Getting to Java in 5 Practical Steps

How mrc turned its legacy software tool, the mrc-Productivity Series into the powerful Java m-Power™

When the top brass at mrc met to discuss the company's overall technical strategy and determine where mrc's software products were headed, their primary focus was on how mrc would shape future development tactics to fit this new plan.

At the time, the mrc-Productivity Series—mrc's original application development tool suite—was still largely character-based (**Figure A**) and tied inextricably to IBM 's iSeries (AS/400) platform.

But, despite these legacy underpinnings, the mrc-Productivity Series also offered its users Web capabilities that allowed them to develop Java servlet (server-side Java) based Web applications that were platform independent. The mrc team decided they wanted to take the product to the next level and make the mrc-Productivity Series an independent solution.

Step 1: Determine direction

You can't plan a route if you aren't sure where you're headed, so the first step is determining direction.

Figure A: *This is what the original mrc-Productivity Series' text-based interface looked like.*

```
Applications   Settings   Dictionary   System   Exit   Help
...............................................................................
MRCMENU                 the mrc-Productivity Series        DD/DD/DD  TT:TT:TT
                              Main Menu

Dictionary:  MRCWORKLIB , Demonstration Data Dictionary

            Inquiry                                Database

 1. Retrievals/Graphics                  4. Summaries
 2. Reports                              5. Maintenance
                                         6. Batch Job Streams
            Other

70. Change Current Dictionary
80. Electronic Customer Support
90. Signoff

                                                                    More...
 Selection

F3=End  F5=Work with outq  F6=Display messages  F7=Work submitted jobs
F10=Actions  F12=Cancel  Help
```

mrc knew that it wanted its modernized software to be Web-based, and platform- and database-independent. And they determined that Java was the best way to accomplish this goal. Another option available is Microsoft's .NET. However, when mrc completed an analysis of Java and .NET architecture, industry support, and open capabilities, Java beat .NET in every category. **(Figure B)**

Step 2: Determine your starting point.

You also can't plan a route if you're not sure where to start.

Now, unless you are a software vendor like mrc, or business application package vendor, you're probably not going to need to bring ALL of your current applications to Java. So, the best plan is to figure out exactly where your needs lie and break your project up into related application groupings, or departmental needs, and prioritize based on timeframe, scope, technical complexity, and ROI.

In mrc's case, because the overall project was the modernization of a full software tool, their analysis had to include all of the applications running within the tool. Your analysis, most likely, will not.

Free your mind, and the rest will follow.

You may need to change the way you think about development. Traditionally, applications were often built as one long block of code with multiple screens within each program. One reason for designing applications this way was that processors originally were able to process the code within one long program at a much faster speed than opening and closing many individual smaller programs.

Today, that is no longer the case. Instead, the recommended architecture is to modularize the code into smaller programs and interlock them to build larger application systems. That

means re-architecting your applications. mrc calls this technique the "building block" or Lego approach: creating smaller individual programs (like Legos) and then using (and re-using them) by linking or "snapping" them in myriad configurations to create robust and complex business systems.

Figure B: *This table compares Java and .NET based on architecture, industry support, proprietary v. open platform choices, and technological longevity.*

	Java	.NET
Where it runs	**Anywhere.** Java's n-tier architecture allows for platform- and database-independence. n-tier applications are made up of 3 or more modules or tiers that can live anywhere independently, but work together as one seamless application.	**Only .NET.** .NET only allows you to run on the .NET proprietary operating system, and serve applications from the .NET application server. Additionally, it has difficulty accessing outside databases such as Oracle or Sybase.
Who stands behind it?	**All the major players.** ...with the exception of Microsoft. Java is vendor-independent and privy to a community brain trust of thousands of developers and users. Bugs are solved more quickly, and the more that open source code moves into the enterprise, the greater transparency—and accountability—there will be in development.	**Only Microsoft.** .NET is dependent on one vendor, Microsoft. This is also the software vendor renowned for bugs, viruses, poor support, and serial upgrades. mrc did not want its enterprise-level software locked into that prospect.
Open or Closed?	**Open.** **Java's APIs and source code are open to everyone.** This gives the marketplace a push toward variety and promotes free enterprise.	**Closed.** .NET's source code is proprietary and under tight wraps, and their APIs are not only proprietary, but they are actually in the process of pursuing a patent to shut out competition. Software solutions are limited to Microsoft-sanctioned options.
Life Expectancy?	**Wide industry support points to long-life.** Not only do all the major players back it—with the exception of Microsoft—but there's a steady stream of new companies, products, and users/new college graduates flocking to the platform daily.	**Microsoft-dependent.** At the end of the day, .NET's longevity, as with any proprietary software, is entirely dependent on the whims of, or management of, Microsoft. Additionally, users who choose this solution are needlessly locking themselves into proprietary licensing and contracts.

