

# Rewriting existing software – The coming wave

What are the forces that are making software rewrites attractive and what can be done to kick-start the process within your company?

Ravindran Kittu  
Aditi Technologies Private Limited  
2002 156<sup>th</sup> Ave NE  
Bellevue, WA 98007

Phone : 425-378-6500  
Fax : 425-653-3031  
Email : [sales@aditi.com](mailto:sales@aditi.com)  
Web : [www.aditi.com](http://www.aditi.com)

**A D I T I**®

Many existing software applications are coming up for a rewrite – to overhaul dated architectures and to better capitalize on new market opportunities. While rewriting is often an appealing option, many software companies lack the bandwidth and sometimes the technical skills to move to a new architecture.

What are the forces that are making rewrites attractive and what can be done to kick-start the process within your company?

## The Setting

The software industry has long recovered from the dot-com blues and is regaining strength. M&A activity is increasing; venture capital in software is holding steady; and more companies are turning a profit.

But something is wrong under the hood. Many of today's software applications are behind the times – while they strive to keep up with user needs, they have dated architectures that are unable to support radical functionality changes. Software companies that survived the dotcom crash had adopted a cautious approach, limiting R&D spend to functionality upgrades without major changes to underlying architecture.

Now, for many of them, a complete rewrite is a cleaner alternative to evolutionary change.

There are two forces driving this transformation. New opportunities that make rewrites attractive and ageing code bases that are getting difficult to maintain.

## New Opportunities

Rewriting software is not merely an attempt to create the same functionality using newer platforms or technologies; it is an opportunity to rethink user benefits and to fundamentally redesign software to accommodate those needs.

### **New wave of technologies**

After a long gap, major platform vendors are launching technologies that application vendors need to adapt to. Microsoft's last major release was in 2001, but it is now almost simultaneously launching new versions of major products – a new OS (Vista), a new development environment (VSTS), a new database version (SQL 2005), and more.

Such new platform tools represent new opportunities for application developers:

- First, an opportunity to replace custom code with out-of-the-box functionality. For instance, while many companies had built their own portal and repository functionality for their products, they can now simply integrate SharePoint Portal Server, which offers 60% of such functionality out-of-the-box. By retooling their software to take advantage of such advances, application developers can better focus on enhancing core business logic rather than building infrastructure.
- Second, new tools offer potential for improved functionality or user experience. Examples of such improvements could include enhanced UI, support for multiple devices, and desktop over the web functionality.

### **New paradigms**

Client-server applications gave way to web-enabled applications. Now simple web-enabled applications are being overtaken by more sophisticated concepts.

#### *Software as a Service*

Software as a Service (SaaS) and hosted enterprise applications have long been talked about, but their time may have finally come. Gartner predicts a

third of existing enterprise applications to move to a hosted model and 40% of new applications to be built using it. The model has obvious benefits. It makes life easier for users, with lesser investment and faster time to deploy. And it makes life easier for software vendors, with no distribution headaches and no waiting to bunch features into releases.

The promise is not new. But what is new is that the enablers are in place to make this model mainstream. The only element lacking is for software companies to redesign their software to support a multi-tenant hosted model.

### *Web2.0*

Software products being Web enabled is the norm today, but the Web is already a changed place. Microcontent objects, Tagging, Wikis, Open social networks – welcome Web 2.0. Companies like Salesforce.com and Amazon have already integrated such new concepts into their software. While it will take time for Web 2.0 concepts to become mainstream, companies that move today are the ones that will hold the competitive edge tomorrow.

### **New user needs**

#### *User environments have changed*

And this has resulted in new user demands. As an example, consider how ubiquitous mobile devices have become and the impact this has on user demands. Users now expect their software – from email, to line of business applications – to follow them on their devices. And software makers consequently have to adapt their software to multiple devices and protocols.

#### *New business processes*

Enterprises in many industries, such as Healthcare, have adopted new business processes to comply with new regulations. While software companies that serve them have been incorporating changes into existing products, such changes have tended to be unwieldy workarounds, leaving users less than happy.

#### *New markets*

Companies frequently target new market segments – enterprise software companies target SME customers, or vice versa; and single platform companies target customers on a second platform – and such initiatives often necessitate changes to the architecture.

## **Problems with existing code-base**

While the above opportunities pull companies towards rewriting their software, the problems of maintaining an ageing code base is pushing companies towards change.

### **Spaghetti architecture**

Most established software companies have products that have been developed over many years, with their architecture a patchwork of independently evolved subsystems developed using different platforms and languages. Apart from being difficult to maintain, such architectures make it difficult to incorporate new functionality. Witness the widening gap between the long list of functionality demanded by product managers and the time it takes to actually develop these features on existing architectures.

### **Poorly maintained code**

Code that has evolved over a long time tends to be in poor shape, making maintenance a nightmare. People who wrote the old code have long left and there is little documentation. It is difficult to touch something without breaking something else, and

quite understandably, even the best engineers do not want to touch certain parts of the code.

### High R&D spend

Products developed using old technologies extract a high ongoing R&D cost. They would not easily integrate with mainstream third party tools, often forcing companies to write their own development and reporting tools, and making it difficult for them to integrate with automated test tools. In addition, finding engineering talent familiar with the older technology becomes difficult.

## Strategy

Clearly then, remapping customer needs and rewriting software is often much cleaner than trying major surgery to old architecture. The investment makes products *future-proof* and brings down cost in the long run.

### So what is the problem?

Even when software executives see the need, making the transition is difficult for two reasons.

- *No bandwidth*: Conceptualizing a new architecture takes time and bandwidth but software companies generally have neither. Senior executives are occupied in battling everyday market pressures, and engineering resources are tied up in running the treadmill of releasing new features. Re-architecting takes time and the existing version needs to be in the market till the new version is ready; but tasking R&D teams with maintaining the existing product as well as creating a new version is an overload.
- *Inadequate skills*: While software companies are usually expert in their current technology platforms, they tend to be unfamiliar with the new platforms they can leverage. Without an intimate understanding of possible destinations, it is difficult to conceptualize new architectures or develop them.

### What then are the options?

Large software companies set up separate in-house task forces to drive the change. Such teams are often formed by pulling in stars from existing product teams and supplementing them with new hires who are expert in the destination platform.

This is often not a viable option for small and medium software companies that already have stretched R&D teams. A good strategy for such companies is to consider a phased re-architecture with support from an external technology partner.

- *Phased rewrite*: A complete one-shot rewrite is a daunting task and not the best approach, except in rare cases where the existing version is beyond redemption, or where the company wants to move swiftly to tap a new market opportunity. An alternate approach is to phase the rewrite over time, replacing components or layers one at a time, allowing time for stabilization.
- *Technology partner*: Bringing in a technology partner is a tested way to ease your resource constraint. A good partner can bring expertise in a variety of new technologies, specialist design skills, and the right processes to drive software rewrites. The catch: No partner would come in knowing enough about your market or users. Here is one approach to make partnering work: Start by outsourcing a piece of product maintenance to the partner. It would let them understand your product; cut your maintenance cost by half if they use offshore resources; and free up your star resources to think about the future. You can then let your partner shoulder additional design or development responsibilities while you build in-house skills on the new platform.

## In Summary

A software rewrite is certainly not a simple exercise. It's a long road, but the sooner you start, the stronger your foundation for the future. And there is at least one thing about a rewrite that you can smile about: nothing can get your engineering teams more excited.