EVALUATION OF IDE’s FOR JAVA ENTERPRISE APPLICATIONS

Jozo Dujmović and Hajime Nagashima
Department of Computer Science
San Francisco State University, USA

Abstract. In this paper we propose a quantitative model for evaluation and selection of Integrated Development Environments (IDE’s) for Java enterprise applications. Our goal is to determine the extent to which major IDE’s satisfy typical software developer requirements. Our evaluation model is based on the Logic Scoring of Preference (LSP) method for system evaluation. We present an overview of the LSP method, the structure of IDE evaluation criterion, and a sample evaluation and comparison of three competitive systems: IBM WebSphere Studio Application Developer, Borland JBuilder, and SUN ONE Studio.

Keywords: Software Evaluation, LSP Method, IDE

Introduction

Software evaluation is a process of systematic analysis of software quality. Software quality models must reflect the requirements of specific users. Such requirements are used to create criterion functions that generate a quantitative indicator of the overall satisfaction of requirements. Typical criterion functions for software evaluation are based on software quality attributes for product operation (functionality, usability, efficiency, reliability) and product evolution (maintainability, testability, portability, and reusability). Identification of general quality attributes and corresponding criteria can be based on the classical work of Boehm et al. [1], and the ISO 9126 and IEEE 1061 standards for software quality metrics [10, 11].

In the case of IDE’s for Java enterprise application development the evaluation criterion should reflect the needs of software developers. Consequently, the emphasis is on functionality, usability, efficiency and reliability of IDE’s as tools for development of sophisticated multi tier applications. Of course, the functionality, usability, efficiency and reliability are complex criteria that include a variety of individual quality attributes. These attributes are inputs for the evaluation process, and the first step in the development of a software evaluation model is a systematic process of identifying attributes that are not redundant and that completely express all relevant user requirements.

After identifying indicators that are inputs for the evaluation process, the next step is to develop a quantitative model (a criterion function) for computing the global quality of the evaluated system. In this paper, we propose an IDE evaluation criterion function based on the LSP system evaluation method. A rather extensive description of the method can be found in [3,4]. Papers that survey the method and its tools include [5,6,8]. The LSP method was first used for software evaluation and selection in the case of database systems [15]. Other recent applications include evaluation of windowed environments [7], web browsers [9], search engines [12] and various web sites [14,15,16]. The LSP method includes and substantially expands and generalizes the software evaluation model outlined in the ISO 9126 standard.

The development of enterprise applications

Java enterprise applications are based on the Java 2 Enterprise Edition (J2EE) framework presented in Fig. 1. The three main components are the desktop client tier, the server tier hosted by the application server, and the database server. The major components of the server tier are Java Server Pages (JSP), servelets, and Enterprise Java Beans (EJB). JSP/servelets serve HTTP requests with dynamic data, and EJB’s encapsulate business logic as well as the interaction with the database tier.

![IDE Diagram](image1)

Figure 1. Interaction of IDE with the J2EE framework

During the development process of enterprise applications, the IDE is mainly used to develop the application tier by what are usually large development teams. The evaluation of IDE needs to reflect such a context.

An overview of the LSP method

Software systems can be evaluated from different points of view. For example, software users and software maintenance engineers regularly have different criteria. Consequently, the first step in the evaluation process is to clearly...
define the evaluation standpoint, by specifying for whom we create the criterion function.

Software evaluation criteria always have many components and these components can be systematically identified using a system requirement tree. Such structures are defined in all software quality standards [3,4] and can be used as an initial step in building customized requirement trees. For example, if we want to evaluate performance of a software product we could use the following decomposition structure:

Performance
- Measured performance
  - Response time
  - Throughput
- Resource consumption
  - Processor utilization
  - Disk utilization

The decomposition process terminates when we derive components that cannot be further decomposed and that can be measured and evaluated. Such components are called performance variables, and denoted $x_1, x_2, \ldots, x_n$. For example, the response time and throughput can be measured and directly evaluated. The basic goal of this process is to derive attributes that are complete and not redundant.

