mrc

Crash Course in Modernization

A whitepaper from mrc

Introduction

Modernization is a confusing subject for one main reason: It isn't the same across the board. Different vendors sell different forms of modernization. Each claims their way is the best way. Each promises a result that looks modern. Be warned, not all modernization methods are equal.

Allow me to share with you the most important rule of modernization. This is what many modernization vendors hope you don't know: **There is much more to modernization than an end result that looks modern.**

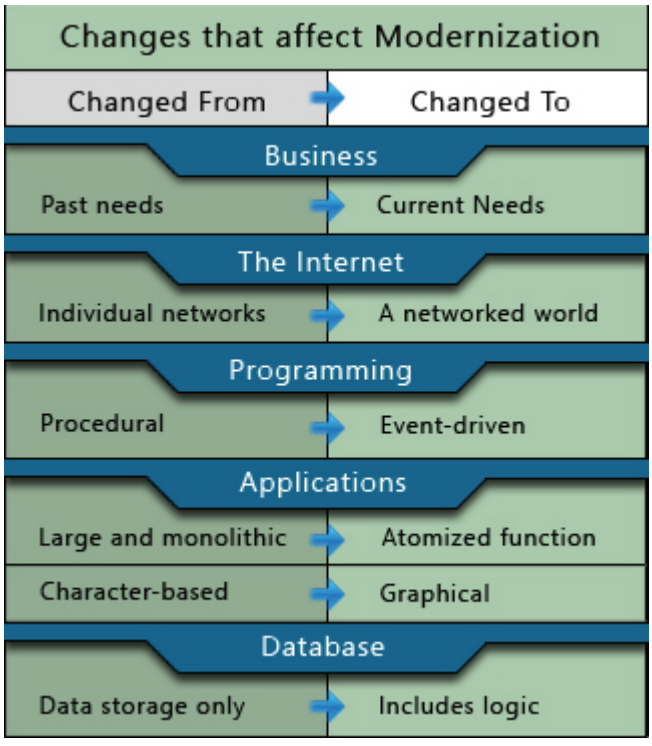
This whitepaper will give you a crash course on modernization. It will help you understand what must happen in the modernization process and explain your modernization options.

In the first part of this whitepaper, you will learn about common areas of change that potentially impact modernization. The second part of this whitepaper outlines the many modernization options and explains the advantages and disadvantages of each.

First, let's examine what has changed to cause the need for application modernization.

What has changed

Understanding what has changed since your original applications were created is the first step towards successful modernization. This chart provides a brief overview of the common changes we will examine in this whitepaper:



The business has changed

Make a list of all the ways your business has changed since the creation of your original applications. Here are a few questions to consider. You will think of more. Has the government passed any laws that affect your

business? Have taxes changed? Have your products or customers changed? Has your business grown? Have your selling methods changed?

Once you've made your list, take a moment and review it. If your company is like most, the list is probably longer than you thought. Now, ask yourself this question: Do my old applications address all of my current business needs? Again, if your company is like most, the answer is "no."

Keep this in mind because it's important: **If your business has changed to the point where your old applications no longer address your business needs, those applications must change.** Now, some vendors will tell you that you can put a modern-looking face on top of those old applications and fix your problems. That is false. A modern face doesn't magically make your old applications address your present business needs. If you want to address your current business needs, the underlying applications must change.

How exactly should your old applications change? It all depends on when they were created. For example, if they are over 15 years old, they were built for a world that didn't rely on the internet. If they were designed for use before the internet became widely used, they require major changes. Here's why:

The internet has changed

Over the last 20 years, we've moved from individual company networks, to a network that covers the world (the web). The rise of the web forever altered the business landscape. For example, the web broadened most companies' potential customer base. It changed the way many companies sell to their customers, communicate with their inside and outside sales forces, and manage their distribution channels. It reduced the need for companies to operate within a physical location.

Applications created before the web became widely used in the business world are fundamentally different than those created after. They weren't built to communicate with other applications. They weren't built for a connected world. They were largely character-based and often required some training before use.

How have current applications changed? Current applications are built for a networked world. They communicate via the web and with other applications. They include graphical user interfaces and are designed with usability in mind.

