LSP method and its use for evaluation of Java IDEs

Jozo J. Dujmović *, Hajime Nagashima

Department of Computer Science, San Francisco State University, 1600 Holloway Avenue, San Francisco, CA 94122, USA

Received 1 January 2005; accepted 1 June 2005
Available online 29 August 2005

Abstract

In this paper we propose a quantitative model for evaluation and selection of integrated development environments (IDEs) for Java enterprise applications. Our goal is to determine the extent to which major IDEs 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. In this paper we also introduce rectangular diagrams, an efficient new notation of LSP criteria.

© 2005 Elsevier Inc. All rights reserved.

Keywords: Software evaluation; LSP method; IDE; Andness; Orness; GCD; Partial conjunction; Andor; Partial disjunction; Orand; Partial absorption; Rectangular diagrams

1. 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 works of Boehm et al. [2], McCall et al. [19], and the ISO 9126 and IEEE 1061 standards for software quality metrics [14,15]. While quality metrics are mostly standardized, the evaluation methods that operate with these metrics are rather heterogeneous [1,8,9,16–18,20,26,28], and usually address only a specific segment of the evaluation process.

Our interpretation of evaluation methods is based on industrial decision making practice, and includes the following steps: (1) identification of the evaluated system and its user, (2) identification of user needs with respect to the evaluated system, (3) quantitative specification of user needs in the form of a criterion function, and (4) evaluation and comparison of competitive systems from the standpoint of the extent to which evaluated systems satisfy user needs. In this paper, the evaluated systems are Java IDEs, and their users are industrial developers of Java software. Our goal is to develop a general LSP criterion for IDE evaluation by expanding quantitative models for software evaluation introduced in [9,12,10,26].

In the case of IDEs for Java enterprise application development the evaluation criterion should reflect the needs of software developers. Consequently, the emphasis is on the functionality, usability, efficiency and reliability of IDEs as tools for development of sophisticated multi tier applications. Of course, 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 [5,6]. Papers that survey the method and its tools include [7,8,10,26]. The LSP method was first used for software evaluation and selection in the case of database systems [26]. Other recent applications include evaluation of windowed environments [9], web browsers [12], search engines [17] and various web sites [23–25]. The LSP method includes and substantially expands and generalizes the software evaluation model outlined in the ISO 9126 standard. The LSP method interprets IDE evaluation as a logic decision problem, and the goal of evaluation is to determine the level of global satisfaction of a comprehensive set of user requirements.
2. 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), servlets, and Enterprise Java Beans (EJB). JSP/servlets serve HTTP requests with dynamic data, and EJBs encapsulate business logic as well as the interaction with the database tier.

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

3. 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 (e.g. a software evaluation criterion can reflect needs of a typical software developer).

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 [14,15] 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:

Fig. 1. Interaction of IDE with the J2EE framework.
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 are 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. The elementary preference score belongs to the interval $[0,1]$ (or $[0,100\%)$) and it is strictly interpreted as the degree of truth in the statement that the evaluated performance variable completely satisfies user requirements. The elementary preference can be approximately interpreted as a percentage of satisfied requirements. For example, if $x$ denotes throughput, we can first determine the maximum throughput $x_{\text{max}}$ that completely satisfies given user requirements, and the minimum throughput $x_{\text{min}}$ 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 an increasing function consisting of three linear segments, as follows:

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

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

In the case of evaluating the response time $R$ the elementary criterion can be a decreasing function:

$$ E = \begin{cases} 
1, & R \leq R_{\text{min}} \\
\frac{(R_{\text{max}} - R)}{(R_{\text{max}} - R_{\text{min}})}, & R_{\text{min}} < R < R_{\text{max}} \\
0, & R \geq R_{\text{max}} 
\end{cases} $$

Preference scales can be refined using multiple linear segments and linear interpolation. The following example illustrates a criterion with four linear segments ([0, 2], [2, 4], [4, 8], [8, $+\infty$]):
In this case, a response time greater than or equal to 8 s is not acceptable, and a response time less than or equal to 2 s is considered excellent. The response time of 4 s satisfies 80% of requirements and all other preferences are based on linear interpolation. For example, 3 s satisfies 90% of requirements and 5 s satisfies 60% of requirements. The degree of truth in the statement “the response time of 5 s satisfies all user requirements” is 0.6.

