Impact of Source Code Availability on the Economics of Using Third Party Components

A White Paper

Desaware Inc.1100 E. Hamilton Ave #4, Campbell, CA 95008 (408) 377-4770

The Economics of Third Party Components

The decision of whether to buy or build a component is fundamentally an economic choice. However, the economics of this choice has changed with the transition to .NET. This paper will analyze the economics involved, the nature of this transition, and the impact of source code availability on the buy vs. build decision.

Economic factors influencing the buy vs. build choice

There are many factors to consider when deciding whether to use third party components in a project. It is essential to consider and try to quantify each of these for your own particular situation before making the build vs. buy choice. Only by doing so can you truly compare the total cost of ownership of each approach.

Developer costs

This is the cost of developer time to implement a certain set of functionality. For components you have the cost to evaluate the component and integrate it into an application. Generally speaking these costs will be much smaller than the cost to implement the functionality on your own.
Note: A major factor in the buy vs. build decision is often psychological rather than economic, where developers want to build a features simply because it is more fun. Without a good economic analysis, this choice can raise costs to the point where a project fails completely.

Feature costs

When you build your own component, you can build it to implement the exact feature set you require. With third party components the component may lack a feature that leads to a reduction in the functionality of your application. This reduction in functionality may result in costs in the form of reduced sales, or competitive disadvantage.
On the other side, a component will often implement features you do not use. While these features have no direct costs, they do increase the overall cost of deploying the component because they demand additional time to learn[1] and add complexity to the component[2].

Time to market costs

Deploying a third party component is dramatically faster than building your own. You should estimate the amount of time it would cost to develop the feature on your own[3] and estimate the impact of the delay on the overall profitability of the project. In many cases this will turn out to be a major factor in the decision.

Short term support and risk costs

When purchasing a third party component, you benefit from the testing and QA process of the component vendor. It is important to choose a vendor with a reputation for quality components and support, because unsupported software can incur significant costs. For example: the cost of first deploying a solution with a component, then going back and reimplementing the feature on

---

[1] Though it may not be obvious, features you do not use also incur educational costs. They lengthen the time it takes to determine what features you do need to use, and a developer typically has to be familiar with those features before choosing not to use them. An extreme example of this is the .NET framework, where the cost of just becoming familiar with the huge feature set is a major factor in the cost of migrating to .NET.
[2] Additional complexity usually incurs additional costs in the form of greater support costs and higher long term risk (it is harder to maintain backward compatibility in later versions of a component when it is more complex).
[3] Try to be as realistic as possible. Software schedules are notorious for going over initial estimates.

your own is higher than implementing the feature on your own from the beginning. However, you can also look at it this way: Trying one or even two third party components to see if they will work before implementing your own solution potentially increases your costs by some small percentage, while giving you the chance to reduce costs considerably.
For example: Let's say you have three possible choices:
Component A – Total cost $2000[4] - 2 weeks to deploy
Component B – Total cost $4000 – 2 weeks to deploy
Build custom – Total cost $10000[5] - 2 months to develop and deploy
Cost of time - $1000/week – This represents lost revenue or savings due to delays.
Your cost of going directly to a custom component under this scenario is $18,000
Your worst case cost is $28,000 if you try both component A and B and they fail.
However, by trying component A you have the opportunity of reducing your costs to just $4,000 – a dramatic savings.
This are, of course, hypothetical numbers. In practice the costs of developing your own components and the costs of delays are likely to be significantly greater, and through proper evaluation you can increase the chance of a component working.
In terms of pure support costs, most component vendors provide free support for a limited time – plenty of time for the initial deployment and evaluation. Supports costs after that time are still typically less or comparable to what it would cost to have your own developer support a component.
A study by ComponentSource titled "The Return on Investment on Commercial off-the-shelf (COTS) Software Components" shows return on investment starting at 200:1 or higher. Their model, which focuses on development costs, does not include many of the factors discussed here (including risk factors and longer term costs), however it is substantially correct – the actual cost savings of using commercial components is typically far greater than the numbers described in this hypothetical example.

Long term support and risk costs

It is important to factor in the long term support costs when considering the buy vs. build decision. This is not just a case of knowing upgrades will be available. The big question is – will the component still be available in the long term should the application need to be rebuilt, perhaps in order to work on a newer operating system. Will upgrades be available for that system? Will the company that provided the component still be in business.
Some use this concern as an argument for building your own components, but in practice this is not a compelling argument. While there is a chance the component vendor will be gone (or no longer support the component) even if you had built it yourself, chances are high the original developer will long since be gone. To reduce the risk, you should make sure the vendor either provides source, or a source code escrow option (in which the source code becomes available in case the vendor goes out of business).