How do these changes affect modernization? Applications created before the web was widely used were built for a different era. They were built to perform different functions. To modernize these types of applications for a web-based world, two things must change: First, they must change from a character-based to a graphical user interface. Second, the entire application structure must change.

Why do pre-internet applications need a different structure? When the internet became widely used in the business world, two things happened. First, programming paradigms shifted to address the changing needs brought on by the web. Second, the entire application structure changed to better fit the requirements and opportunities of the web.

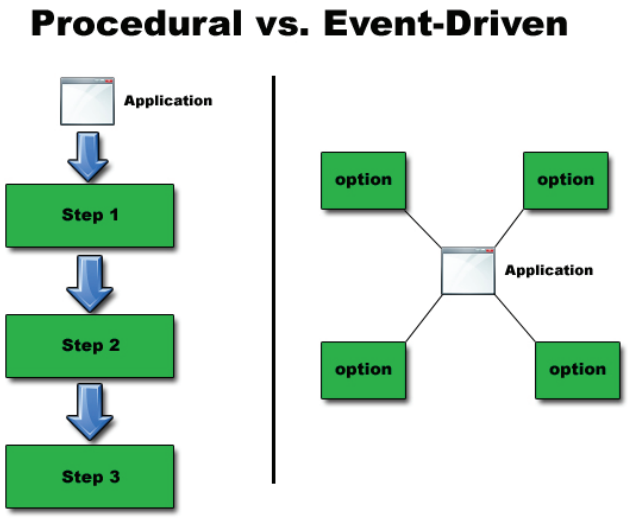
Now, some vendors will tell you that you can place a graphical user interface on top of your old applications and call them "modern." This fails to address two important problems. First, it doesn't address the business changes. Second, it doesn't address the application structure and changing programming methods. In other words, it leaves you with a modern-looking application that doesn't account for changes to your business and wasn't designed for a web world.

Why is the changing programming paradigm and application structure so important? Keep reading…

Programming paradigms have changed

Applications built before the web became widely used were often created using a procedurally-based programming paradigm. Without diving too deep into the technical aspects of programming, procedural programming provides a series of step-by-step instructions for the application to carry out. The application follows these instructions and cannot deviate from the pre-determined path. This method worked well in the past because these applications didn't live on the web and were only used by trained employees.

This changed when the internet became popular. Companies realized that their old applications didn't translate to the web. They were character-based. They were ugly and difficult to use. As a result, programming paradigms changed, and event-driven programming was born. Event-driven programming is designed for use on the web, because it provides the user with choices. Rather than pre-determined steps, the application is guided by user-generated events, such as mouse clicks. The image below illustrates the differences between procedural and event-driven paradigms. You'll notice a pre-determined path in the procedural application, while the user determines the path in the event-driven application.



When you modernize, you must design your applications for use on the web, which means they must take an event-driven approach. Any modernization method that leaves you with procedural applications isn't really modernization.

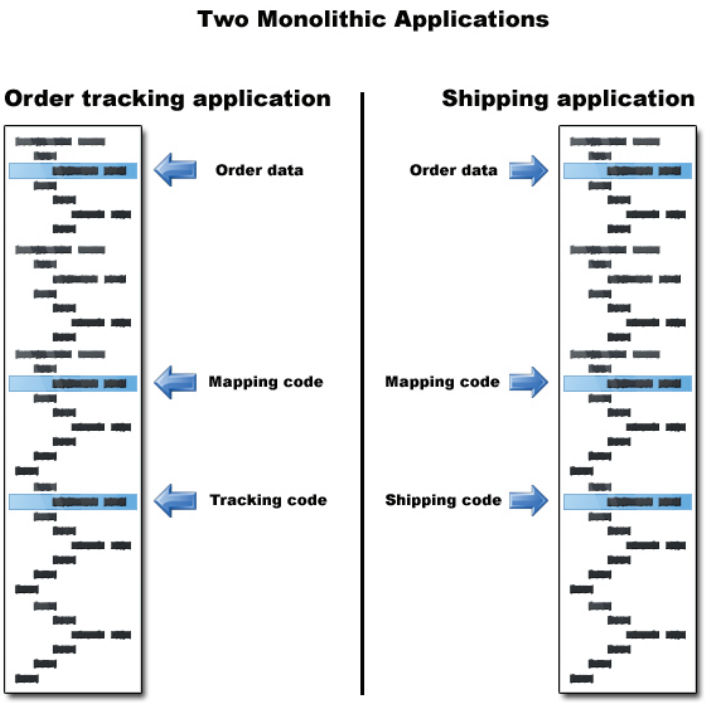
Now, some vendors will tell you that you can keep your old procedural applications and add events to them. Technically, it's true. With enough work, you can make a procedurally-based application appear event-driven (although it is still procedurally-based). But that's like saying you can add fog lights to your car with a pair of flashlights and some duct tape. It might do the trick in the short term, but you're just patching up outdated technology. It doesn't change the programming paradigm. It also doesn't change another critical aspect of the application that has changed since the web rose to prominence: The application structure.

