

The “J2EE Journey” is Longer than Most Organizations Realize

Jacquie Barker

jjbarker@objectstart.com

Copyright 2003 by Jacquie Barker; All Rights Reserved

Many IT organizations seeking to attain proficiency with Java 2 Enterprise Edition (J2EE) technologies are unfortunately being unrealistic about the amount of time that it takes “legacy” software developers to gain such proficiency.

- Despite the proficiency of such developers with procedural programming languages such as COBOL, C, or FORTRAN, many of them missed the object technology “wave” that occurred in the early 1990’s. Thus, many are starting at square one – i.e., seeking to obtain basic proficiency with object concepts so as to harness the power of an object-oriented programming language such as Java.

The pre-OO way of designing software was known as functional decomposition. With functional decomposition, we started with a statement of the overall function that an application was to perform – e.g., “process payroll” – and then proceeded to decompose that functional requirement into subfunctions – e.g., “timecard entry”, “compute paychecks”, “print paychecks”. This (top down) decomposition into smaller and smaller units of functionality continued until we’d reached the point where any one unit was trivially simple to program; we’d then program the smallest functional units first, progressively testing and integrating them from the bottom up to build a software application.

With functional decomposition, data was an afterthought. With the object-oriented paradigm, on the other hand, we begin by focusing on the data that will be required to serve as the “bone structure” of our application: data in the form of classes of objects. Functions are considered as a second step in the object modeling process: only after we’ve invented the data structure of our classes/objects do we focus on what behaviors they’ll need to perform in order to collaborate in carrying out the “mission” of the application as a whole.

As anyone having made the transition from functional decomposition to OOP can attest, switching from a function-centric to a data-centric view when designing an application seems inside out and backwards at first! Doing so represents a major paradigm shift: that is, a fundamental change in the way we view a software engineering problem, and hence is not something that can be mastered quickly. In fact, it is not unusual for programmers to require a year or more of practice “doing it” before they really “get it”, and then only if there is a seasoned object oriented software engineer available to mentor them.

- Beyond understanding the fundamental object paradigm, one must also learn the specific Java syntax for rendering object concepts. This isn't in and of itself too hard, but with it comes the daunting task of familiarizing oneself with the hundreds of preexisting classes built into the Java language in the form of various APIs/class libraries (called "packages" in Java) so as to intelligently (re)use them in our applications.
- Next comes the challenge of comprehending/mastering the web application paradigm, which is again significantly different from traditional application development.
 - First, we must learn how to function in what is essentially a "stateless" environment by virtue of a web app's dependence on the HTTP "request-response" protocol. In a traditional application (object-oriented or otherwise), the programmer has the ability to keep data around in memory as long as it is still serving a useful purpose for the application (the notion of data being "in scope"). With a web application, on the other hand, we must learn new ways of getting data to persist across round trips between the browser and the server.
 - With a conventional application, we typically compile our application into a single executable or, in the case of Java, load various classes and objects into a single instance of the Java Virtual Machine to execute their logic. By contrast, pieces of a web application are disjointed, in the form of JavaServer pages (JSPs), HTML forms, helper "bean" classes, XML configuration files, and the like.
 - One must master a variety of different languages (Java; JavaScript; HTML; XML; custom tag libraries; etc.), each with a significantly different syntax, to craft these various components.
 - Even if each of these pieces is crafted flawlessly, the application as a whole can break if any one of these pieces is deployed incorrectly on a web server; such problems are often difficult to debug.
 - Myriad problems can be introduced with respect to how the deployment environments themselves – i.e., the web servers – are configured. Many a web application developer has beaten his/her head against the wall for days, weeks, or even months, trying to debug an application problem that ultimately turns out to be a web server configuration issue.
 - Unfortunately, in many organizations, the responsibility for web server administration falls to a separate group of individuals from those who develop web apps. Political battles often arise as fingers point back and forth between the brick wall that exists between the two organizations as to whether a web application is "broken" because of a developer error or because of a web server configuration issue.

- Finally, there are all of the individual J2EE component technologies to master – servlets; JSPs; JDBC; etc. – each with its own particular conceptual hurdles and challenges.

Regardless of one's overall proficiency as a software developer, it is unrealistic to expect that someone who is:

- a) New to the object paradigm;**
- b) New to the web app paradigm;**
- c) New to the Java language; and**
- d) New to how the various J2EE component technologies work, or even more fundamentally, what purpose each is to serve,**

can develop proficiency with all of the above in less than a two year timeframe, and only then if they receive formal training and are properly mentored. Yet, it is not unusual for organizations to embark upon their first J2EE project expecting that their “legacy” developers, starting out cold, will be able to come up to speed on all of the above, delivering a robust, non-trivial, mission-critical J2EE application in six months or less!

Unless we can convey to management how daunting a task it is to retool an IT workforce in these paradigms, technologies, and tools, projects will be fraught with missed deadlines, developers will become increasingly more stressed, and organizations as a whole will be sorely disappointed.

About the Author

Jacque Barker is a published author, professional software engineer, and adjunct faculty member at The George Washington University. With over 25 years of experience as both a “hands-on” software developer and project manager, she has spent the past 11 years focusing on object technology, and has become proficient as a “hands on” object modeler and Sun Microsystems certified Java programmer.

Her book, Beginning Java Objects (published by Apress LP; ISBN 1590591461), is focused on addressing the object crisis by:

- ❑ *Making the reader comfortable with fundamental object-oriented terminology and concepts – that is, how to ‘think’ in terms of objects;*
- ❑ *Giving them hands-on, practical experience with object modeling: that is, with developing a UML ‘blueprint’ that can be used as the basis for subsequently building an object-oriented software system;*
- ❑ *Teaching them the basics of the Java language;*
- ❑ *Illustrating how an object model is translated into a working Java application.*

Visit her web site, <http://objectstart.com>, for more information about Beginning Java Objects and her associated training offerings. Or, contact Jacquie by email at jjbarker@objectstart.com.