The Role of Enterprise Generation Language (EGL) in a Long History of Innovation on Developer Productivity

IBM has had a long history of providing software development tooling with the core goal on improving developer productivity. This article guides you through a history of innovation from the introduction of the Cross System Product in 1981, through the Visual Age family of products and to the present evolution under the IBM Rational brand.

At the core of this long history of innovation stands a mission with a set of guiding principles that have led IBM to produce its world class development platform. The use of a technology neutral language has been a key component of its success story. IBM continues to support this technology with the integration of EGL into its latest offering.

This article represents a scaled back version of Steve Choquette’s article titled VisualAge Generator – Past, Present & Future, April 2002. The author has also brought the article up to date with the inclusion of the latest chapter in this history: IBM’s acquisition and integration of Rational Software.

Long Standing Mission for IBM Developer Tooling: Developer Productivity & Robust Platform Support

Throughout the period from the introduction of CSP to today, the core product mission has been:

To provide an integrated tools environment for the rapid development of scalable, robust, mission-critical applications using traditional enterprise skills and capable of running under a variety of environments and topologies.

Over the years, IBM products continue to be shipped with many enhancements that cover both productivity improvements and technology enhancements (enhancements geared to adopt the latest technology advancements and new runtime platforms). But the core focus of all enhancements is based on the set of concerns that have always motivated customers to purchase IBM products in the first place:

- Platform neutrality,
- High productivity for enterprise programmers, and
- Scalability across a number of target environments.

These products have been able to deliver on the original mission by following these basic guiding principles:

- **Abstraction**: The most important principle behind delivering developer tooling that provides high levels of productivity and supports platform neutrality across a variety of different target runtime platforms is abstraction. Abstraction provides
a mechanism for developers to design and implement their applications with a language that is not tied to a specific technology. In doing so, the actual deployed application can be generated from this neutral development environment. As technology changes, the tooling vendor can provide drivers that transform the neutral application to the new target technology.

- **Language Neutrality**: Language neutrality provides the developer with a neutral development language which can be transformed into the language best suited for the selected target platform (COBOL, C++, Java™).

- **Platform Neutrality**: Like language neutrality, platform neutrality provides the ability to support the runtime platform that is best suited for the application. To effectively provide platform neutrality, you must support virtually any platform in the market - from the largest mainframe running to the smallest workstation.

- **Code Generation**: Code generation is the bridge between the abstract application and a concrete implementation that can be deployed to a target runtime platform. Tooling vendors can provide generation drivers that transform neutral applications to technology specific applications that can be deployed. Generating code provides a high level of productivity by generating a high percentage of application code that is associated with application architecture “plumbing”. Business rules, typically a smaller percentage of the entire application code set is written by developers. If language neutrality is used for writing business rules, the entire application can be cast into a new technology with the right code generation drivers.

- **Debugging**: A key enabling element to making these principles work in real practice is to provide the developer with a workstation-based test facility (debugger) that permits stepping through the source code – the abstract, neutral code using real data before the application is deployed into the target environment.

These guiding principles result in real, tangible benefits that will increase the likelihood of success for every project. These benefits include;

- **Less code to write**: By generating application code, particularly the “plumbing” that is often required as part of any architecture such as J2EE, the developers is shielded from having to learn and to write code for a large percentage of the application. The developer can focus on only writing the business rules.

- **Reduce training requirements**: Due to the time and cost required to train legacy developers, training has proven to be a barrier for legacy developers to move into new technologies. Code generation reduces the cost and time needed to become proficient in designing and implementing applications.

- **Proxy to new technology**: As technology evolves (and we know that this change is a constant), the training cost and the disruption caused by bringing applications to the new technologies are both very high. The neutral application combined with a code generation driver for the new technology make this transition much easier and less painful. This way, you are keeping the application definition constant but you are able to leverage the improvements in technology.
• **Proven performance and quality.** Code generation provides the benefit that the code that is generated has been tested for quality and performance. The amount of code that developers need to write for a given application is reduced to only the code that defines your business rules. The result is a reduced number of bugs for your application and a net gain in quality and performance.