Application Structure has changed

As programming paradigms adapted to a changing world, application structure changed as well. Over time, applications moved from a monolithic structure to an atomization of function. The old procedural programming methods resulted in very large (monolithic) applications, containing thousands of lines of code. Wikipedia provides a good definition of monolithic applications:

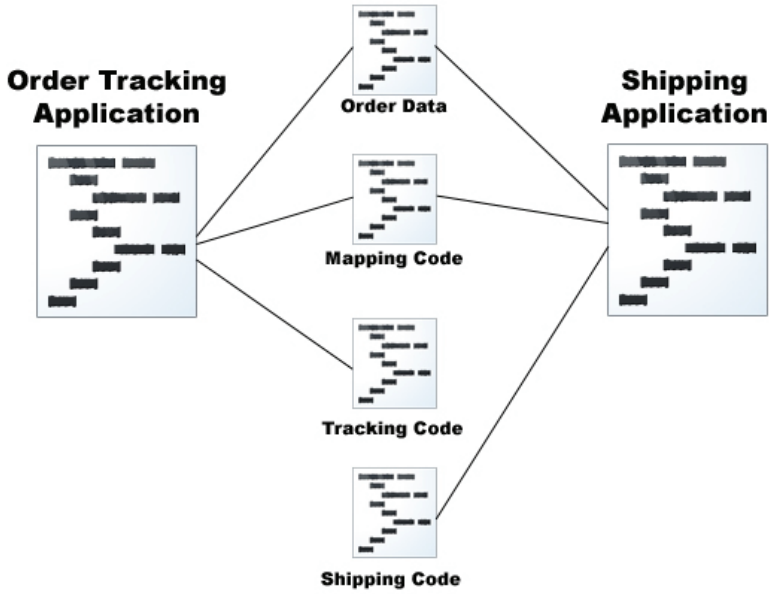
"A monolithic application is self contained, and independent from other computing applications. The design philosophy is that the application is responsible not just for a particular task, but can perform every step needed to complete a particular function."

These applications were so large because every function of the application was programmed into the code itself. That means if you had different applications sharing common functions, each function was built into each application separately. They couldn't share code with other applications. This image illustrates the monolithic application structure.



Because of their large size and inability to share code, monolithic applications were difficult to create and maintain. As a result, application structure moved to an atomization of function. These days, applications use "building blocks" of code (also called "objects") which work together as a larger system of applications. Each object performs a specific task and can communicate with other objects to create larger applications. The image below illustrates how the two previously mentioned applications would be built using modern methods:

Two modern applications (Atomization of function)



As you can see, atomizing function is a far more intelligent design structure. Applications built in this way are easier to build and maintain, and work well on the web. Here are a few differences between the two structures:

1. **Maintenance:** Maintaining a monolithic application is difficult, whereas maintaining applications that atomize function are far easier. Here are three of the biggest issues that contribute to this difference in maintenance difficulty:

    a. **The workforce has changed:** Because these applications were created so long ago (often 20-30 years), the original developers are usually no longer with the company. Additionally, they were created with outdated languages (COBOL, Pascal, Fortran, etc…) which are not taught much in colleges these days. These applications are usually maintained by those who don't understand the language and didn't build the initial application.

    b. **Too many dependencies:** Monolithic applications were built with too many dependencies between procedures. This meant that a change to one area of the application could alter, or even break, another area. On the flip side, using the atomization of function approach, a change to one object does not affect other objects.

    c. **No modularity:** Monolithic applications were built without modularity, which is defined by Wikipedia below:

    "Modularity is desirable, in general, as it supports reuse of parts of the application logic and also facilitates maintenance by allowing repair or replacement of parts of the application without requiring wholesale replacement."