After defining elementary criteria \( g_i: R \rightarrow [0,1], i = 1, \ldots, n \) for all performance variables, we can evaluate a given system and generate \( n \) elementary preferences: \( E_1 = g_1(x_1), \ldots, E_n = g_n(x_n) \). The next step is to aggregate elementary preferences and compute the global preference \( E_0 = L(E_1, \ldots, E_n) = L(g_1(x_1), \ldots, g_n(x_n)) = G(x_1, \ldots, x_n) \) that reflects the global ability of the evaluated system to satisfy all of an evaluator’s requirements. The function \( G: R^n \rightarrow [0,1] \) is the global criterion for system evaluation.

The aggregation function \( L: [0,1]^n \rightarrow [0,1] \) is created using a stepwise process of logic aggregation of preferences shown in Fig. 2. This process regularly 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 (denoted \( \Lambda \) in Fig. 2). 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 \( E_0 \) is computed.

We use five basic logic aggregation operators (aggregators):

- **Simultaneity aggregator** (partial conjunction or full conjunction).
- **Replaceability aggregator** (partial disjunction or full disjunction).
- **Neutrality aggregator** (arithmetic mean).
• Mandatory/desired aggregator (conjunctive partial absorption).
• Sufficient/desired aggregator (disjunctive partial absorption).

Each of these aggregators has specific logic properties, and the operators can be nested and combined in other ways to create a wide spectrum of logic relationships that exactly reflect user needs. The simultaneity operator (partial or full conjunction) is used when we want a simultaneous high satisfaction of all requirements in a group. The replaceability operator (partial or full disjunction) is used whenever the high satisfaction of any requirement can (partially or completely) replace the satisfaction of all 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. The simultaneity, neutrality, and replaceability are three fundamental related operators that are special cases of the Generalized Conjunction/Disjunction function (GCD, symbol $\diamond$). We implement GCD using the weighted power mean [4,11]:

$$E_1 \diamond \cdots \diamond E_k = \left( \frac{1}{k} \sum_{i=1}^{k} W_i E_i^r \right)^{1/r}, \quad -\infty \leq r \leq +\infty,$$

$$0 < W_i < 1, \ i = 1, \ldots, k, \ \sum_{i=1}^{k} W_i = 1$$

The weights are used to express the relative importance of input preferences, and the exponent $r$ is used to adjust logic properties of this aggregation function. The special cases of GCD that are used to model simultaneity are the full conjunction ($r = -\infty$, $E_1 \diamond \cdots \diamond E_k = E_1 \land \cdots \land E_k$) and the partial conjunction (andor function, $-\infty < r < 1$). The symbolic notation of the partial conjunction (andor) function is $E_1 \Delta \cdots \Delta E_k$. The special cases of GCD that are used to model replaceability are the full disjunction ($r = +\infty$, $E_1 \diamond \cdots \diamond E_k = E_1 \lor \cdots \lor E_k$), and the partial disjunction (orand function, $1 < r < +\infty$). The symbolic notation of the partial disjunction (orand) function is $E_1 \lor \cdots \lor E_k$. The desired logic properties of GCD are adjusted by selecting the appropriate value of exponent $r$. We use $E_1 \diamond \cdots \diamond E_k$ as a symbolic notation of GCD, assuming that the weights and the exponent are both adjustable. In cases where we want to provide explicit visibility of weights, a suitable symbolic notation is $W_1 E_1 \diamond \cdots \diamond W_k E_k$.