**Enterprise Generation Language (EGL)**

A strategic component of the mission lies in providing a language that is neutral to the set of potentially available technology platforms. This is where EGL plays an important role. As you will see in the article below, IBM’s development tooling has always had a high level procedural language called 4GL. This language enabled business oriented enterprise developers to design and implement applications without having to focus on the underlying technology.

4GL ultimately evolved into EGL. EGL is the modern version of the neutral language and it is being continuously enhanced with new language constructs, integration with new technologies such as JSF, and new code generation drivers for new runtime platforms. Today’s EGL continues to provide developers with an unparalleled abstraction layer that enhances productivity and that provides the conduit to multiple runtime platforms. EGL has been central to making developers highly productive.

In the remainder of this article, you will see that IBM has been in the business of creating world class, high developer productivity products for a very long time. We now bring that knowledge as we create the next generation products.

**1981: Cross System Product (CSP)**

In 1981, IBM introduced an application development tool named **Cross System Product (CSP)**. Cross System Product was a mainframe application development tool intended for use by data processing professionals to increase their application development productivity. To improve programmer productivity, the product used a COBOL-like scripting language called 4GL. 4GL shielded the programmer from the programming complexity unique to each target operating platform and data store. To go to a different target platform, a user simply had to regenerate; no source code changes were required. The 4GL also provided the means to quickly validate and unit test your source code.

Cross System Product evolved over the following 11 years. Initially designed to support mainframe file systems such as Virtual Storage Access Method (VSAM), the product was gradually extended to include support for emerging database technologies. In the mid-1980s, Cross System Product enjoyed tremendous success in the marketplace as the rapid application development tool for the rapidly emerging relational databases. Cross System Product V4.1, introduced in 1992, switched from an interpretive environment to a compiled COBOL environment, significantly improving the performance of applications in the MVS® execution environment. Along with the COBOL code generation, V4.1 also introduced a workstation development environment through the CSP/2AD feature; this allowed for improved application development productivity by preventing errors, reducing keystrokes and exploiting a multi-tasked windowed environment. This feature
allowed the creation of new applications and the editing of old ones using a mouse and a graphical interface. By the early 1990s, Cross System Product was in use at over 5500 installations, including some of the world’s largest banks, retail, insurance, and manufacturing enterprises.

1994: Visual Age Generator
In the early 1990s, a large number of IBM customers were interested in taking advantage of the workstation for application development. Additional requirements included the ability to develop client/server applications (particularly the addition of Graphical User Interfaces (GUIs) to applications), the ability to access data from non-IBM vendors’ data stores, and the ability to execute application in operating environments beyond the mainframe. IBM delivered its response to these requirements in the summer of 1994 with a product called VisualGen®. VisualGen provided the means to remotely test a program on a workstation using real data, and then move it to the appropriate target platform for production verification.

VisualGen was built around emerging object-oriented technology to support GUI construction, but at its heart it preserved the same 4GL that was used by its predecessor. That meant that ALL Cross System Product users could transfer their skills and migrate their applications to this new product.

In September 1999, IBM introduced VisualAge Generator V4.0. This release provided the web transaction capability to produce applications that are accessible through the Internet. Web applications, often-called “n-tiered” applications consist of a client tier (typically browsers and thin clients), a middle tier (web and application servers), and a back-end tier of legacy applications and databases.

With VisualAge V4.0, a user could generate a web solution without knowing Java. VisualAge Generator produced the JSPs, which interact with the user via a browser, the middle tier Java code running under an application server, and the backend server code (often a COBOL application) that interacted with a database. This drastically reduced the training required to make legacy developers productive with new, emerging technologies.

In 2001, IBM also introduced a packaging option called VisualAge Enterprise Suite (VAES). VAES targeted a change form traditional mainframe based applications to those of an e-business. It was simply a packaging offering of enterprise-type application development tools bundled together into a suite. This suite gave development groups the flexibility to use the product that fit the developer skill set and the specific needs of a project.

2001: WebSphere Studio Tools
On November 5, 2001, IBM announced a new family of WebSphere application development tools to deliver the industry's broadest support for new emerging technologies such as J2EE, Web Services, XML, HTML, rich media, site design, voice,
Having built these tools on the Eclipse platform provided a common, easy-to-use interface that provides a consistent "look and feel," regardless of vendor, which can significantly reduce training requirements.

