IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

#ITB4339

Java integrated development environments' support for reuse-oriented software development

Jenni Ristonmaa, Jarmo Ahonen, and Marko Forsell
IT Research Institute, University of Jyväskylä, P.O. Box 35, 40351 JYVÄSKYLÄ, Finland
Phone +358 14 260 1211, Fax +358 14 260 2544, [jenni.ristonmaa | jarmo.ahonen | marko.forsell]@titu.jyu.fi

ABSTRACT

Component reuse is a promising direction to develop software more efficiently and cost effectively. One part of software development is the actual programming with an integrated development environment (IDE). We studied three Java IDEs and how they support reuse-oriented software development. We derived evaluation criteria from a known reuse model. As a conclusion we suggest that current Java IDEs need to improve their support for the reuse process.

1. INTRODUCTION

To cope with the current trend to produce quality software in tightening schedules software developers see reuse as one possible answer (e.g., Lim, 1997, McIllroy, 1968). Reuse of components is one approach to handle reuse (Biggerstaff & Richter, 1987). The basic idea in component reuse is to use some results of the development effort more than once (Basili et al., 1992, Krueger, 1992). To be successful, reuse has to be systematic: it has to be planned in advance and it must be acknowledged in every phase of software development cycle (Lim, 1997). One part of this cycle is programming. Here integrated development environments (IDEs) are especially important. Early IDEs included such tools as an editor and a compiler but currently these environments may include, among other things, source code control, library management, support for workgroups, and version control (Kölling & Rosenberg, 1996).

Java has emerged as one of the most popular programming languages and its advantage is that it closely follows emerging trends in software development. One such trend is the support for component-based development in the form of JavaBeans standard. JavaBeans brings component technology to the Java platform. With JavaBeans you can create reusable, platform-independent components (Sun, 2000).

Our research question is: "Do Java IDEs support the creation and reuse of code components?" As the only components supported by the chosen IDEs are JavaBeans, we limited our study to JavaBeans components.

2. SELECTION OF JAVA IDE:S AND RESEARCH METHOD

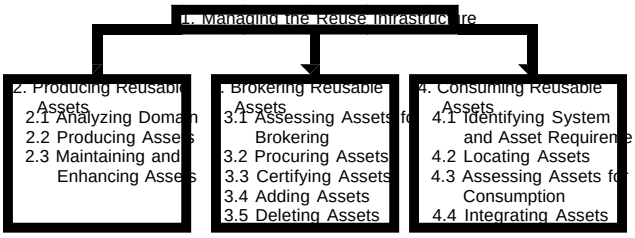
We selected three Java IDEs that reflect the current state-of-the-practice in Java programming. We chose the environments considering the market share and how well the supplier is known. Using our criteria we selected:

- Forte for Java Community Edition 1.0 Windows Version by Sun Microsystems Inc.,
- Borland JBuilder 3.0 Professional by Inprise Corporation, and
- VisualAge for Java Enterprise Edition Version 3.0 by IBM.

We planned to use Visual J++ 6.0 Professional Edition from Microsoft but we excluded it for its strong orientation towards ActiveX and Windows.

We wanted to find out how Java IDE supports reuse processes involved in component (JavaBean) reuse. To base our evaluation framework on a well known model we chose Lim's (1997) reuse model over alternatives (e.g., NATO, 1992, Karlson, 1995, see Forsell et al., 2000). Lim's model is not biased toward any specific implementation technology and it includes code components as well as other software development artifacts (assets in Lim's vocabulary). Figure 1 shows Lim's reuse model, its four major activities, and tasks in them.

Figure 1. The Reuse Process (Lim, 1997)



Because Lim's model is not focused solely on the code component reuse it has activities and tasks which are irrelevant for the programming phase. We argue that the following tasks are not and should not be concern for IDE: analyzing domain, component assessing, procuring an asset, certifying an asset, identifying system requirements, and assessing assets. Also, managing the reuse infrastructure as a whole is not a concern for IDE.