The fundamental property of the partial conjunction is andness, or the conjunction degree $\alpha$ (introduced in [4]), that is defined as a level of similarity between the partial conjunction and the full conjunction. Andness belongs to the unit interval, $0 \leq \alpha \leq 1$, and $\alpha = 1$ denotes the full conjunction. The fundamental property of the partial disjunction is orness, or the disjunction degree $\omega$, that is defined as the level of similarity between the partial disjunction and the full disjunction. Orness also belongs to the unit interval, $0 \leq \omega \leq 1$, and $\omega = 1$ denotes the full disjunction. The andness and orness are complementary indicators ($\alpha + \omega = 1$), and consequently $\omega = 0$ denotes the full conjunction and $\alpha = 0$ denotes the full disjunction. Therefore, GCD is a mix of conjunctive and disjunctive properties. In the case of partial conjunction, conjunctive properties predominate ($\alpha > 0.5$, $\omega < 0.5$), and in the case of
partial disjunction, disjunctive properties are predominant ($\alpha < 0.5$, $\omega > 0.5$). In the case of neutrality (arithmetic mean), the conjunctive and disjunctive properties are perfectly balanced ($\alpha = \omega = 0.5$). Fig. 3 shows the central location of neutrality between eight simultaneity operators and eight replaceability operators. These operators have different levels of andness/orness: the andness increases from 0 to 1 going from the top to the bottom of the list. Its complement, orness increases from 0 to 1 going from the bottom of the list to its top.

The concepts and quantitative indicators of andness and orness were introduced in 1973 as local indicators

$$a = \frac{1}{c_0} \left( E_1 \cap \ldots \cap E_k \right), \quad \omega = \frac{1}{c_0} \left( E_1 \cup \ldots \cup E_k \right).$$

These operators have different levels of andness/orness: the andness increases from 0 to 1 going from the top to the bottom of the list. Its complement, orness increases from 0 to 1 going from the bottom of the list to its top.

The concepts and quantitative indicators of andness and orness were introduced in 1973 as local indicators $a(E_1, E_2) = 1 - \omega(E_1, E_2) = [(E_1 \cap E_2) - (E_1 \therefore E_2)]/[\alpha(E_1 \cap E_2) - (E_1 \cap E_2)], W_1 = W_2 = 0.5$, and then generalized in [4] using mean values as follows:

$$\alpha = 1 - \omega = \frac{(E_1 \cup \ldots \cup E_k) - (E_1 \therefore \ldots \therefore E_k)}{(E_1 \cup \ldots \cup E_k) - (E_1 \cap \ldots \cap E_k)}, \quad (W_1 = \ldots = W_k = 1/k)$$

$$E_1 \therefore \ldots \therefore E_k = \int_0^1 \ldots \int_0^1 \left( \frac{E_1' + \ldots + E_k'}{k} \right)^{1/r} \text{d}E_1 \ldots \text{d}E_k = \mu(k, r)$$

$$\alpha(k, r) = \frac{k - (k + 1)\mu(k, r)}{k - 1}, \quad \omega(k, r) = \frac{(k + 1)\mu(k, r) - 1}{k - 1}$$

**Fig. 3.** The GCD function: 17 levels and their symbols.
The relationship between the exponent $r$ and the levels of andness and orness, based on numerical values of $\mu(k, r)$, for $k = 2$ is shown in Table 1. For $k > 2$ the values of $r$ are slightly different. The shaded area of Table 1 includes the cases where $r \leq 0$. These cases are very important in system evaluation because if the input preference is zero, the output preference must also be zero. This means that all inputs to such aggregation blocks must be positive, and the corresponding blocks are used to model mandatory requirements. Examples of such blocks for $k = 2$ include for $r = 0$ the geometric mean $E = E_1^{W_1} E_2^{W_2}$ and for $r = -1$ the harmonic mean $E = E_1 E_2 / (W_1 E_2 + W_2 E_1)$. In both cases, if $E_i = 0$, $i \in \{1, 2\}$ then $E = 0$, and it is mandatory to satisfy all input requirements if we want a positive output.

