IMPLEMENTATION

Prime Your Primary Weapon: Introducing IDEs

IDEs (Integrated Development Environments)
Professional software engineers rarely use notepad-like primitive editors. They use more sophisticated tools called Integrated Development Environments (IDEs). The aim of an IDE is to support all development-related work within the same tool environment.

An IDE generally consists of:

- A source code editor that includes features such as syntax coloring, auto-completion, easy code navigation, error highlighting, and code-snippet generation.
- A compiler and/or an interpreter (together with other build automation support) that facilitates the compilation/linking/running/deployment of a program.
- A debugger that allows program errors to be located.

In addition, IDEs may include features such as support for automated testing, drag-and-drop construction of UI components, version management support, and modeling support.

Eclipse, IntelliJ IDEA, NetBeans, Visual Studio, DevC++, DrJava and XCode are some examples of popular IDEs.

Note: Some experienced developers, in particular those with a UNIX background, prefer lightweight yet powerful text editors with scripting capabilities (e.g. Emacs http://www.gnu.org/software/emacs/) over heavier IDEs.

Debugging

Debugging is the process of discovering defects in the program. There are several approaches to debugging.

- **By inserting temporary print statements**: This is an ad-hoc approach in which print statements are inserted in the program to output variable values, as well as program states. While this approach is simple, it incurs extra effort when inserting and removing the print statements. Moreover, unnecessary program modification increases the risk of introducing errors into the program. These print statements, if not promptly removed, may even appear unexpectedly in the production version. Therefore, this approach is not recommended.

- **By manually tracing through the code**: Otherwise known as 'eye-balling', this approach has its merits. However, it is a difficult, time consuming, and error-prone technique. Indeed, the effectiveness of manual debugging is highly dependent on one's programming proficiency.

- **Using a debugger**: A debugger tool allows us to pause the execution, then step through one statement at a time while examining the internal state if necessary. Most IDEs come with an inbuilt debugger. This is the recommended approach for debugging.