    In other words, applications designed without modularity can't re-use or share other parts of application logic. For example, if your product pricing algorithm changes, every application

containing the old pricing algorithm must be changed. Comparatively, the atomization of function approach relies on code sharing. When a change is needed, only one block of code needs to be altered.

d. **Spaghetti code:** As more changes and updates are added to these monolithic applications, the code becomes more and more convoluted. After a while, the application code turns into what is commonly known as "spaghetti code", a name given to twisted and tangled code. The more updates and fixes that are applied to monolithic applications, the more difficult they are to maintain.

2. **Developing monolithic applications is not efficient:** As you can probably imagine, developing these large applications is not very efficient. Developing applications that share and re-use code is a far more efficient method for development. Would you rather hand-code thousands of lines of code, or call pre-made functions?

What does the change in application structure mean to modernization? If you want modern applications that are easy to maintain and address your changing business needs, the underlying application structure must change to employ an atomization of function.

Now, some vendors will tell you that you can update your old monolithic applications with web capabilities. You can even lay a graphical interface on top of your old applications and make them appear modern. Be warned, while this approach will give you modern-looking applications, it leads to problems. First, you still must maintain and support your old monolithic applications, which as mentioned above, is no easy task. Second, now you also must maintain the graphical interface. Essentially, this approach provides modern-looking applications while significantly increasing maintenance complexity.

Additionally, the changing application structure alters another aspect of your business applications: The database. The next section addresses the changing database and what it means for your applications.

The database has changed

In the past, databases only stored data and applications stored logic. Thanks to database and programming advancement, current databases do more than simple data storage. These days, much of the logic previously included in the applications can be moved down to the database level. This results in smaller, more maintainable application code.

For example, when a customer placed an order in the past, the application instructed the database to decrease inventory. Now, that can happen automatically in the database.

Those who still use outdated applications cannot take advantage of new database enhancements, such as triggers, constraints, user defined functions (UDFs), etc... They are confined to their old database until they modernize their applications.

Summary

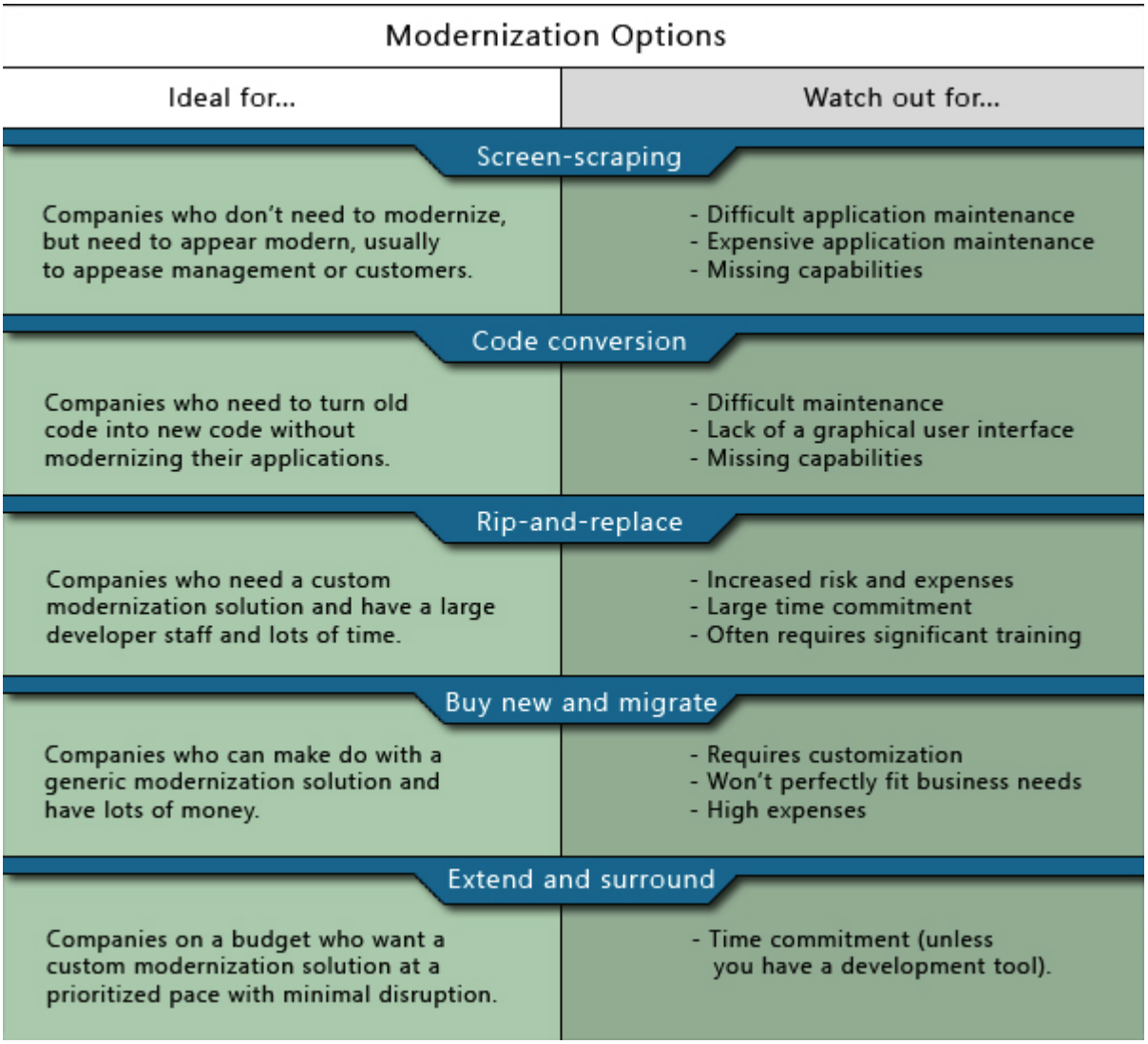
Much has changed since your original applications were created. First and foremost, if your company is like most, your business needs have changed. If your underlying applications no longer address your current business needs, they must change.