So, you need to plan your re-architecture. Start by thinking of your new Web applications as one program per screen. To get your bearings, it's important to go through and measure each project or application grouping by how many total screens you'll need rather than how many individual programs you currently have.

For mrc's project, the company determined they had about 263 screens to deal with.

Now, rank your screens from easy to difficult. "Some programs are just simpler than others," explains Brian Crowley, mrc's Director of Development. "For example, lists of items in a database file, or an application for maintaining one file at a time. These are easy screens. But then, there are those that don't fit a standard look, or have more complicated code behind them. These are the difficult screens. The good news is, when you actually go through your application base, you'll probably find that the vast majority of programs fall into the easy screen category."

In mrc's case, 209 of the 263 screens fit into the "easy" category.

Then, determine your easy/difficult ratio so you can form a plan of attack. If 209 out of 263 of mrc's screens are easy, that covers 80% of the project. This 80/20 easy-difficult ratio, is mrc's ratio. However, mrc is comparing its redevelopment of a complex legacy software tool to the average application development project. It should be noted that the vast majority of most businesses will have a much easier time of it than mrc, and their easy/difficult ratio will likely be looking more like a 95/5 ratio.

But, you'll need to determine that for yourself.

Step 3: Improve Productivity

Before you start any project, you need to select the tools you will need to get the job done. Think of it as the Home Depot run before that weekend warrior project you've been putting off.

mrc knew the easiest way to accomplish their end-goal was to use their own tool, the mrc-Productivity Series, to do the majority of the modernization. Remember, even though the mrc-Productivity Series had a character-based (green screen) interface at the time, the tool was already adept at building Java servlet graphical Web applications.

One of the advantages to using the mrc-Productivity Series is that it uses fully customizable templates. These are built to not only deploy a Web-based GUI (Graphic User Interface), but to also separate that interface from the application logic and database commands into J2EE (Java servlet) applications built with n-tier architecture.

Java's n-tier architecture is what allows for true platform- and database-independence. n-tier applications are made up of 3 or more modules or tiers that can live anywhere independently, but work together as one seamless application.

Benefits of using the mrc-Productivity Series to create Java servlets:

- **Save time and money:** By using the mrc-Productivity Series to take care of modernizing the vast majority of your applications, you are saving years in development time through mrc's template-based code generator. And, even if you later choose to learn Java, or hire a Java developer…you're only paying for a fraction of the applications you would have had to manually develop otherwise. That adds up to huge cost-benefits.

- **Freedom from mrc:** When mrc says that you are free from vendors, they include themselves. m-Power™ and mrc-Productivity Series applications are written in 100% modifiable Java. All of your applications are pure Java applications as if you wrote them by hand yourself. And, because of mrc's Open Template Technology (OTT), if you have a particular way of coding you prefer, you can actually teach the mrc-Productivity Series and m-Power™ to write the code exactly as you like it.

- **Reduce risk:** You can safely modernize the majority of your applications in Java without fear of breaking anything, and without knowing Java or performing any manual coding. As you build new Java applications, you can continue to call your time-tested legacy code (logic, e.g.) as an "external object" behind the scenes. That means you can quickly deliver the Web interface your users have been demanding. And, then, if you choose to rebuild this logic in Java, you can just swap out this legacy object with the new Java code, and users are none the wiser. It's seamless.

- **Built-in Java education:** mrc has a hidden bonus in that you gain real exposure to Java, its structure, and how it works. The accessibility you have to the underlying Java source code can start you and your development team well on your way to learning Java as you use it, giving you yet another added advantage on the road to modernization.

Step 4: Move to the Web

Within each development phase, it is important to set comprehensive, and reachable goals.

mrc first addressed the 209 "easy" screens to bring them 80% closer to the first part of their goal to create a Web interface for the mrc-Productivity Series in Java servlets. They called this their "BED" interface, for "Browser-Enhanced Development."

This first 80% took one mrc developer just *four months* to complete.

The second part required addressing the remaining 54 screens, or other 20%, of the

project that were a little more difficult. (mrc considered difficult screens to be those with multiple lists, complex calculations, complicated SQL logic, maintaining unrelated data at once, etc.)

