BCT 2308
SOFTWARE DEVELOPMENT TOOLS AND ENVIRONMENTS

CHAPTER 1

Introduction to Software Development Tools and Environments

Chapter Objectives

- By the end of this lesson, learners should be able to:
  - Define software development Concepts.
  - List software development tools.
  - Describe the evolution of development tools and environments.
  - Identify Software development phases

Introduction

- Software Engineering is about methods, tools and techniques used for developing software.

Introduction

- Software surrounds us everywhere in the industrialized nations in domestic appliances, communications systems, transportation systems and in businesses.
- Software comes in different shapes and sizes from the program in a mobile phone to the software to design a new automobile.

Software

- A generic term for computer programs including system programs which operate on computer itself, and applications programs, which control the particular task at hand.
- The accompanying documentation is also considered an essential part of the software, even though it does not involve actual programming.

Types of Software

- In categorizing software, we can distinguish two major types:
  - Systems software
  - Application software

Types of Software

- *System software is the software that acts as tools to help construct or support applications* software.
- Examples are operating systems, databases, networking software, compilers.

Types of Software

- *Applications software is software that helps perform some directly useful or enjoyable* task.
- Examples are games, the software for automatic teller machines (ATMs), the control software in an airplane, e-mail software, word processors, spreadsheets.

Types of Software

- Within the category of applications software, it can be useful to identify the following categories of software:
  - Games
  - Information systems.
  - Real-time systems
  - Embedded Systems
  - Office Software.
  - Office Software.
  - Scientific software.

Types of Software

- Information systems – systems that store and access large amounts of data, for example, an airline seat reservation system.

Types of Software

- Real-time systems – in which the computer must respond quickly, for example, the control software for a power station.

Types of Software

- Embedded systems – in which the computer plays a smallish role within a larger system, for example, the software in a telephone exchange or a mobile phone.
- Embedded systems are usually also real-time systems.

Types of Software

- Office software – word processors, spreadsheets, e-mail
- Scientific software – carrying out calculations, modeling, prediction, for example, weather forecasting.

Software Engineering

- It is generally recognized that there are big problems with developing software successfully.
- A number of ideas have been suggested for improving the situation.
- These methods and tools are collectively known as software engineering.

Main Ideas in Software Engineering

- Greater emphasis on carrying out all stages of development systematically.
- Computer assistance for software development – software tools.
- An emphasis on finding out exactly what the users of a system really want (requirements engineering and validation).

Main Ideas in Software Engineering

- Demonstrating an early version of a system to its customers (prototyping).
- Use of new, innovative programming languages.
- Greater emphasis on trying to ensure that software is free of errors (verification).
- Incremental development, where a project proceeds in small, manageable steps

Software Development

- The process concerned with specifying, designing, developing and maintaining software applications by applying technologies and practices from computer science, Computer Technology project management and other fields.

Software Development

- Also called software engineering, software application development.

Software Development Process

- This is the process used to develop computer software.
- It may be ad hoc process, devised by the team for one project, but the term often refers to a standardized, documented methodology, which has been used before on similar projects or one, which is used habitually within an organisation.

Creating Software Involves

- Recognizing the need for a program to solve a problem.
- Planning the program and selecting the tools to solve the problem. The tools may be hardware platforms, programming languages, databases browsers and development kits.
- Writing the program in the programming language of choice.

Creating Software Involves

- Translating the human readable source code into either machine-readable executable code. This is done by compilers, assemblers and interpreters.
- Testing the program to make sure it works.
- Documentation, deployment and delivery.

Software Development

- Constructing software is a challenging task, essentially because software is complex.
- The perceived problems in software development and the goals that software development seeks to achieve are:
  - Meeting users' needs.
  - Low cost of production.
  - High performance.
  - Portability.
  - Low cost of maintenance.
  - High reliability.
  - Delivery on time.

Tasks in Software Development

- Feasibility Study.
- Requirements Engineering.
- Design:
  - User Interface Design.
  - Architectural (Large-Scale)Design.
  - Detailed Design
  - Database Design.

Tasks in Software Development

- Programming.
- System Integration.
- Verification.
- Validation.

