some module, app or aggregate level and the cost savings in estimation is based on the maturity level of the product being re-used. Many times modified code is called re-use or subtle distinctions are made that allow modified to be called re-use when less than five percentage change is made to the baseline code. It is difficult to realize the expected savings for re-use when the lineage of the code is untraceable and the artifacts that go with a module or app like requirements, unit test history and defect density either were never tracked at that level or are no longer relevant at least at the test level based on modification. Going forward all software developers should be considering the partitioning of functionality and interfaces in their designs that would allow efficient re-use. Minimizing modification and the overall size of the end product will lower the cost to integrate and maintain.

Defect density is not universally tracked and the necessary reliability of the end product is driven by the application. Safety, security, and reliability are all end product application requirements that should be addressed early in the development process. Reliability and functionality to protect against errors and failures is a driver for software cost based on more stringent requirements, different or potentially modified development processes, and increased test requirements. It is worth considering the inherited properties of the end product based on re-use and the introduction of schemes that can provide isolation and minimize risks. These system level design considerations are difficult to make after the design is complete and they need to be made early in development at the architecture level.

No matter what software development process you follow, waterfall, agile, etc., it is important to understand the requirements and interfaces associated with the modules, applications or aggregate software you are developing and integrating. When decisions of make-or-buy and re-use including are made, those interface and requirements are fixed and cost will be a factor if changes are necessary.

The prospects for better-faster-cheaper products as we evolve to new delivery environments and mechanisms is exciting, but on our journey to develop "open" architectures that will allow us to re-use the investments from many programs we cannot lose sight of the actual product we are reusing as it is this product at the lowest level that will be the source of savings or inherited lifecycle costs.

G. A. Graton

Gary Graton
SW Engineering Manager
NAVAIR SW Engineering Division