This unified interface and easy integration encouraged team-oriented programming and developer collaboration among people with different skills and different responsibilities (programmers, web content developers, business analysts, database administrators, wireless and voice application developers, and graphic artists).

It can also enable developers to customize their environment and mix-and-match tools of their choice.

In addition, WebSphere Studio tools enable developers to create applications and test them on middleware, all within the same environment. WebSphere Studio Tools provide a common development environment across Windows and Linux, so Linux developers can create enterprise-ready applications directly on top of Linux, without having to port them from Windows. This saves time and creates higher-quality applications.

**WebSphere Studio Enterprise Developer**

WebSphere Studio Enterprise Developer took advantage of the latest Internet technologies while keeping these technologies “invisible” to the business oriented developer. The product also addressed the following problem areas that effected the building of large-scale, dynamic applications deployable to WebSphere and traditional enterprise environments:

- Lack of integration of diverse skills sets across a multi-site enterprise
- Inability to reuse mission-critical legacy code when developing new applications
- Weak development processes and tooling for creation of web applications
  WebSphere Studio Enterprise Developer solves these problems through the following approaches, which will be described in subsequent sections:
- Use of abstractions to leverage traditional enterprise environments (e.g. CICS)
- Use of componentization to encourage the rapid creation of well-structured web applications
- Use of visual assembly tooling to simplify the implementation of new and changing requirements

**EGL: 4GL Renamed to EGL**

The familiar 4GL language was renamed to **Enterprise Generation Language (EGL)** with the WebSphere Studio Enterprise Developer. VisualAge Generator customers can recognize that EGL is extremely similar to 4GL. Syntax changes that make EGL consistent with the other languages used by WebSphere Studio Enterprise Developer programmers – Java, COBOL, JSPs, and PL/1 were made and migration tooling was provided to convert your existing 4GL programs to EGL.
2003: Rational

In October of 2004, IBM unveiled the next version of its development platform. This development effort represents IBM’s vision of integrating Rational Software into the IBM software portfolio. The release is designed to support “business-driven development”. With Business Driven Development the application life cycle starts with the business and moves through design, implementation and to deployment and operations.

This release integrates the WebSphere Studio tooling with the life cycle and modeling capabilities brought in by Rational – all under the umbrella of the powerful open source Eclipse platform. The new, comprehensive IDE enables developers to quickly design, develop, analyze, test, profile and deploy Web, Web services, Java, J2EE and portal applications. Optimized for IBM WebSphere software, and supporting multi-vendor runtime environments, IBM Rational Application Developer for WebSphere Software is powered by the Eclipse open source platform so developers can adapt and extend their development environment to match their needs and increase their productivity. When used with the IBM Software Development Platform, developers can access a broad range of requirements and change management functions directly from Rational Application Developer for WebSphere Software.

Again, the name of the game is developer productivity and support for multi-vendor runtime environments. However, this IDE takes it one step further by assuming a more complete role in the development life cycle. Designed to support Business-Driven Development, this development platform starts with the business and moves all the way down to deployment and operations. This completes the vision IBM began to articulate when it acquired Rational Software in 2002.

Summary

IBM’s has had a long history of bringing the industries’ most influential development tooling to market. IBM uses the expertise gained over the years, customer feedback and innovation to provide developers with the most productive, world class IDE in the industry.

The strategic importance of a technology neutral language has been proven for years and it has been a constant and a key ingredient to achieve a high level of developer productivity. First with 4GL and now with EGL, IBM has continued to invested in this technology by integrating it into several components of its latest IDE, including Rational Web Developer (RWD), Rational Application Developer (RAD), and WebSphere Enterprise Developer (WSED).

Acknowledgements

The author would like to acknowledge Steve Choquette, VisualAge Generator Product Manager, and his article, VisualAge Generator – Past, Present & Future, upon which this article is based.
The author would also like to thank the following people, whose earlier papers were shamelessly borrowed from to produce this document: Joe Pesot, Stephen Hancock, Cliff Meyers, Tim Wilson, Larry England, Rusty Edmister, Stefano Sergi, and Daphne Green.