The first step in modeling simultaneity is to decide whether to use simultaneity operators $\lor$ and $\land$ that model weak conjunctive polarization without mandatory requirements, or to use $\lor$ and other operators that model stronger conjunctive polarization and mandatory requirements. 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 and mandatory satisfaction of both the user and the provider, then this request can be modeled by selecting two independent parameters of the simultaneity operator:

**Table 1**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Orness $\omega$</th>
<th>Andness $\alpha$</th>
<th>Exponent $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full disjunction (or)</td>
<td>$D$</td>
<td>1.000</td>
<td>0</td>
<td>$+\infty$</td>
</tr>
<tr>
<td>Partial Disjunction (or and function)</td>
<td>$D^{++}$</td>
<td>0.9375</td>
<td>0.0625</td>
<td>20.63</td>
</tr>
<tr>
<td></td>
<td>$D^{+}$</td>
<td>0.8750</td>
<td>0.1250</td>
<td>9.521</td>
</tr>
<tr>
<td></td>
<td>$D^{-}$</td>
<td>0.8125</td>
<td>0.1875</td>
<td>5.802</td>
</tr>
<tr>
<td></td>
<td>$D^A$</td>
<td>0.7500</td>
<td>0.2500</td>
<td>3.929</td>
</tr>
<tr>
<td></td>
<td>$D_{+}$</td>
<td>0.6875</td>
<td>0.3125</td>
<td>2.792</td>
</tr>
<tr>
<td></td>
<td>$D_{-}$</td>
<td>0.6250</td>
<td>0.3750</td>
<td>2.018</td>
</tr>
<tr>
<td></td>
<td>$D^{--}$</td>
<td>0.5625</td>
<td>0.4375</td>
<td>1.449</td>
</tr>
<tr>
<td>Neutrality</td>
<td>$A$</td>
<td>0.5000</td>
<td>0.5000</td>
<td>1</td>
</tr>
<tr>
<td>Partial Conjunction (and or function)</td>
<td>$C^{-}$</td>
<td>0.4375</td>
<td>0.5625</td>
<td>0.619</td>
</tr>
<tr>
<td></td>
<td>$C^{-}$</td>
<td>0.3750</td>
<td>0.6250</td>
<td>0.261</td>
</tr>
<tr>
<td></td>
<td>$C^{+}$</td>
<td>0.3125</td>
<td>0.6875</td>
<td>-0.148</td>
</tr>
<tr>
<td></td>
<td>$C^A$</td>
<td>0.2500</td>
<td>0.7500</td>
<td>-0.72</td>
</tr>
<tr>
<td></td>
<td>$C^{+-}$</td>
<td>0.1875</td>
<td>0.8125</td>
<td>-1.655</td>
</tr>
<tr>
<td></td>
<td>$C^{+}$</td>
<td>0.1250</td>
<td>0.8750</td>
<td>-3.510</td>
</tr>
<tr>
<td></td>
<td>$C^{++}$</td>
<td>0.0625</td>
<td>0.9375</td>
<td>-9.06</td>
</tr>
<tr>
<td>Full conjunction (and)</td>
<td>$C$</td>
<td>0</td>
<td>1.000</td>
<td>$-\infty$</td>
</tr>
</tbody>
</table>
• Degree of simultaneity (andness).
• Relative importance of inputs (weights).

If the evaluator considers that the throughput and the response time are equally important (both weights are 50%), and if the selected intensity of simultaneity is medium (CA) in the mandatory range, this yields the aggregation block shown as a “rectangular diagram” in Fig. 4. In this example, the requirement for simultaneous satisfaction of input requirements yields relatively low resulting preference (65%) caused by low satisfaction of the throughput requirement.

