Lecture 07 Software Tools and Environments

Programming environments

Simply stated, a programming environment is the collection of tools used in software development. It is a system which supports a activities in writing programs, such as editing, compiling, linking and debugging.
The usefulness of a programming environment for programmers largely depends on the type of software system being developed and the programming language applied.
In the following section we will look at two distinct types of programming environments: Command-line and Integrated Development Environments:

A. Command-line Environments

A command-line environment is a collection of commands that can be typed in to edit files, compile source code, and run programs. The command-line programming environment enjoys the greatest degree of configurability as one can access any options that are available in the program unlike the IDE counterpart.
However, this control comes with an ease-of-use penalty and one has to put in a lot of effort to learn and practice.
Use of the command-line can be simplified when one makes use of scripts and batch files. The commonest example of this is Unix programming.

B. Integrated Development Environments (IDE)

An IDE is a software application that provides comprehensive facilities to the programmer for software development i.e. the programmer need not apply tools manually. For instance IDE's provide flexibility in that one can execute and debug a program during an editing session even when the program is incomplete.
Together with a user interface, an IDE normally consists of:

- a text or source code editor
- a compiler and/or an interpreter
- build automation tools
- a debugger
- GUI

IDEs are designed to **maximize programmer productivity** by providing tightly-knit components with similar user interfaces, however, because an IDE is by its very nature a complicated piece of software, this high productivity only occurs after a lengthy learning process.
They are usually dedicated to a specific programming language. However, there are some multiple-language IDEs in use, such as Microsoft Visual Studio, the open source cross-platform Eclipse and KDevelop, e.t.c. Others include Borland JBuilder, DreamWeaver.

Software Engineering Tools

Tools allow repetitive, well-defined actions to be automated, thus reducing the cognitive load on the software engineer. They provide automated or semi-automated support for the process and the methods. The engineer is then free to concentrate on the creative aspects of the process. Tools are often designed to support particular methods, reducing any administrative load associated with applying the method manually. Like methods, they are intended to make development more systematic, and they vary in scope from supporting individual tasks to encompassing the complete life cycle.
Software Engineering tools can be classified according to the different phases in a software process:

Software Requirements Tools

These are tools that are useful for dealing with software requirements. They have been classified into two categories: modeling and traceability tools.

- Requirements modeling tools. These tools are used for eliciting, analyzing, specifying, and validating software requirements
- Requirement traceability tools. These tools are especially important for complex software which requires a system for tracing the requirements to other work products. They are thus also relevant in other life cycle processes.

Software Design Tools

Software Design tools are tools used to create and check software designs. They are usually diagramming tools.

Software Construction Tools

These tools are used to produce and translate a high level program representation (such as source code) into machine readable code to enable machine execution.

- Editors. These tools are used for the creation and modification of programs, and possibly the documents associated with them. They can be general-purpose text or document editors, or they can be specialized for a target language.
- Compilers and code generators. Traditionally, compilers have been non-interactive translators of source code, but there has been a trend to integrate compilers and program editors to provide integrated programming environments.
- Interpreters. These tools provide software execution through emulation. They can support software construction activities by providing a more controllable and observable environment for program execution.
- Debuggers. These tools are considered a separate category since they support the software construction process, but they are different from program editors and compilers.

Software Testing Tools

These tools are important in the software testing phase and also the maintenance phase as they enable testing of products by; generation, execution, evaluation or management of tests.

- Test generators. These tools assist in the development of test cases.
- Test execution frameworks. These tools enable the execution of test cases in a controlled environment where the behavior of the object under test is observed.
- Test evaluation tools. These tools support the assessment of the results of test execution, helping to determine whether or not the observed behavior conforms to the expected behavior.
- Test management tools. These tools provide support for all aspects of the software testing process.
- Performance analysis tools. These tools are used for measuring and analyzing software performance, which is a specialized form of testing where the goal is to assess performance behavior rather than functional behavior (correctness).

Software Configuration Management Tools

Tools for configuration management have been divided into three categories: tracking, version management, and release tools.

- Defect, enhancement, issue, and problem-tracking tools. These tools are used in connection with the problem-tracking issues associated with a particular software product.
- Version management tools. These tools are involved in the management of multiple versions of a product.
- Release and build tools. These tools are used to manage the tasks of software release and build. The category includes installation tools which have become widely used for configuring the installation of software products.

Tool Integration Mechanisms

Tool integration is important for making individual tools co-operate. Typical kinds of tool integration are presentation, process, data, and control.