The evaluation of performance variables is based on elementary criteria. Elementary criteria are functions that determine the level of satisfaction (the elementary preference score) for each value of the evaluated performance variable. For example, if $x$ denotes throughput, we can determine the maximum throughput $x_{\text{max}}$ that completely satisfies the user’s requirements, as well as the minimum throughput that is considered too low and unacceptable. The simplest function that computes the elementary preference score $E$ as a function of $x$ can be defined as follows:

$$E = \begin{cases} 
0, & x \leq x_{\text{min}} \\
(x - x_{\text{min}})/(x_{\text{max}} - x_{\text{min}}), & x_{\text{min}} < x < x_{\text{max}} \\
1, & x \geq x_{\text{max}}
\end{cases}$$

This elementary criterion can also be graphically presented as a preference scale:

- \begin{tabular}{c|cccccccccccc}
  $x_{\text{min}}$ & Throughput [1/sec] & $x_{\text{max}}$
  \hline
  0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 & 80 & 90 & 100 &%
- \end{tabular}

Preference scales use linear interpolation and are suitable for designing criteria with piecewise linear segments, as in the following example:

- \begin{tabular}{c|cccccccccccc}
  Response time [sec] & 8 & 4 & 2
  \hline
  0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 & 80 & 90 & 100 &%
- \end{tabular}

In this case a response time greater than or equal to 8 seconds is not acceptable and a response time less than or equal to 2 seconds is considered excellent. The response time of 4 seconds satisfies 80% of requirements and all other preferences are based on linear interpolation. For example, 3 seconds satisfies 90% of requirements and 5 seconds satisfies 60% of requirements.

After defining elementary criteria for all performance variables, we can evaluate a given system and generate $n$ elementary preferences: $E_1, E_2, \ldots, E_n$. The next step is to aggregate elementary preferences and compute the global preference $E_0 = L(E_1, E_2, \ldots, E_n)$ that reflects the global ability of the evaluated system to satisfy the evaluator’s requirements.

The aggregation function $L$ is created using a stepwise process of logic aggregation of preferences. This process usually follows the system requirement tree, going from the leaves towards the root. Preferences that are related (like the response time and throughput in our example) are aggregated using appropriate logic operators. The results are subsystem preferences (e.g., the subsystem preference reflecting measured performance). The stepwise aggregation process continues by aggregating subsystem preferences until the single global preference is computed.

We use five basic logic aggregation operators:

- Simultaneity operator
- Replaceability operator
- Neutrality operator
- Mandatory/desired operator
- Sufficient/desired operator

Each of these operators has specific logic properties. The simultaneity operator (partial conjunction) is used when we want a simultaneous satisfaction of two or more requirements. The replaceability operator (partial disjunction) is used whenever the satisfaction of some requirement can replace the satisfaction of other requirements in the group. The neutrality operator (arithmetic mean) is located between replaceability and simultaneity: it combines a moderate need for simultaneous satisfaction of requirements with a moderate replaceability capability. These three operators can be considered special cases of the “and/or” function, where the intensity of simultaneity and replaceability can be selected in the 8-level range presented in Fig. 2.

In our previous example the response time reflects the satisfaction of the user and the server throughput reflects the satisfaction of the provider. If the evaluator wants the simultaneous satisfaction of both the user and the provider, then this request can be modeled by selecting two independent parameters of the simultaneity operator:

- Degree of simultaneity
- Relative importance (weight) of inputs
The sufficient/desired operator, S/D, (or Disjunctive Partial Absorption (DPA)) is an asymmetric compound operator that combines a sufficient input \( x \) and a desired (optional) input \( y \). If the sufficient requirement is completely satisfied \((x=1)\) the output preference is \( z=1 \) regardless the value of \( y \). The main features of the mandatory/desired \([M/D]\) and sufficient/desired \([S/D]\) operators are summarized in Table 1.

### Table 1. Asymmetric (M/D and S/D) operators

<table>
<thead>
<tr>
<th>( x ) [M or S]</th>
<th>( y ) [D]</th>
<th>( z ) [M/D]</th>
<th>( z ) [S/D]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 \leq x &lt; 1 )</td>
<td>( 0 \leq y &lt; 1 )</td>
<td>( 0 )</td>
<td>( 0 \leq z &lt; y )</td>
</tr>
<tr>
<td>( 0 &lt; x \leq 1 )</td>
<td>( 0 )</td>
<td>( x-p )</td>
<td>( x-p )</td>
</tr>
<tr>
<td>( 0 &lt; x &lt; 1 )</td>
<td>( 1 )</td>
<td>( x+r )</td>
<td>( x+r )</td>
</tr>
<tr>
<td>( 1 )</td>
<td>( 0 &lt; y &lt; 1 )</td>
<td>( y &lt; z &lt; 1 )</td>
<td>( 1 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measured performance</th>
<th>( M )</th>
<th>CPA -10</th>
<th>Performance and resource consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource consumption</td>
<td>( D )</td>
<td>+5</td>
<td>---</td>
</tr>
</tbody>
</table>

**Figure 4. An example of the CPA aggregation block**

By combining appropriate preference aggregation operators, it is possible to derive sophisticated criterion functions having high expressive power. The resulting global preference \( E_0 \) can be combined with the global system cost \( C_0 \) using a cost/preference analysis (e.g. competitive systems can be compared using the \( E_0/C_0 \) ratio).