In system evaluation models, we must frequently combine mandatory and non-mandatory (desired or optional) inputs. The mandatory/desired operator, M/D, (or Conjunctive Partial Absorption (CPA) [5,6]) is an asymmetric compound operator that combines a mandatory input \( x \) and a desired (optional) input \( y \). If the mandatory requirement is not satisfied \( (x = 0) \) the output preference is \( z = 0 \) regardless of the value of \( y \). If \( 0 < x \leq 1 \) and \( y = 0 \) then \( z \approx x - xP \), where \( P \) denotes a penalty (typical range: \( 10\% < 100P < 30\% \)). If \( 0 < x < 1 \) and \( y = 1 \) then \( z \approx x + xR \), where \( R \) denotes a reward (typical range: \( 5\% < 100R < 15\% \), and regularly \( P > R \)). The properties of CPA are summarized in Table 2.

CPA represents a generalization of the classic absorption theorem \( z = x \wedge (x \vee y) = x \), obtained if conjunction is replaced by the partial conjunction and disjunction is replaced by the partial disjunction \( (z = x \Delta (x \vee y)) \) or the arithmetic mean \( (z = x \Delta (Wx + (1 - W)y)) \).

The parameters of this function can be computed from the desired average penalty and reward values. Fig. 5 shows two rectangular diagrams of a CPA aggregator with an average penalty of 10% and an average reward of 5% for our previous example of combining measured performance and resource consumption. Measured performance is considered critical and it is a mandatory requirement. It is desired that a

![Fig. 4. An example of the medium simultaneity mandatory aggregation block (partial conjunction, andor).](image)

| Table 2 | Asymmetric (M/D and S/D) operators |
|---|---|---|---|
| \( x \) [M or S] | \( y \) [D] | \( z \) [M/D] | \( z \) [S/D] |
| 0 | \( 0 < y \leq 1 \) | 0 | \( 0 < z < y \) |
| \( 0 < x \leq 1 \) | 0 | \( x - xP \) | \( x - xP \) |
| \( 0 < x < 1 \) | 1 | \( x + xR \) | \( x + xR \) |
| 1 | \( 0 < y < 1 \) | \( y < z < 1 \) | \( \approx 1 \) |
high performance is attained with a low resource consumption; however, the resource consumption is not mandatory. The first diagram in Fig. 5 is the compound symbolic notation of the CPA block. The second diagram is detailed and shows the anatomy of the CPA block that consists of combined DA and CA operators. More details about asymmetric operators can be found in [6].

The sufficient/desired operator, S/D, (or Disjunctive Partial Absorption (DPA) [5,6]) 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 $z \approx 1$ regardless the value of $y$. The main features of the mandatory/desired [M/D] and sufficient/desired [S/D] aggregators are summarized in Table 2. A sample compound and detailed rectangular diagrams of a DPA function are presented in Fig. 6.

DPA represents a generalization of the classic absorption theorem $z = x \lor (x \land y) = x$, obtained if disjunction is replaced by the partial disjunction and conjunction is replaced by the partial conjunction ($z = x \lor (x \land y)$) or the arithmetic mean ($z = x \lor (Wx + (1 - W)y)$).

The basic preference aggregation operators (simultaneity, neutrality, replaceability, CPA, and DPA) can be combined and nested in a variety of ways. As a typical example, Fig. 7 presents a compound CPA operator with two mandatory and three desired inputs.

Figs. 8–11 and Table 3 show two nested operators:

- Mandatory/Desired/Optional (MDO) [26].
- Sufficient/Desired/Optional (SDO).

In the MDO case, we distinguish three levels of asymmetry:

- Mandatory input (if its value is zero then the output is zero regardless of other inputs).