3. RESULTS OF THE EVALUATION

The results of the evaluation are presented below. The main interest is in the differences and interesting features.

3.1 Producing a Component

All IDEs have fairly similar tools and techniques. The main tools are wizards and dialogues, but only VisualAge offers dialogs for the creation and addition of methods. Only VisualAge creates the BeanInfo class for the produced component. With JBuilder and Forte users have to add the BeanInfo manually, which is easy. The possibility to create this functionality by using visual tools is present only in VisualAge.

Forte feels slow, which makes it difficult to use, and its structure of menus is not consistent or intuitive. JBuilder offers methods to add comments to the code and its BeanExpress is well thought out. VisualAge offers fairly sophisticated and usable visual tools for the creation of components, it is even possible to create complicated components with minimal programming. In VisualAge a great deal of functionality can be defined by applying the visual tools.

VisualAge is clearly the best tool for producing components. JBuilder has better structure and is more practical than Forte.

This paper appears in the book, *Managing Information Technology in a Global Economy*, the proceedings of the Inforrmation Resour es Management Association International Conference. Edited by Mehdi Khosrow-Pour.

3.2 Maintaining and Enhancing a Component

Maintenance and enhancement generally means bug removal and feature enhancement. Maintenance is unnecessarily complicated if IDE allows users to save source code components without compilation. VisualAge allows users to save and add source-level components, but it compiles the code before saving the component. Forte allows users to save incorrect components. JBuilder allows only .class- and .JAR-files to be saved as JavaBeans requiring fairly strict conformance from components. Unfortunately all environments allow users to modify components in incorrect ways.

The only IDE of these three to support versioning of components is VisualAge. This feature is important when considering the support offered by IDEs for maintenance and enhancement.

3.3 Adding a Component

In Forte and VisualAge it is possible to add source code components to the menu structure. In VisualAge those components are confirmed to be at least syntactically right. JBuilder allows only .class- and .JAR-files to be added to the menu structure.

All IDEs have specific menu structures into which components should be added. Both VisualAge and JBuilder allow users to add their own components to any position in a JavaBeans menu structure. Forte stops working if users try to add a component to any other position than the Beans-sheet. JBuilder and VisualAge offer a straightforward and logical way for adding components. In Forte the technique is more complicated and not so easy to use.

3.4 Deleting a Component

The support for removing a component from the menu structure is present in JBuilder and VisualAge, while Forte has no such support. If users want to delete components from Forte, they have to use filesystem tools and remove the files associated with the components.

The deletion of a component is immediately shown in the menu structures of VisualAge. JBuilder and Forte remove the component from the menu structure only after the IDE has been restarted.

3.5 Identifying a Component

IDEs do not identify the JavaBeans or Enterprise JavaBeans component requirements correctly. The only requirement identified by every IDE is that the component class must be public. JBuilder often correctly identifies all requirements, but it makes mistakes. VisualAge checks if the component is syntactically right and if it is generated by using the automatic tools of VisualAge it fulfills all requirements.

3.6 Locating a Component

The menu structures reserved for components are very similar and they are easy to use. The menu structures do not support classification of components, although every IDE has specific menu sheets or structures for custom components and components shipped with the IDE. The only IDE which offers additional tools for the location of components is VisualAge with its Choose Bean tool.

3.7 Integrating a Component

Only VisualAge and Forte have visual tools for the integration of components. Users can either use those tools or do the integration through traditional programming, or users may use a combination of programming and tools. In this respect VisualAge is more sophisticated than Forte.

3.8 The Best Features and Ranking of the Evaluated IDEs

The evaluated IDEs are considered in the perspective of Lim's model. The best features of Forte are its visual tools for component integration and intuitively appealing naming conventions in its menu structures. The best feature of JBuilder is that it allows only .class and .JAR -files to be added as components, and its BeanExpress-tool is a good tool. The best features of VisualAge are the versioning of classes, its visual integration tools, its tool for component location, and its automatic compilation of source code before saving.