**The LSP criterion for IDE evaluation**

Our criterion for IDE evaluation reflects the needs of the typical software developer of an enterprise application. This criterion incorporates traditional components of software quality and is structured according to the following system requirement tree:

1 **Functionality**
   11 Basic functions
   111 Editor functions
      1111 Basic editor functions
      11111 Syntax sensitive editor features
      111111 Color coding
      111112 Indentation
      111113 \( (\ ) \) and \( \{\} \) balancing
      111114 Real time syntax check
      111115 Statement completion assistance
      11112 Undo and redo
   1112 Optional editor functions
      11121 Program header generation
      11122 Code documentation and presentation
      111221 JavaDoc templates and syntax check
      111222 Print format variety
   112 Debugger functions
      1121 Break points
      11211 Break point types supported
      112111 Line break points
112112 Class break points
112113 Exception break points
112114 Method break points
112115 Variable break points
112112 Break point actions
1121121 Conditional break points
1121122 Deactivating break points
112112 Execution control
1121121 Step in/out/over
1121122 Suspend thread/process
1121123 Cancel process
1121124 Remote debug
12 Templates and wizard
121 Basic application server
1211 EJB templates
1212 Servlet JSP templates
1213 App server specific deployment descriptors
122 Web services
1221 Server code generation
1222 WSDL generation
1223 Client generation
13 Optional functions
131 Completeness and quality of error messages
132 Project
1321 Project setting dialog
1322 Importing source directory
2 Usability
21 Basic usability
211 Editor usability samples
2111 Creating projects
2112 Adding classes
2113 Changing method signatures
2114 List method usages
212 Debugger usability samples
2121 Setting break points
2122 Entering expressions
2123 Number of debug panes
2124 Choosing frames
2125 Choosing threads
213 On line help and tutorial
22 Optional usability
221 Installation
2211 Installation complexity
2212 Disk space requirement
222 OS platforms supported
3 Performance and reliability
31 Performance
311 Build time
312 Memory consumption
32 Robustness and reliability
321 Edit buffer auto backup
322 Child process crash handling
323 Breaking busy loops
4 Interaction with other tools
41 Basic work flow support
411 Class design support
4111 External CASE tool integration
4112 Class/package visualization
41121 Built-in class/package visualization
41122 Availability of 3rd party class visualization
412 Version control system integration
413 J2EE servers runtime support
4131 Application server runtime support
4132 Web server runtime support
414 Web service deployment
4141 UDDI publishing
4142 UDDI browsing
42 Optional Tools
421 Built-in profiler
422 DB integration (import/export of schema design)
423 Junit driver

This system requirement tree defines 59 performance variables (listed in italics). Some of the elementary criteria are based on a single system attribute, and others are compound aggregates of several system attributes. We used more than 100 IDE attributes to build the presented criterion.

For each performance variable we need to define an elementary criterion. Following are examples of three characteristic elementary criteria:

1. Build time (311) is evaluated using a relative criterion. If the average competitive build times are $t_1, t_2, ..., t_m$ the relative build rate is $R_i = \min(t_1, t_2, ..., t_m)/t_i, 0 < R_i \leq 1$.

   The corresponding criterion is:

   $0 \quad 10 \quad 20 \quad 30 \quad 40 \quad 50 \quad 60 \quad 70 \quad 80 \quad 90 \quad 100$ %

   Elementary preference score

   $\quad 0 \quad 10 \quad 20 \quad 30 \quad 40 \quad 50 \quad 60 \quad 70 \quad 80 \quad 90 \quad 100$ %

   Relative build rate $R$

2. Creating projects (2111) evaluates the actions needed to create projects using the following point-additive scheme:
   - One mouse click selection: 1 point
   - Each menu action: 2 points
   - Each dialog box activity: 3 points

   The total sum of points $P$ reflects the complexity of creating projects, and we evaluate it using the following criterion:

   $0 \quad 10 \quad 20 \quad 30 \quad 40 \quad 50 \quad 60 \quad 70 \quad 80 \quad 90 \quad 100$ %

   Elementary preference score

   $\quad 20 \quad 17 \quad 11 \quad 8$ %

   Project create activity $P$

3. Class break points (112112) are evaluated using a binary criterion: 1 (or 100%) denotes the availability of class break point mechanism, and 0 denotes the absence of this feature.