---

<table>
<thead>
<tr>
<th>Sufficient input</th>
<th>S</th>
<th>DPA</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>A</td>
<td>61</td>
<td>DA</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td>39</td>
</tr>
</tbody>
</table>

---

Fig. 6. Compound and detailed rectangular diagrams of a sample DPA aggregation block.
Table 3
Properties of nested D-M/D/O and S/D/O operators

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 &lt; t ≤ 1</td>
<td>0</td>
<td>≈ trR</td>
</tr>
<tr>
<td>0</td>
<td>0 ≤ y ≤ 1</td>
<td>0</td>
<td>0</td>
<td>≈ y(1 − p)R</td>
</tr>
<tr>
<td>0</td>
<td>0 &lt; y ≤ 1</td>
<td>0 &lt; t ≤ 1</td>
<td>0</td>
<td>0 &lt; z &lt; max(y, t)</td>
</tr>
<tr>
<td>0 &lt; x ≤ 1</td>
<td>0</td>
<td>0</td>
<td>≈ x(1 − p)</td>
<td>≈ x(1 − P)</td>
</tr>
<tr>
<td>0 &lt; x ≤ 1</td>
<td>0</td>
<td>1</td>
<td>≈ x(1 − P)</td>
<td>z &lt; max(r, x)</td>
</tr>
<tr>
<td>0 &lt; x ≤ 1</td>
<td>1</td>
<td>0</td>
<td>min(1 − p, x) &lt; z</td>
<td>min(1 − p, x) &lt; z</td>
</tr>
<tr>
<td>0 &lt; x &lt; 1</td>
<td>1</td>
<td>1</td>
<td>≈ x(1 + R)</td>
<td>≈ x(1 + R)</td>
</tr>
<tr>
<td>1</td>
<td>0 &lt; y &lt; 1</td>
<td>0 &lt; t &lt; 1</td>
<td>min(y, t) &lt; z &lt; 1</td>
<td>≈ 1</td>
</tr>
</tbody>
</table>

Fig. 7. An example of compound CPA operator.

Fig. 8. A general form of the D-nested M/D/O operator.

Fig. 9. A general form of the M-nested M/D/O operator.

Fig. 10. M/D/O operator with weighted compensation.

Fig. 11. A general form of the nested S/D/O operator.
• Desired input (if the mandatory input is positive, the desired input provides the penalty/reward effects typical for CPA operators).
• Optional input (similar to the desired input, but with lower compensation power).

Three versions of the M/D/O operator shown in Figs. 8–10 differ in their properties. In the case of the D-nested operator the desired and optional inputs are not independent: the zero value of the desired input prevents the optional input from making any (positive or negative) effect. In the case of M-nesting, the optional input is independent of the desired input and the desired input has higher compensation power because it affects the mandatory part of the operator. The M/D/O operator with weighted compensation is not nested; it simply splits the compensation part of the desired input into two parts, where the desired component is two times more effective than the optional component. The differences between these three aggregation operators are subtle, and our point is to show that the LSP method offers subtleties in the logic aggregation of preferences that can be used to build highly precise complex criteria.

The SDO aggregator (Fig. 11) is symmetrical to the MDO operator, but it has much lower usability. The complete satisfaction of the sufficient input causes the complete satisfaction at the output. If the sufficient input is zero, then this can be compensated by the desired input, and (to a lesser extent) by the optional inputs. A summary of basic properties of the D-nested M/D/O and S/D/O operators is shown in Table 3.

The presented hierarchical process of logic aggregation of preferences could be compared with hierarchical models based on fuzzy integrals [21]. Some hierarchical models based on fuzzy integrals are equivalent to models having only two layers; consequently, such hierarchical models contribute to modularity, but not to the logic expressiveness. In the case of the logic aggregation of preference based on GCD, such an effect may occur only in unlikely cases where all aggregators use the same value of exponent $r$. For example, the following equivalence holds only for $r \in \{-\infty, 1, +\infty\}$:

$$W_1 x_1 \diamond W_2 x_2 \diamond W_3 x_3 = W_1 x_1 \diamond (W_2 + W_3) \left(\frac{W_2}{W_2 + W_3} x_2 \diamond \frac{W_3}{W_2 + W_3} x_3\right)$$

$$= (W_1 + W_2) \left(\frac{W_1}{W_1 + W_2} x_1 \diamond \frac{W_2}{W_1 + W_2} x_2\right) \diamond W_3 x_3$$

For other values of $r$ this equivalence of one-layer and two-layer models holds only approximately. However, the LSP criteria (including the criterion presented in this paper) can use different aggregators in each block, and consequently the hierarchical reduction of GCD models is neither desirable nor possible. The goal of this process is to maximize both expressiveness and modularity, giving an opportunity to the user to exactly express desired logic relationships and minimize errors in the intuitive selection of parameters (modules with a small number of inputs are more accurate than modules with a larger number of inputs).