Secondly, when the web became widely used in the business, programming paradigms and application structure changed to address the new requirements of the web. As a result, if your applications were created before the web became widely used, they must be replaced. The old programming methods and applications structures don't translate over to a web world. Companies who refuse to modernize these applications are left with maintenance difficulties and applications that can't take advantage of the web.

Now that you have a better understanding of what's changed, let's look at 5 different modernization options, as well as the pros and cons of each.

Modernization Options

Figuring out which modernization method provides the most benefit to your company is difficult. Every vendor will tell you their method is best. Before you even begin looking, it's important that you understand your options. The next part of this whitepaper explains the five methods available as well as a few pros and cons of each. The image below summarizes each method:



Screen-scraping

Screen-scraping creates new, modern-looking interfaces for your legacy applications. It makes old applications look new without actually updating the old applications. A screen-scraping tool does nothing more than scrape the information off your existing applications and present it in a graphical interface.

Pros

Vendors sell this modernization method on speed and price. It is usually the cheapest option and provides modern-looking applications quickly without altering the underlying applications. Additionally, with enough work, modern features can be integrated into the screen-scraped applications.

Cons

There are quite a few drawbacks to the screen-scraping approach. First, it does not actually modernize your applications. While it leaves you with a modern-looking application, it does not address the issues associated with legacy applications that we've just covered. Specifically, you're left with old monolithic, procedurally-based applications which lack capabilities and aren't designed for a web world. In the end, maintenance is actually more difficult, time consuming, and expensive as you will have to maintain your old underlying applications as well as the screen-scraped applications. On top of that, every time the underlying applications change, the screen-scraped application must be remade. To learn about all the drawbacks to screen-scraping, read this whitepaper entitled, "What screen-scraping vendors don't want you to know."

Code Converters

Code converters take old applications written in outdated programming languages and convert the code to a modern programming language. For example, if you have applications written in COBOL, you can run them through a code converter and turn them into Java.

Pros

Code converters address the problem of maintaining an old programming language. They provide a quick way to turn outdated code into modern code without learning a new programming language. There is usually more developer support for new languages, and you won't need to rely on a dwindling talent pool to maintain your old applications.

