Off-the-Shelf Software: Practical Evaluation

Lee Fischman and Karen McRitchie
Galorath Incorporated, The SEER Product Developers

This article provides practical advice on evaluating off-the-shelf software by exploring the constellation of success factors common to all such software, from stand-alone applications to low-level components. The article attempts to reduce much of what has been discussed in previous literature into a few succinct sections. The factors that we develop are particularly suited to evaluating contractor proposals for utilizing off-the-shelf solutions.

With defense and commercial technology so tightly interwoven, developers must evaluate commercial and other off-the-shelf[1] options—and a client must evaluate the developer's proposed solution—in practically every project undertaken. An evaluation is best carried out by engaging in a cost-benefit analysis: costs and risks of implementation vs. benefits delivered. This paper presents the criteria for this analysis.

No Free Lunch

Off-the-shelf software may offer tremendous benefits but there is no such thing as a free lunch. Any department manager forced to support desktop software and locked into an endless upgrade cycle can tell you that.[2] The essential drawback to off-the-shelf solutions is that they do not allow absolute control over the origin and often the ongoing baseline of code upon which they rely.

These are a few of the cautions that go with using off-the-shelf software:

Items are often unique. Uniqueness demands that developers adapt to new ideas and methods, which can be time-consuming and costly.

The efficacy of stand-alone applications is debatable. Advertising does not always translate into reality, and this may lead to costly technical resets.

New paradigms may be unwelcome. Off-the-shelf technologies may satisfy user needs, but may be incompatible with current doctrine.

Implementation may not be rigorous enough. An off-the-shelf item may not be developed to the performance standards required in some applications.

Coverage may only be partial. Off-the-shelf software may not be intended specifically for the target application and so may offer only partial solutions.

Vendor support may be critically unavailable. Outside developers are not directly under your control and are often not available when most needed.

*Products may be volatil*e. Volatility can increase maintenance costs as technical-refresh requirements are increased. As support for a baseline technology disappears, advanced obsolescence may be the result.

Do not let everything above scare you, for off-the-shelf software still is fundamentally a good thing. The task is to maximize the return on its use while minimizing potential drawbacks.

Off What Shelf?

The many flavors of "off-the-shelf" range from fully commercial-off-the-shelf (COTS) to simply reused code. Our classification scheme is provided as a framework for thinking about the success factors that follow, and also to dispel the notion of COTS as a monolithic entity. Everything in Chart 1 can be COTS.

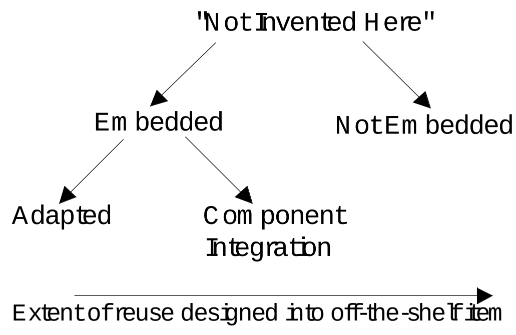

Chart 1 *Framework for considering COTS success factors*

Nonembedded items are essentially stand-alone applications, while embedded items undergo tight integration directly into delivered systems. The latter are further divided into adapted software (old code, designs, modified applications) vs. components intended for reuse (programming libraries, application program interfaces, stand-alone code such as DLLs). As Chart 2 shows, the choice between using embedded and nonembedded items partially rests with how specialized the application is.

Integration Factors

To get an off-the-shelf item ready for use, you have to integrate it. Integration is the cognition and effort required to get something you did not create to work properly. This can be as simple as installing something like a commercial word processor on your desktop, or as complex as the big-team, multiyear effort required for integrating enterprise resource planning software.

The following sections cover the interrelated components of an off-the-shelf implementation analysis: criticality, scope, and "meta" and "micro" success factors. Analysis components must be linked to assess the significance, risk and prospects for fulfilling integration (See Table 1).