The process of logic aggregation of preferences is systematically applied to all subsystems as shown in Fig. 2. At the end, it generates the global preference, which is a
single scalar indicator of the quality of the evaluated system as a whole. The resulting
global preference $E_0$ can be combined with the global system cost $C_0$ using a cost/
preference analysis [5] (e.g. competitive systems can be compared using the $E_0/C_0$
ratio). An example of cost/preference analysis is presented later in this paper, in
the context of a comparison of IDE environments.

4. The LSP criterion for IDE evaluation

Our criterion for IDE evaluation reflects the needs of the typical software develop-
er of an enterprise application. This criterion incorporates traditional components
of software quality, and it is structured according to the following system require-
ment 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
            11212 Break point actions
               112121 Conditional break points
               112122 Deactivating break points
      1122 Execution control
         11221 Step in/out/over
         11222 Suspend thread/process
         11223 Cancel process
1123 Expression evaluation
1124 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 support
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 \( t_1, t_2, \ldots, t_m \) the relative build rate is

\[
R_i = \frac{\min(t_1, t_2, \ldots, t_m)}{t_i}, \quad 0 < R_i \leq 1.
\]

The corresponding criterion is

![Relative build rate R](image)

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

![Project create activity P](image)
### Inputs: Elementary Criteria

<table>
<thead>
<tr>
<th>Operator Block ID</th>
<th>Operator Block ID</th>
<th>Operator Block ID</th>
<th>Operator Block ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color coding</td>
<td>Syntax sensitive editor features</td>
<td>Basic editor functions</td>
<td>Editor functions</td>
</tr>
<tr>
<td>Indentation</td>
<td></td>
<td></td>
<td>CPA -10 +5</td>
</tr>
<tr>
<td>Balancing of ( ) and [ ]</td>
<td></td>
<td></td>
<td>Basic functions</td>
</tr>
<tr>
<td>Real time syntax check</td>
<td></td>
<td></td>
<td>CPA +6</td>
</tr>
<tr>
<td>Statement completion</td>
<td></td>
<td></td>
<td>CPA +10 / -10</td>
</tr>
</tbody>
</table>

**Color coding:** 20  **Indentation:** 20  **Balancing of ( ) and [ ]:** 20  **Real time syntax check:** 20  **Statement completion:** 20  **Undeclared:** 20  **Program header generation:** 33  **Java Doc:** 75  **Print format:** 25  **C-++ Syntax sensitive editor features:** 80  **Code documentation:** 67  **C-++ Breakpoint types:** 60  **Breakpoints:** 30  **C++ Breakpoints:** 50  **Deactivating breakpoints:** 50  **Step in/out/over:** 33  **Suspend threads/processes:** 33  **Cancel process:** 34  **Expression evaluations:** 20  **Remote debug:** 20  **EJB templates:** 33  **C++ Basic application servers:** 75  **Templates and wizard:** 20  **App. server specific deployment descriptors:** 33  **Web services server code:** 40  **Web services:** 40  **Web services client:** 20  **Quality of error messages:** 60  **Project setting dialog:** 40  **Projects:** 40  **C++ Optional functions:** 40  **Creating projects:** 25  **Adding classes:** 25  **Changing method names:** 25  **Listing method usages:** 25  **Setting break points:** 20  **Entering expressions:** 20  **Number of panes during debug session:** 20  **Choosing stack frame:** 20  **Choosing threads:** 20  **Quality of on line help:** 20  **Installation complexity:** 45  **Installation:** 40  **C++ Optional usability:** 40  **Supported OS:** 60  **Build time:** 50  **Memory consumption:** 50  **C++ Performance:** 40  **Edit buffer auto backup:** 33  **Robustness & reliability:** 33  **Child process crash handling:** 33  **Breaking busy loop:** 34  **Embedded class visualization:** S DPA 0/+60  **Class visualization:** D CPA 0/-70  **CASE tool support:** 20  **C++ Basic workflow support:** 20  **Version control system:** 20  **Application servers runtime support:** 50  **J2EE runtime:** 40  **Web server:** 50  **UDID publishing:** M CPA -20 +10  **Web service deployment:** 20  **Profiler:** 40  **DB integration:** 20  **C++ Optional tools:** 40  **JUnit test driver:** 40