Product Costs

The actual cost of a third party component is typically an insignificant factor in the total cost of ownership of a component.

The Economics of COM Components

In order to understand the impact of .NET on the buy vs. build decision, let us begin by evaluating the relative weight of the economic factors for COM components.
The component market for COM components is effectively divided into the Visual Basic 6 market, and all other markets. Of these, the VB6 market is the larger one.

[4] Includes cost of the product and developer time to integrate the component.
[5] This is a very conservative number given the cost of developer time.

This is because with Visual Basic 6, the dominant factors are developer and feature costs. Visual Basic 6, for all its power, is simply not suitable for many types of components. Complex user interface components, certain kinds of multithreaded components and components that interact with the system can not be written in Visual Basic with any degree of reliability of acceptable performance (if at all). As an example of this: Desaware's Spyworks consists of components that allow VB6 programs to push the limits of what is possible in VB6 – but the SpyWorks components themselves are written in C++.

This is important because historically VB6 developers are much less expensive than C++ developers. The need to use expensive C++ developers for certain types of components is one of the major factors in the success of the COM component market for VB6.

The fact that VB6 programmers simply could not create certain types of components reduced the psychological drive towards building components – even if building a component might be "fun", it was effectively impossible.

Third party components were not as successful in the C++ market. The desire of many C++ developers to "write it themselves" is one of the major reasons for this.

Unfortunately, many VB6 developers have experienced problems with the long term support of components they've used. Developers supporting earlier legacy applications have had trouble supporting and upgrading their products as some of the early vendors have gone out of business, and others have ended sales and support of earlier products[6]. For the record, Desaware still sells legacy VBX products for those who need them.

These problems have led some developers to overreact against the use of third party components. Unfortunately, without a sound financial analysis, this too can lead to an expensive choice.

The Economics of .NET Components

The influencing factors in the buy vs. build choices are still the same with .NET as with COM, but the weighting is very different.

Unlike the case with VB6 and C++, VB .NET and C# developers are on an even footing in terms of the types of software that can be created. The requirement to use third party components because it was impossible to implement functionality in VB is reduced. This means, for the first time, VB .NET developers have the same psychological factors as the old C++ developers – it is more fun to build a feature yourself, even if it is not the best economic choice.

The .NET platform, while it does have a significant learning curve, ultimately reduces the short term costs to build and deploy a component. The increased functionality of the framework (as compared to COM technologies) lends developers to correctly search first for solutions within the framework before considering either third party components are development of their own components. In many cases an adequate solution can be found within the framework. In those cases the economic question becomes primarily based on features: "Will the increased features provided by a third party component result in revenue/savings that justify the cost?". In other words, the choice is no longer between third party components and developing your own component. It's between third party components, developing your own component, or using existing functionality within the .NET framework that may not be perfect, but may be good enough.

The long term support costs and risks for .NET components is similar to those of COM components, but have an added complexity. Because of the ability of the different versions of the .NET runtime to run "side-by-side", .NET poses a more complex versioning story that the COM world, where a new version of a component replaces all previous versions. Microsoft does not guarantee backward compatibility of newer runtime versions, meaning you may need to upgrade all of your components in order to migrate your application to a newer version of the runtime.

---

[6] This is not just a problem with component vendors. Many VB6 developers today are seriously concerned about Microsoft's plans for VB6 itself – given the huge cost to port to .NET, Microsoft may prove to be the source of the greatest long term risk of all.

The Economics of Source Code Availability

Increasing numbers of customers have been calling on vendors to provide source code. In this section we will explore the economic issues of source code availability.

Justification and Cost of Source Code Availability

There are a number of arguments given for having vendors make source code available.

Guarantee of long term support

Having been "burned" by long term support issues in the past, some developers look to source code availability as their way of guaranteeing that they will be able to support today's applications for the long term.

It is important to understand that the primary motivation here is not cost savings (at least as compared to developing your own code). It is safe to assume that the people who developed your code will not be those providing the long term support – so whether they have to learn to support code you created, or source code obtained for a third party component is largely irrelevant – both are big jobs.

The primary motivation here is the ability to keep a legacy product working at any cost. Source code provides the ultimate backup in cases where the original component is no longer available, or the original documentation or design time tools have been lost.

In these cases, source availability is primarily a form of insurance. Escrow services can provide similar insurance, however escrow services have not achieved a great deal of popularity, especially for small to medium size companies.

Ability to add new features and fix bugs

The third party component business model is fundamentally based on amortizing costs across many units – that is where the cost savings occurs. In most cases this means that customization for individual customers is not possible – at least not without charging significant consulting fees that can reduce the benefits of choosing the third party component in the first place.

