INFLUENCE OF JAVA ON SOFTWARE ENGINEERING EDUCATION

Pete Knoke (Chair)
Univ. of Alaska Fairbanks
ffpjk@uaf.edu

PANELISTS:

Wendy Doube
Monash University
Wendy.Doube@infotech.monash.edu.au

John Lewis
Villanova University
john.lewis@villanova.edu

Ariel Ortiz
ITESM State of Mexico
aortiz@campus.cem.itesm.mx

Alejandro Teruel
Simon Bolivar Univ
teruel@ldc.usb.ve

ISSUES:
How much influence will Java have on SE Education, and why?
How should SE Education change to accommodate Java/OOD?
What are some good Java-specific SE practices?

John Lewis position

Java will have a positive influence on software engineering practices at several levels. However, Java can also lead to software that lacks certain desirable qualities. The burden is still on educators and practitioners to uphold the best ideals. At least Java provides several language-based elements that directly support these high-quality ideals. Some good Java characteristics include its formal interface structure, its clean polymorphism model, and its support for automatic documentation generation. Java interfaces allow a designer to provide clean interactions between system components, and creates a formal distinction between the interface and underlying implementation. The consistent polymorphism model in Java results in a cleaner design model. Some present Java techniques take insignificant short cuts but inflict long-term penalties. For example, the use of an anonymous inner class to implement an event listener is a short cut that greatly increases the cognitive overhead for the software, and is useful only in that it saves a few keystrokes. The use of anonymous inner classes results in horribly structured and undocumented code. The fact that Java provides such a technique does not facilitate the goal of creating well-engineered software. These issues are but a few that contribute to the discussion of Java as a sound software engineering tool. Overall, I would argue that Java provides better facilities in this regard than other languages, and those features that are ill-conceived can be avoided with proper education and standards.

Alejandro Teruel position

At the Universidad Simón Bolívar we have been using Java in undergraduate-level Software Engineering courses since January 1999. Most of what I have to say applies equally well to most object-oriented programming languages. Teaching OOSE is harder and requires different skills from teaching traditional SE. It requires more modeling activities and provides a larger design space to play around in. It requires working on larger projects in order to appreciate its benefits and the size of these projects threatens to overflow any one-term, one-teacher courses. Current SE requires treating "software maintenance" as the key technical problem and team skills as the key cultural problem. While the OOSE penchant for incremental methodologies is a step in the right direction as regards the key technical problem, it should be combined with more material on group dynamics in order to tackle the key cultural problem.Modeling takes up far more time to cover and to do in an OO approach than in traditional structured approaches; it is also more complex. More time is required to teach OO design. Most student designs do not really take advantage of OO programming concepts and they tend to revert to more traditional styles.

* The students' pressure to start coding becomes worse in an OOSE course.
* It is hard to find the time to cover OO testing in a first course.
* Java promises to pull together domains that seemed to be drifting apart: data bases, operating systems, embedded systems and graphical user interfaces. However some of Java's class libraries are still in a state of flux. To program well in Java, you need to know the language (easy), you need to acquire an object-oriented style (difficult) and you

have to acquire knowledge about the philosophy and components in the class libraries (hard and time-consuming, respectively).

In order to use Java classes you have to understand how they fit together. Just how much of this knowledge can you fit into a SE course that has a host of other demands? Does this mean Java library specifics need to be more covered in previous courses? Java will also play a key part in SE education because it's free and it is widely distributed. These factors should not be underestimated in a world of spiraling education costs and dwindling university resources. Especially for poorer countries, where a simple PC may cost a "privileged" university professor more than a month's salary, free, state of the art software may play an important part in helping their educational systems prepare the competent information and computer professionals necessary to belong to a better world.

Ariel Ortiz position

Languages such as Scheme, Prolog, Logo and Smalltalk may be taught to Computer Science students using a translator-based approach with Java as a metalanguage. During 1999, I started using this approach to teach the undergraduate Programming Languages course at the ITESM CEM. Simple lexical analysis can be taught using predefined Java classes, such as Stream Tokenizer. More advanced lexical and syntactical analysis may require tools such as JavaCC, which allow the construction of complete parsers. JavaCC is much easier to learn and use than the traditional Lex and YACC tools, and of course, the generated source is Java, not C. The classes found in the Java class library allow students to develop their translators faster and with fewer bugs, thus permitting the instructor to request more interesting and complex projects. My students are required to develop compilers that generate assembly language output for the Java Virtual Machine. Afterwards, the Jasmin assembler tool is used to produce a portable.class file that may be executed by the Java Virtual Machine of any platform. When using this approach, my pupils are able to conclude that it is possible to obtain the benefits of the Java platform without necessarily having to use the Java programming language.

Wendy Doube position

The panel topic is related to the broader issue of integrating OO strategies into CS Ed. We use UML as a tool to provide comprehensive representational support for most stages of OOD. However some activities in detailed design still rely on structured techniques with representations such as pseudocode. We have developed some representations which integrate UML and OO programming structures into traditional structured detailed design representations. We then present these representations in detailed design strategies which complement OO development approaches.