**Fig. 12. Main components of the IDE preference aggregation structure.**
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. 12 shows the global structure of the preference aggregation process.

5. Comparison of three major IDE systems

Our criterion for IDE evaluation can be applied for evaluation of the majority of commercially available IDEs. In this Section, we use it to evaluate the IBM WebSphere Studio Application Developer 5.0 (WSAD) [13,22], Borland JBuilder 9 (BJB) [3], and SUN ONE Studio 4 Update 1 (S1) [27]. In our analyses and measurements 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.

Input data (all performance variables, and costs) were collected in June 2003. Of course, costs and some system parameters evolve over time. Consequently, the primary goal of our presentation is to show properties of the LSP decision model for evaluation of a complex software system. The exact comparison of the three analyzed systems may vary over time depending primarily on current costs and development of new features.

### Table 4
Subsystem preferences [%]

<table>
<thead>
<tr>
<th>IDE</th>
<th>Function</th>
<th>Usability</th>
<th>Performance</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAD</td>
<td>88.6</td>
<td>93.8</td>
<td>86.5</td>
<td>73.5</td>
</tr>
<tr>
<td>BJB</td>
<td>90.7</td>
<td>95.4</td>
<td>73.5</td>
<td>90.5</td>
</tr>
<tr>
<td>S1</td>
<td>63.9</td>
<td>61.8</td>
<td>51.4</td>
<td>62.5</td>
</tr>
</tbody>
</table>

Fig. 12 (continued)
The preferences of all major subsystems are presented in Table 4. 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 cost/preference analysis (based on list prices) are shown in Table 5. 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 IDEs 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 5 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 5 are computed for $w = 0.3$ and

<table>
<thead>
<tr>
<th>IDE</th>
<th>$E_0$ [%]</th>
<th>$C_0$ [$$]</th>
<th>$Q_1$ [%]</th>
<th>$Q_2$ [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAD</td>
<td>85.4</td>
<td>3060</td>
<td>100</td>
<td>93.4</td>
</tr>
<tr>
<td>BJB</td>
<td>87.0</td>
<td>3500</td>
<td>97.3</td>
<td>83.2</td>
</tr>
<tr>
<td>S1</td>
<td>59.7</td>
<td>1995</td>
<td>88.4</td>
<td>100</td>
</tr>
</tbody>
</table>
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 either a very high performance, or 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.

6. Conclusions

The presented LSP methodology is based on a comprehensive set of input attributes and elementary criteria that precisely reflect all individual user requirements. The main power of the LSP approach is the ability to build a versatile and precise model of logic aggregation of preferences. By combining appropriate preference aggregation operators, it is possible to derive sophisticated criteria that have high expressive power and flexibility to model a wide spectrum of logic relationships that precisely reflect the user’s needs.

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 results could differ. However, we feel that our analysis provides two stable results: (1) the technology implemented by current IDEs 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. The model can be expanded to cover situations where we need a more detailed analysis, and situations were the evaluation must include those products that closely cooperate with IDEs.

References