   After developing all elementary criteria we can generate 59 elementary preferences for each of the evaluated systems. The final step is the aggregation of these preferences using an appropriate criterion aggregation structure. Fig. 5 shows the final stages of the aggregation process.
Comparison of three major IDE systems

Our criterion for IDE evaluation can be applied for evaluation of the majority of commercially available IDE’s. In this Section we use it to evaluate the IBM WebSphere Studio Application Developer 5.0 (WSAD), Borland JBuilder 9 (BJB), and SUN ONE Studio 4 Update 1 (S1). In our analysis we used evaluation copies of these products with their default parameters and list prices. The evaluation platform was Windows XP with hardware parameters that satisfy manufacturers' requirements.

The preferences of all major subsystems are presented in Table 2. Both WSAD and BJB attain very high usability levels, and have similar functionality levels. These two systems outperform S1 in all four major evaluation categories. WSAD has the best performance and BJB has the best interaction with other tools.

The three systems are clearly differentiated in the performance area. We used a benchmark with 700 Java files for measuring the build time. We also evaluated the IDE memory consumption (measured as virtual address space). BJB has very large memory consumption, and this contributed to its second place finish. In the case of S1, the evaluation copy had extremely high build time (caused by a very slow clean-up operation) and this was the primary reason for its unacceptable performance score.

WSAD gets low preference in the “interaction with other tools” category because it only supports the WebSphere application server.

Final evaluation results (global preferences) and a list price based cost/preference analysis are shown in Table 3. Global preferences of WSAD and BJB are very close and indicate that these two systems have similar global quality. Their level of satisfaction of developer requirements is sufficiently high to prove the maturity of technology that is provided by these two systems.

A low global preference of S1 is not only caused by long build time. Another basic limitation of this IDE is that its editor does not assist users by interpreting the entered program while other IDE’s understand the program structure during the editing session and offer help in correcting errors and altering class structures. S1 also has a problem with a menu structure that is substantially harder to use than in the case of WSAD and BJB.

The cost of an evaluated system is certainly one of decision parameters. However, while typical developer requirements that are used to compute global preferences are rather stable, the cost requirements vary from case to case. Consequently, our cost/preference analysis shown in Table 3 is merely an example of modeling that can be applied in actual situations. We use two formulas for computing a global quality indicator $q$:

$$q_1 = \left( \frac{C_{\text{min}}}{C} \right)^w \left( \frac{E}{E_{\text{max}}} \right)^{1-w}, \quad 0 \leq w \leq 1$$

$$q_2 = E \frac{C_{\text{min}}}{C}$$

The first formula uses the weight $w$ to express the relative importance of cost $C$, while $1-w$ is the relative importance of the global preference $E$. The minimum cost $C_{\text{min}}$ and the maximum preference $E_{\text{max}}$ are used to normalize the values of $C$ and $E$. The second formula reflects the situation where the global cost and the global preference are equally important. The values in Table 3 are computed for $w=0.3$ and are normalized ($Q = 100q/q_{\text{max}}$), so that the best system is characterized by a global quality of 100%. These results reflect the fact that the best global quality can be achieved through both a very high performance and a very low cost. In many practical cases, however, it is reasonable to restrict the analysis to only those competitive systems that satisfy a selected minimum preference level.
Conclusions

Integrated development environments must satisfy a variety of requirements. The global level of satisfaction of these requirements is used for their evaluation and comparison, and the LSP method provides a framework for building sophisticated evaluation models.

The presented evaluation results reflect the specific systems we evaluated and our general software development criterion. In other situations (tuning of IDE system parameters, different size and complexity of software development projects, and different prices), the final results could differ. However, we feel that our analysis provides two stable results: (1) the technology implemented by current IDE’s is sufficiently mature and satisfies more than 85% of general user requirements, and (2) the leading manufacturers of IDE, IBM and Borland, provide similar global system quality levels. This excludes prices that reflect marketing strategies and can change from user to user.

The presented methodology, which includes the LSP criterion model followed by a cost/preference analysis, can be used in all IDE evaluation and selection projects. Our model is based on more than 100 system attributes and can be considered sufficient for general-purpose analyses. This model can be expanded to cover situations where more detailed analysis is needed, and situations were the evaluation must include those products that closely cooperate with IDE’s.

References