The IDEs have been ranked according the results and the answers in table 1. The ranking is shown in table 2 (1 = best, 3 = worst).

Table 1: Additional Questions for Ranking.

Criteria	Question	Forte	JBuilder	VisualAge
Producing	Is the BeanInfo class required?	No	No	Yes
Maintaining and enhancing	Is versioning of classes supported?	No	No	Yes
Adding	Is syntactical correctness required?	No	Yes	Yes
Deleting	Are there tools for the deletion of components?	No	Yes	Yes
	Are menus and data structures updated immediately after the deletion?	No	No	Yes
Identifying	Does the IDE prohibit the saving of syntactically erroneous components?	No	No	Yes
Locating	Are there additional tools for the location of components?	No	No	Yes
Integrating	Are there visual tools for the integration of components?	Yes	No	Yes

Table 2: The Ranking of the IDEs

Criteria	Forte	JBuilder	VisualAge
Producing	3	2	1
Maintaining and enhancing	3	2	1
Adding	3	1	2
Deleting	3	2	1
Identifying	3	1	2
Locating	2	3	1
Integrating	2	3	1

4. DISCUSSION

We have evaluated Java IDEs from the point of view of how they support reuse process. We based our evaluation on Lim's reuse model and derived evaluation criteria from it. It seems that Java IDEs, in general, still need improvement. IBM's environment shows promising directions for future in keeping with their efforts to provide reuse-oriented features. The support for reuse requires improvement.

Companies can use these results and our approach when they want to choose a Java IDE for themselves. However, this implies that companies know their needs for reuse support and find out which tool fits these needs best. Furthermore, we believe that the evaluation method is not limited to the evaluation of Java IDEs, it can be used to evaluate any programming language specific IDE. Also, Lim's framework as a whole can be used to evaluate software development methods (Forsell et al. 2000), and we believe it can be used to evaluate software engineering environments, i.e.

repositories, CASE tools, and project management tools.

Main limitation of the study is that it focuses on Java language and three Java IDEs. We need more research on how to add reuse support in any IDE. Furthermore, IDEs and other tools that support software development should be integrated more closely together and these should support and integrate components in different levels of abstraction from the reuse point of view.

5. REFERENCES

Basili, V., Caldiera, G., Cantone, G., A reference architecture for the component factory. ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 1, January 1992, 53-80.

Biggerstaff, T., Richter, C., Reusability framework, assessment, and directions. IEEE Software, March 1987, 41-49.

Forsell, M., Halttunen, V., Ahonen, J., Use and identification of components in component-based software development methods. Software Reuse: Advances in Software Reusability, Proceedings of the 6th International Conference , ICSR-6, , 2000, 284-301.

Karlson, E. (Ed.), Software Reuse: A Holistic Approach, Chichester: John Wiley and Sons, 1995.

Krueger, C., Software reuse, ACM Computing Surveys, Vol. 24, No. 2, June 1992, 131-182.

Kölling, M., Rosenberg, J., An object-oriented program development environment for the first programming course. Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, 1996, Pages 83 - 87

Lim, W., Management of Software Reuse. Addison-Wesley, 1997.

McIlroy, D., Mass produced software components. Report on a conference by the NATO science committee, Garmish, Germany, October 7-11 1968, in Naur, P., Randel, B., Buxton, J. (Eds.) Software Engineering: Concepts and Techniques, New York: Petrocelli/Charter, 1976, 88-98.

NATO, NATO standard for the development of reusable software components. Volume 1 (of 3 documents), 1992. http://www.asset.com/WSRD/abstracts/archived/ABSTRACT_528.html, accessed 1st of June 2000.

Sun, 2000, http://java.sun.com/docs/books/tutorial/javabeans/index.html, accessed 11th of September 2000.