- Presentation integration: The goal of presentation integration is to improve the efficiency and effectiveness of the user's interaction with the environment by reducing his cognitive load.
- Data integration: The goal of data integration is to ensure that all the information in the environment is managed as a consistent whole, regardless of how parts of it are operated on and transformed.

  The approach seeks to store all process artifacts in a repository – a common data representation for artifacts that different tools can use to communicate with each other.
- Control integration: The goal of control integration is to allow the flexible combination of an environment's functions, according to project preferences and driven by the underlying processes the environment supports.
- Process integration: The goal of process integration is to ensure that tools interact effectively in support of a defined process.

CASE Tools

When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called *computer-aided software engineering*, is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

Examples of CASE tools include:

- Graphical editors for system model development
- Data dictionary to manage design entities
- Graphical UI builder for user interface construction
- Debuggers to support program fault finding
- Automated translators to generate new versions of a program

Case classification helps us understand the different types of CASE tools and their support for process activities from different perspectives:

- *Functional perspective:* Tools are classified according to their specific function (*See Table1*)
- *Process perspective:* Tools are classified according to process activities that are supported (*See Figure 1*)
- *Integration perspective:* Tools are classified according to their organisation into integrated units

Table 1: Functional Classification of CASE Tools [*Sommerville 8th Ed*]

Tool type	**Examples**
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program re-structuring systems

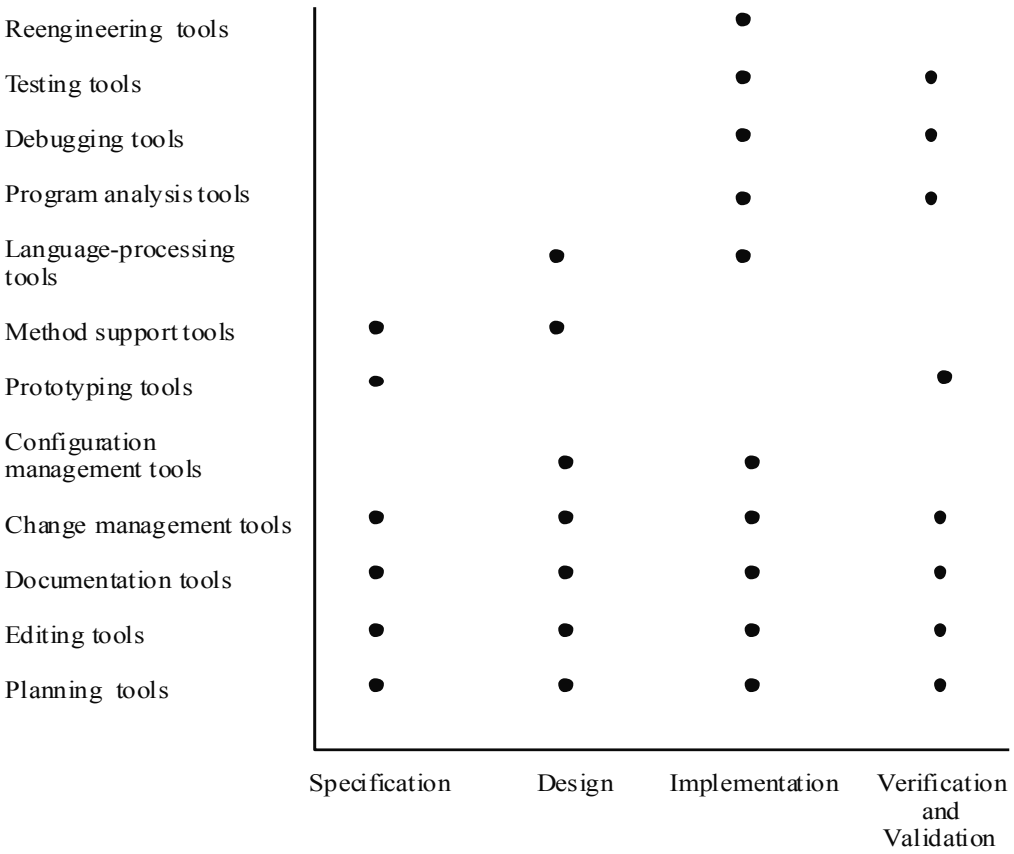

Figure 1: Activity based classification of CASE Tools

Further Reading

Sommerville, I., *Software Engineering*, 8th ed., Pearson, 2006.
Pressman, Roger S., *Software Engineering-A Practitioner's Approach,* 5th ed., McGraw-Hill, 2001.