		Effort	Fulfillment of Project Goals	Risk to Overall Project
Scope	Size of the job	X		
Criticality	Criticality of integration effort to overall application requirements			X
"Meta" Factors	High-level indications of success		X	X
"Micro" Factors	Low-level indications of integration effort	X		

Table 1 *Assessing significance, risk, integration prospects*

Criticality

If an integration effort fails, what must be done to insure the entire project is not jeopardized? Any management approach with (the hope of) strong risk controls must extend its realm into off-the-shelf issues as well:

- Do alternate off-the-shelf options exist as backups?
- Has an up-front evaluation insured that failure will not occur?
- If this off-the-shelf integration effort fails, can the project stand to lose the functionality?

Scope

Integration factors are only useful when the scope of the integration effort is understood. For example, a large project with a small integration task does not face much risk due to integration uncertainties. We present two scoping methods depending on the amount of knowledge you have:

Quick Sizing—Although an imprecise method, quick sizing does not require much information and it can be used either with code-level or "black box" integration. The idea is to establish the magnitude of off-the-shelf items by deciding whether (by fuzzy analogy) they simply are small, medium, or large. This magnitude is then adjusted by the proportion of the application that will be used. Table 2 illustrates the process:

Attribute Sizing—This method yields fairly precise indications of size, although its use is limited to code-level integration. A size metric of some sort is used to quantify the off-the-shelf software; we recommend the number of functions and data tables actually being used. If the software is object-oriented, its number of parent classes and methods across all instantiations might be counted. Alternate metrics could probably be substituted with similar intent, although correlation with effort needs to be determined.

"Meta" Factors

The factors laid out in Table 3 are most appropriate for high-level evaluation of reuse plans,[4] particularly when compared against column entries representing key management objectives. A client or program office could use these as a scoring chart to gauge the relative merit and risk of contractor proposals for integrating off-the-shelf solutions. We have provided our assessment of factor impacts with arrows indicating positive or negative effect; darker arrows represent a more definite effect.

There is not enough space to discuss how all row weightings in Table 3 were derived, but we will discuss the process behind "Maturity/Stability of Product" to give an idea of the thought process involved.

Development Schedule—The less mature a product, the greater the frequency of more significant upgrades. Waiting for upgrades could introduce development lags.

Project Risk—A more mature, stable product reduces risk by having a well-understood COTS baseline to work with.

Purchase Price—A less mature product could cost less to induce purchases, but it could cost more because the developer needs to recoup costs.

Development Cost—Lower maturity implies increased component volatility, and changes will take time for developers to learn.

Maintenance Cost—A mature product will refresh less and require less ongoing effort to incorporate new versions.

Quality—The likelihood of undetected defects greatly increases with a less mature product, therefore, maturity will increase delivered quality.

As you can see, the weighting process is quite intuitive; comparative weights are recommended instead of absolute, numerical ones. Weightings can vary significantly depending on the component and project, so we recommend you re-evaluate weights as necessary. Evaluate off-the-shelf options against a custom-developed one, if that is the alternative.

"Micro" Factors

These factors are intended to address detailed integration issues involving off-the-shelf code. A developer could use these to gauge reuse risks and potential costs at a highly detailed (perhaps component) level.

OFF-THE-SHELF PRODUCT CHARACTERISTICS

Component Type—The level of access provided, and required. Ranges from pre-existing code requiring extensive modification to pre-built, encapsulated components with discrete interfaces.

Component Volatility—How mature is the product, how often will it be upgraded, and to what extent? Ranges from an alpha release to a highly mature product.

Component Application Complexity—The difficulty of the application, in terms of the amount of study and experience required to become proficient in using it. Ranges from a simple, general application to one that is very peculiar and difficult to fully understand.