This second part, for the remaining 20% of this BED Interface goal, took mrc 9 developer-months to complete.

If mrc had developed this by hand, it would have taken about 40 months to get through the first 80% and another 17 months to get through the remaining screens. Manual development total project time: 57 months or over 4-1/2 years. (**Figure C**)

By using the mrc-Productivity Series, they developed 80% of their Java applications in four months and followed up with the remaining 20% of their applications (54 screens) in the next 9 months. mrc's total project time using a tool: 13 months or 1 year and 1 month. (**Figure D**)

That's a timesavings of 90%.

There are many benefits to this stage of Moving to the Web:

- **Save money on training:** Easier to train employees on an intuitive Web interface than a complex character-based system. Most incoming employees have already had experience with Web sites, navigation, and point-and-click menus.
- **Easier to Use / Access:** Developers and users can access applications remotely and securely via a password-

Figure C: *This diagram represents mrc's Phase 1, if mrc's developer had elected to develop* ***without the mrc-Productivity Series.***

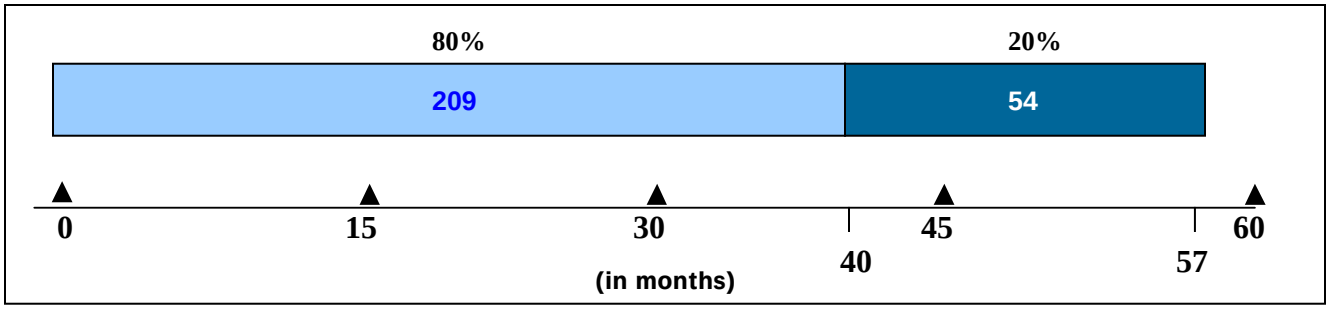
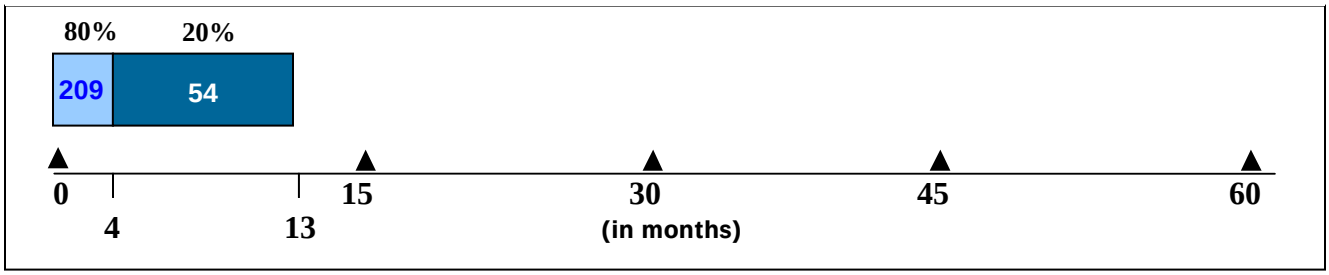


Figure D: *This diagram represents mrc's Phase 1,* ***the actual timeline****. It took one mrc developer just four months to modernize 80% of mrc's applications using the mrc-Productivity Series…and the total development time to get to mrc's BED interface took just 13 months.*



protected Web interface from any computer in the world with a Web browser.

- **Better Functionality:** A graphical Web interface improves functionality, allowing links to online manuals, service forms, product photos, links to maps/directions, even how-to videos…options that simply can't exist in character-based solutions.

- **Relieve IT pressure:** Since any legacy code still involved is behind the scenes at this point, users are generally happy to have a fast and efficient Web interface, which removes most of the immediate pressure on IT. This allows you time to begin learning Java, or figure out your next step if you decide to move toward full portability.