The same economics applies unfortunately to bug fixes. A reputable vender will make every effort to fix a bug even if it is only experienced by a single customer, however no software is completely bug free, and at some point the economic pressure to ship the product or live with the bug become extreme[7].

Source code availability makes it possible, if the license permits, for you to modify or customize a component to suit your needs. Note that this is not a trivial task, especially for COM components. First, there is obviously the need to learn the source code – not as expensive a task as developing your own component from scratch, but not trivial either.

In the case of COM components, you must also be careful to rename the component and replace existing GUID and IID values. This is necessary in order to make sure that later versions of your component do not conflict with the one the vendor is shipping. With .NET components, the process of modifying the component is easier (just change the root namespace), and the component can be shipped in the application's directory, thus eliminating the potential for conflict between the components.

In practice, few customers will actually build their own version of a component.

Ability to understand better how the component works

Components suffer from a problem that is common to all software – it is poorly documented compared to other technologies. If you were, for example, to compare the documentation for a microprocessor to that of software of comparable complexity, you will inevitably find the CPU

[7] Microsoft's own products are classic examples. Most do ship with known bugs, if only because if they didn't, they would never ship a product. Component vendors actually tend to be more responsive than the larger companies in terms of providing rapid bug fixes.

documentation superior in every way. This is because the costs to develop a CPU are so high, and the consequences of error so great, that an intense design effort is required to both control costs, and ensure the highest possible quality. These detailed design documents can translate easily into highly detailed documentation and specifications.
Software is relatively inexpensive to develop, thus while software benefits greatly from a strong design effort, it is possible to develop working software with remarkably little design work. And few developers create the rigorous product definitions common in the hardware world. As a result, there are no detailed design specifications on which to base documentation, and software tends to be poorly characterized.
Developers using components (whether third party or those developed in-house), often have to rely on trial and error to figure out subtle behaviors of components, especially once deployed in real world scenarios.
Having access to source code can help developers to understand what a component is actually doing when problems occur. It can also help developers to understand how to make the best use of features, especially those that are poorly documented.
This has proven to be a great asset for users of the .NET framework, where source code is available both as C# code (for some portions of the framework) and as disassembled IL code (for the entire framework).

Consequences of Source Code Availability

The arguments for having source code available are compelling from the perspective of a customer. However, it is important to realize that there are consequences and indirect costs of having source code available that are especially significant in the long term.

Costs of Piracy

Software piracy is prevalent and costly to third party component vendors, and the risks of piracy are greater when source is distributed. Obviously, this reduces a vendor's revenue which both reduces the funds available to develop new components, and in extreme cases can jeopardize the survival of the company. This in turn increases the risk of losing long term support (a risk which is partly mitigated by the availability of source code in the first place)

Impact on Pricing

Source code availability inevitably eliminates any competitive advantage gained from the code itself. If one company has a great idea on how to solve a particular problem, and they release the code, obviously it becomes possible for other companies to implement the same idea. As a result, proprietary technology can no longer support higher prices – software needs to be priced low enough to make it easy for customers to remain honest, and to reduce incentive for competition to come in. Third party vendors who release source must look for other competitive advantages.

Impacts on Business Model

The traditional component business model did not include source code distribution but typically included low cost or free support. Because a source distribution model dictates lower pricing, the costs of providing support quickly becomes prohibitive. In fact, companies that distribute source typically end up moving more towards the open source business model, where value added comes in the form of ongoing support and services.
As a result, one of the major consequences of the move to source code availability is an increase in support costs. In fact, support costs are not actually increasing – they are just being shifted from the cost of the product the individuals who actually use support services.
Generally speaking, companies that distribute source code do not provide any support for versions of the component build by the customer (i.e. their own custom versions based on modified code). Those that do inevitably charge high consulting rates to look at customer modified code. This is another reason why few people build their own versions of components.

Conclusion

The arguments in favor of obtaining source code with third party components are compelling for software developers. It allows them to take advantage of the dramatic cost savings made possible by components, while at the same time reducing the long term risk of using someone else's code. It also makes components easier to use and support.

Source code primarily serves as a form of insurance for customers, and secondarily as a source of information that makes the components easier to use and deploy. It is relatively uncommon for customers to release their own versions of components, even in cases where a license permits them to do so.

By reducing the risk of using a third party component, distributing source code allows customers to enjoy the significant cost reductions that are made possible by using these components. These costs reductions go far beyond mere savings, in that they also result in significantly faster time to market, which can result in a major competitive advantage for those companies who use these components.