Interface Complexity—The design and coding overhead

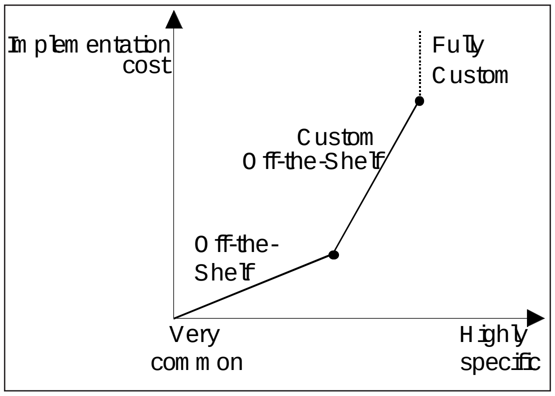

Chart 2[3] *Degrees of application specialization*

Table 2 *Quick-sizing magnitude of off-the-shelf items*

Size of Off-the-Shelf Software	Examples	Extent of Functionality That Is Used		
		Some (< 40%)	Much (30 – 70%)	Most (> 60%)
Small	Minor set of functions that deliver functionality at a subsystem level with no further methodological ramifications; minor application (text editing, etc.) to be incorporated into target system.	Very Small	Small	Small
Medium	Major set of functions that require a "methodology shift"; major subsystem, such as a database engine or graphics rendering.	Small	Medium	Medium
Large	Major library that is fundamental to proper exploitation of the target environment; primary application that will be modified to current requirements, such as desktop software or highly application-specific software.	Medium	Large	Large

required to integrate this component. Ranges from a simple, stable interface to one that incorporates many factors that change with the circumstances of implementation.

Product Support—The degree of access to vendor or effective third-party support. Support may be as good as an onsite consultant or as nonexistent as a disbanded original development team.

INTEGRATION INSTANCE

Component Selection Completion—The extent to which the off-the-shelf component has been identified and evaluated. Ranges from selection completed to no product evaluations done.

Experience With Component—The developers' experience with the component: Is each integration of this a significantly new and complex task, or has it become a standard operation?

Learning Rate—Rate at which people can learn while implementing a component. Ranges from exceptional opportunities for learning-while-doing to a massive task with very little opportunity for learning-while-doing, with each integration being new territory.

Reverse Engineering—The percentage of component functionality that must undergo thorough review by technical staff. Ranges from "black box" to line-by-line review and testing.

Component Integrate and Test—The integration effort required for this component; rank this against what typically is required to integrate a home-grown unit of code into a software program. Ranges from above-average, intimate dependencies that need to be addressed to virtually no effort to be integrated into the target system.

Test Level—The rigor and formality of testing is related to contractual requirements and the potential for loss if the software malfunctions during operation. Ranges from "highest reliability" or "public safety" (rigorous testing, following prescribed plans, procedures, and reporting) to "slight inconvenience" (minimal testing, no prescribed procedures or reporting).

For a risk analysis, these micro-factors can be ranked using the suggested ranges, compiled with a weighting scheme, and overall scores can be compared for differ-

	Notes	Development Schedule	Project Risk	Off-the-shelf Purchase Price	Development & Integration Cost	Maintenance Cost	Quality
Target project/environment							
Openness of architecture	Highly standardized, able to accomodate wealth of pre-built components under stable, well-understood conditions	i	**i**	i	**i**	**i**	h
Flexibility of requirements	An envelope, rather than a rigid benchmark, within which an integrated item's performance and function can lie	**i**	**i**	i	**i**		**h**
Attitude of staff	Progressive staff with a desire to balance what is new and good with a fair degree of caution	i	**i**	i	i		h
Evolutionary emphasis in system requirements	A system architecture and interfaces that are robust enough to accomodate significant product upgrades		**i**	i		**i**	**h**
Off-the-shelf product and provider							
Maturity/stability of product	Proven over time and in the marketplace	i	**i**		**i**	**i**	**h**
Commercial competitiveness of product	A well-developed market with numerous suppliers		**i**	**i**			**h**
Supplier "reputation"	Ability to support and respond over the long term		**i**	h		**i**	**h**
Evolutionary emphasis in product strategy	Version changes are not radical, and are inclusive of the existing customer base		**i**			**i**	**h**
Transparency	Insight into—and ability to modify if necessary—internals of code	**i**	**i**	h	i	**i**	**h**
Refresh Frequency	Pace with which new versions are released, and extent of modification	**h**	**h**		h	**h**	
Functional match	Appropriateness of off-the-shelf item to the current application	**i**	**i**		**i**	**i**	**h**
Performance suitability	Suitability of off-the-shelf item to application's performance requirements	**i**	**i**		**i**	**i**	**h**

Table 3 *Factors for high-level evaluation of reuse plans*

ent components within a project. We also find that component integration is somewhat analogous to other kinds of software reuse, and we have mapped these factors into cost determinants of reuse effort to obtain a rough integration cost model.