Most businesses could probably stop at this point. Unless there is an immediate desire to change your current platform, or to integrate disparate data systems, getting to this stage does the trick for the vast majority of IT departments.

That means getting to your final destination in **1/10 of the time**. Just imagine the additional time savings if your projects lean toward the 95/5 easy/difficult split.

In mrc's case, however, because these applications were all part of a larger solution, mrc needed to move everything to Java in order to offer a fully portable product. For instance, mrc's code-generator program, which is the workhorse behind the mrc-Productivity Series, was still written in RPG entering this stage.

That meant it could only still run on the iSeries. So, in order to make their software truly platform- and database-independent, this RPG component would need to be manually written in Java.

Note: mrc's BED interface **was still able to use its RPG-based code generator** throughout the modernization process without any interruption because the generator program is called as an external object.

Once written, the new Java-based code generator program seamlessly replaces its RPG counterpart behind the scenes…there is no downtime and no one is the wiser.

mrc counted 35 RPG-based stored procedures that needed to be rewritten in Java, and some back-end batch logic that needed to be recreated as well. But, because of all the prior work of the mrc-Productivity Series, this stage could be handled as a simple development task list.

Step 5: Achieve Full Portability

For this last little bit, to get full portability, you will need to know Java, or hire someone who does. The difference with using the mrc-Productivity Series is, by the time you get to this stage you have just a fraction of the manual coding you would have to do

otherwise. This makes a world of difference to both timeframe, and budget.

For mrc, this last stage took 19 months. A word of warning…this was with an experienced Java developer, so if you are new to Java, or you are hiring a consultant, you will need to factor the experience level of the Java developer into the timeframe.

Benefits to Full Portability:

- **True control.** Access any configuration of databases and platforms, you need to get to the truth about your business. And extend your hardware via controlled load balance. Don't let technology run you, when you can run technology.

- **Freedom from vendors.** No longer rely on any hardware or software vendor, or operating system or database. Full portability offers a way for you and/or your customers to determine the most cost-effective methods of running the technology side of business.

Even faster response times. Without the additional legacy calls within a program, Java servlets will access database information at lightning speeds, and response-time is unprecedented.

mrc's final result? m-Power™

m-Power™ is a Java-based application development tool and now, without RPG calls, it is fully portable. It accelerates development by eliminating routine infrastructure programming, giving you ready-to-deploy n-tier Java Web applications in minutes. **(Figure E)**

Because of the n-tier architecture built into m-Power, it can also be served from one platform while building applications over databases that might live on many other platforms. Run it on Linux, OS/400, Unix, Windows, and access Oracle and DB2 and MySQL at the same time…it's entirely up to you.

Additionally, because the mrc-Productivity Series and m-Power™ develop applications in

Figure E: *This figure is the new graphical Java-based Web interface of m-Power's menu, compared here, next to the original text-based menu found in Figure A.*


```
Applications   Settings   Dictionary   System   Exit   Help
...............................................................................
MRCMENU                the mrc-Productivity Series        DD/DD/DD  TT:TT:TT
                                Main Menu

Dictionary:  MRCWORKLIB , Demonstration Data Dictionary

          Inquiry                                Database

 1. Retrievals/Graphics                4. Summaries
 2. Reports                            5. Maintenance
                                       6. Batch Job Streams
          Other

70. Change Current Dictionary
80. Electronic Customer Support
90. Signoff

                                                                    More...
 Selection

F3=End  F5=Work with outq  F6=Display messages  F7=Work submitted jobs
F10=Actions  F12=Cancel  Help
```

J2EE automatically, mrc has given itself, and its customers, a real advantage in the future when it comes to integrating business processes. This distinction paves the way for using Enterprise JavaBeans or EJBs, achieving the added bonus of extended scalability.

Conclusion

In the end, there is no silver bullet. By developing its own 5 step plan, and using the mrc-Productivity Series to accelerate the process, mrc was able to modernize its complex development software product in record time. Whatever path you choose to take will necessarily be unique, and custom to your business needs.

If you're interested in exploring mrc's method, or would just like some help getting started, mrc offers free consultations, and can help you to determine how many screens you'll need and what categories these screens will fall into to get the ball rolling.

Just visit here to get started:
http://www.mrc-productivity.com/infoform.html

mrc

555 Waters Edge, Suite 120
Lombard, IL 60148
630.916.0662
mrc@mrc-productivity.com
http://www.mrc-productivity.com