Cons

Code conversion doesn't fix the underlying problems with legacy applications. It converts the code to a modern language, but it leaves you with procedurally-based monolithic applications. All the maintenance issues associated with monolithic applications will still apply. All the problems associated with procedural-based applications will still apply. Code converters don't magically turn your old code into object-oriented, event-driven applications. Worse yet, it doesn't give you a graphical user interface, which is a must for the web world. They leave you in the exact same place you started, only with applications built in a different programming language. To properly modernize, you will still need to change the application structure and create graphical user interfaces for your applications.

Rip and replace

The "rip-and-replace" method is exactly what it sounds like. Build completely new applications using modern programming languages, methods, databases, etc… In short, you throw out the old applications and create new ones.

Pros

Completely replacing all your old applications fixes all the problems associated with running a business on outdated applications. It leaves you with applications built to address your current business needs. It leaves you with applications that can take advantage of the most modern methods, languages, and databases. The end result is applications that are built for modern-day business, are easy to maintain, and can grow with your business.

Cons
This is perhaps the most risky approach. Throwing out your old, yet functioning, applications increases the chances that you might break something. Often, companies attempting this approach either need significant technical training in new programming methods for their staff that knows their business, or they need to train the newly hired technical staff in how their business applications need to work. Additionally, since you're building new applications from the ground up, this is by far the most time-consuming, and therefore expensive, method. In short, the end result is good, but the risks and expenses are high.

Buy new applications

You can also buy pre-built application packages. It's similar to the rip-and-replace method, but you don't have to build new applications. This method requires that you implement the new software and migrate your data over for use on the new platform.

Pros
It fixes the problems associated with running a business on outdated applications. It also leaves you with applications built for the web, complete with an event-driven structure. It's usually less time consuming than the "rip-and-replace" method, and typically comes with less risk. Also, since all the applications are pre-built, they require little testing.

Cons
This method is typically extremely expensive. You're paying more for the luxury of moving to pre-built, pre-tested applications, and the time savings that come with this method. Also, because the applications weren't designed for your company, they aren't customized to your specific business needs and won't exactly fit how you do business. These solutions usually require a fair amount of customization, either by you, or by the vendor. This increases the cost associated with an already expensive modernization method.

Extend and Surround

This approach involves gradually surrounding your old applications with completely new and modern applications. It lets you modernize different parts of your systems gradually, as the need arises. For example, if you need to bring your order entry applications to the web, you can do so without touching other aspects of your system. After that project is complete, you can focus on other areas to modernize.

Pros
This approach lets you modernize gradually at your own pace. It leaves you with truly modern applications, and eliminates much of the risk associated with the "rip and replace" method. Additionally, it lets you re-use certain parts of your old applications that are still crucial to your business. For example, your old order entry system might be character-based and only used internally. Using the "extend and surround" approach, you can build a web-based order entry application that shares logic and uses the same database as your old applications. Once you've created a web-based order-entry application for your customers, you can focus on modernizing the internal system.

Cons

It takes time to develop these new applications and requires that you have a developer on staff who is familiar with modern programming languages and methods. However, there are development tools that eliminate much of the effort (and time) as well as the need to be fluent in modern programming languages when creating these new applications .

Conclusion

Modernization is a confusing subject if you're not sure about everything that must happen in the modernization process. Before properly taking on a modernization project, you must understand what needs to happen. In order to understand what needs to happen, you must understand what has changed.

The first question you need to ask yourself is: Do my old applications address my current business needs. Chances are, the answer to that question is "no." Most businesses have changed dramatically since their original applications were created. If your applications no longer address all your business needs, then your applications must be changed.

The second question you need to ask yourself is: Were my old applications created before the web became widely used in the business world? If the answer is "yes," and you truly want modern applications, your old applications must eventually be replaced. Why must they be replaced? When the web became widely used, a couple of very important aspects of application development changed. First, programming paradigms shifted from procedurally-based to event-driven. This move was necessitated by the change in the way users interact with applications. Second, application structure moved from monolithic to an atomization of function. Monolithic applications are difficult to create, difficult to maintain, and don't translate well to the web. For these reasons, your applications must be replaced if they were created before the web became widely used.

Now that you understand what has changed, as well as the modernization options available, you can make an informed decision specific to your company. Just remember, there's much more to modernization than applications that appear modern.

--

Since 1981, mrc has helped companies of all sizes modernize their legacy applications. For more information on modernization, visit mrc's website at www.mrc-productivity.com