

software
development

The lifecycle starts here.

Software Development
[May 2004](#)

Eclipse vs. NetBeans: A Plug-In Developer's Point of View

To support innovation, Java IDEs need both smart features and APIs for extending them.

Eclipse and NetBeans both claim to be universal tool platforms. The two can be compared in many ways, ranging from choice of GUI toolkit to number of available plug-ins, but my focus is on their primary purpose as extensible Java IDEs. My experience comes from building several IDE plug-ins for AspectJ, an aspect-oriented programming extension to Java.

In analyzing the current state and future potential of these IDEs, it's important to consider both the features and the APIs for extending them. The features help you decide which IDE is best for your core development needs. APIs count when considering a longer-term investment, since the IDEs' evolution is becoming an increasingly community-driven process.

The key question to consider in judging an IDE is how well it understands your system's structure. For example, most programmers have come to rely on modern IDE features such as structure views, code completion and eager error highlighting. These features make us more productive because they emphasize the program's object-oriented structure, in addition to providing facilities for manipulating and compiling source text. Both Eclipse and NetBeans offer these features.

Eclipse goes a step further, however, by making program structure a fundamental part of its architecture. For Java programmers, the result is a tight integration of advanced features such as refactoring, inheritance and call hierarchy views, structured search and structure-aware source repository support. These features save time because they keep you from squinting and grepping to figure out a problem's structure, easing the inevitable restructuring. Relying on external plug-ins to provide these features isn't good enough—they need to be at the core of the IDE. Eclipse has extensible support for compiling, refactoring and modeling programs. This is why we see features such as code completion working anywhere that references to Java elements are possible (as in Javadoc and dialog box fields). It also makes it possible for plug-in developers to extend these features and avoid reinventing the wheel (for example, with the programmatic AST rewriting needed for code generators).

Contrast this with NetBeans, which relies on the JDK's nonextensible compiler and lacks APIs that expose the object-oriented structure of Java programs. From the programmer's point of view, the tight integration of smart editing features is missing. From the plug-in developer's point of view, there isn't enough support available for extensions. This has implications for both parties, since the core IDE must support extensions specific to different application domains. For example, programmers will soon expect refactoring to work across the various resources that make up their application. In a J2EE app, this might mean that a plug-in extends the "rename" refactoring to include Java references in .xml and .jsp files. As it stands, NetBeans'

lack of an open compiler and AST model will impede the development of these vital extensions—so NetBeans may be comparable as a generic tool framework, but falls short as a Java IDE platform. The lack of Java structure-aware support in NetBeans meant that we had to reinvent the wheel for AspectJ structure-aware support. These shortcomings continue to be a severe limitation to the AspectJ plug-in for NetBeans and cause similar problems to other plug-ins that need to leverage features that are aware of Java structure.

In the near term, we can expect Eclipse to lead, with the smart features that make programmers more productive and by continuing to evolve and foster innovation by providing a better platform for plug-in developers. To catch up, NetBeans must focus on making a deep understanding of Java program structure a fundamental and extensible part of the core IDE. This would enable new ideas, such as AOP, to flourish on the NetBeans platform as they have on Eclipse.

Will such efforts as the Java Tools Community and JSR-198 (The Standard Extension API for Integrated Development Environments) help? With AspectJ, we had the pain of making separate plug-ins for Eclipse, JBuilder and NetBeans. If standardization efforts are successful, that will improve—but that was the easy part. Extending the IDE's structure model is a tougher task, which the standardization efforts aren't addressing. AOP is an idea whose time has come, and aspects are a semantic extension, as are the increasing number of XML and attribute-based languages. Programmers need the structure views and smart editing features that work for Java to make this new structure just as explicit. Eclipse doesn't yet have all the semantic extensibility we need, but it provides a platform on which to build the next round of innovative tools. If standardization efforts are to support tool innovation, they must make semantic extensibility a priority.

*At Xerox PARC, **Mik Kersten** built the IDE support for AspectJ. Today, he's a leading aspect-oriented programming expert based in Vancouver, where he works on AspectJ and related tools.*