Running the Cost-Benefit Analysis

There are two kinds of off-the-shelf vs. custom comparisons, easy ones and hard ones. It is not worth dwelling on the easy comparisons, such as whether to build something like a word processor from scratch or buy one. However, the more difficult choices will need a cost-benefit analysis. Costs and benefits may not always be expressed in dollar amounts; if this is the case, then you must resort to relative comparisons.

Obtain dollar costs for custom development using any suitable method, including an estimating model. Off-the-shelf costs, meanwhile, include purchase and vendor support fees plus actual integration effort; a small number of cost models are available to estimate the latter. The nonmonetary costs of integration have been covered with the micro- and meta-factors presented here. Make certain to apply these factors to both off-the-shelf and custom development options.

To a great extent, software benefits are accounted for in terms of functionality delivered. Off-the-shelf software may offer a substantial solution but not a complete one, at least not what a custom product could deliver. User flexibility therefore goes a long way towards certifying that something not home-grown is good enough. For example, can performance requirements be relaxed? If no single "silver bullet" exists then a fine-grained amalgam of solutions may approach a full solution for the target application.

Make certain that benefit evaluation extends into the hidden realms beyond mere functionality delivered. An off-the-shelf option may offer better vendor support, less expensive service, and plentiful expansion options. Other advantages can be harder to quantify, such as ease of use and a community of like- users. Significant hidden benefits should be carried through into final comparisons.

You are going to have to mix dollar comparisons with less tangible costs and benefits. Zero in on the factors driving your situation and let these guide your analysis, keeping the number of compared items to a manageable amount. This sort of multivariate comparison process is an ideal candidate for a method known as the analytic hierarchy process,[5] which basically provides a formal framework for the mixed comparison of many unrelated factors.

Summary

In planning off-the-shelf integration, some useful general management advice may be available, but it is really up to your careful research, preparation, and skills to make any effort a success. The next time you need to integrate off-the-shelf technology, the best tactic is to ask the supplier for a manual. If there is no manual, start worrying.◆

About the Authors

Lee Fischman is special projects manager at Galorath Inc. in El Segundo, Calif. He conducts research and development of SEER tools and consulting methods, and has explored software economics and estimating in numerous papers over the past several years.

Karen McRitchie is vice president of development at Galorath Inc. She has more than 10 years of experience in software and hardware cost estimating and reliability modeling. She has been a lead member of cost estimating teams on many major projects.

Galorath Incorporated, The SEER Product Developers
100 North Sepulveda, Suite 1801
El Segundo, Calif. 90245
Voice: 310-414-3222
Fax: 310-414-3220
http://www.galorath.com
Fischman's e-mail: fischman@galorath.com
McRitchie's e-mail: karenm@galorath.com

A list of related writings is provided at http://www.galorath.com/COTS_references.html

Notes

1. ***COTS is dead, long live COTS!*** From being an infrequent shortcut to the traditional build everything development emphasis, COTS has become pervasive in today's software development projects. The term is generic and can refer to many types of applications, levels of complexity and integration. Meanwhile, a host of other off-the-shelf solutions have matured, which, although not truly commercial, offer similar benefits. The COTS shorthand has become a kind of mantra, yet the more general challenge of using off-the-shelf software provides a much more complete lesson. We therefore define off-the-shelf software as any cogent code that is being reused.
2. Adequate vendor support frequently must be obtained through upgrades, as support is removed from older versions of software.
3. Chart 2 shows the relationship between specificity of requirements and implementation cost as a function of customization. As requirements grow more specialized, the cost of using off-the-shelf software increases. Past a certain point, off-the-shelf software must be mixed with custom work to fulfill specific requirements. The most specific needs (target recognition, radar cross-section calculation, etc.) may require fully custom solutions.
4. The emphasis in these factors was derived primarily from a literature survey. It is very hard to gather actual data; we are therefore interested in hearing from you. The chart has been posted on our web site at www.galorath.com/COTS_survey.htm
   You can suggest alterations to the factors, including alternate weightings, and your rationale for any changes you suggest. We will send the results of the survey to everyone who participates or simply registers.
5. A great deal of information on this method is available at www.expertchoice.com (no affiliation with Galorath Inc.).