Tasks in Software Development

- Production.
- Maintenance.
- Documentation.
- Project Management

Feasibility Study

- Before anything else is done, a feasibility study establishes whether or not the project is to proceed.
- It may be that the system is unnecessary, too expensive or too risky.

Feasibility Study

- One approach to a feasibility study is to perform cost-benefit analysis.
- The cost of the proposed system is estimated, which may involve new hardware as well as software, and compared with the cost of likely savings.

Feasibility Study

- This comparison then determines whether the project goes ahead or not.

Requirements engineering (specification)

- At the start of a project, the developer finds out what the user (client or customer) wants the software to do and records the requirements as clearly as possible.
- The product of this stage is a requirements specification.

User interface design

- Most software has a graphical user interface, which must be carefully designed so that it is easy to use.

Architectural (large-scale) design

- A software system may be large and complex.
- It is sometimes too large to be written as one single program.
- The software must be constructed from modules or components.

Architectural (large-scale) design

- Architectural, or large-scale design breaks the overall system down into a number of simpler modules.
- The products of this activity are an architectural design and module specifications.

Detailed design

- The design of each module or component is carried out.
- The products are detailed designs of each module.

Database design

- Many systems use a database to store information.
- Designing the database is a whole subject in its own right and is not normally considered to be part of software engineering.

Programming (coding)

- The detailed designs are converted into instructions written in the programming language.
- There may be a choice of programming languages, from which one must be selected.
- The product is the code.

System integration

- The individual components of the software are combined together, which is sometimes called the build.
- The product is the complete system.

Verification

- This seeks to ensure that the software is reliable.
- Verification answers the question:
  - Are we building the product right?.

Verification

- A piece of software that meets its specification is of limited use if it crashes frequently.
- Verification is concerned with the developer's view – the internal implementation of the system.

Verification

- Two types of verification are:
  - Unit testing and
  - System testing.
- In unit testing, each module of the software is tested in isolation.

Verification

- The inputs to unit testing are:
  - 1. The unit specification
  - 2. The unit code
  - 3. A list of expected test results.

Verification

- The products of unit testing are the test results.
- Unit testing verifies that the behavior of the coding conforms to its unit specification.

Verification

- In system testing or integration testing, the modules are linked together and the complete system tested.
- The inputs to system testing are the system specification and the code for the complete system.

Verification

- The outcome of system testing is the completed, tested software, verifying that the system meets its specification.

Validation

- This seeks to ensure that the software meets its users' needs.
- Validation answers the question:
  - Are we building the right product?

Validation

- Validation is to do with the client's view of the system, the external view of the system.
- It is no use creating a piece of software that works perfectly (that is tested to perfection) if it doesn't do what its users want.

Validation

- An important example of a validation activity is acceptance testing.
- This happens at the end of the project when the software is deemed complete, is demonstrated to its client and accepted by them as satisfactory.

Validation

- The inputs to acceptance testing are the client and the apparently complete software.
- The products are either a sign-off document and an accepted system or a list of faults.
- The outcome is that the system complies with the requirements of the client or it does not.

Maintenance

- When the software is in use, sooner or later it will almost certainly need fixing or enhancing.
- Making these changes constitutes maintenance.

Maintenance

- Software maintenance often goes on for years after the software is first constructed.
- The product of this activity is the modified software.

Documentation

- Documentation is required for two types of people – users and the developers.
- Users need information about how to install the software, how to de-install the software and how to use it.

Documentation

- Even in the computer age, paper manuals are still welcome.
- For general purpose software, such as a word processor, a help system is often provided.

Documentation

- User documentation concentrates on the "what" (the external view) of the software, not the "how" (the internal workings).
- Developers need documentation in order to continue development and to carry out maintenance.

Documentation

- This typically comprises the specification, the architectural design, the detailed design, the code, annotation within the code (termed comments), test schedules, test results and the project plan.

Documentation

- The documentation is typically large and costly (in people's time) to produce.
- Also, because it is additional to the product itself, there is a tendency to ignore it or withhold on it.

Project management

- Someone needs to create and maintain plans, resolve problems, allocate work to people and check that it has been completed.

Software Development

- Software Development is a complex and often difficult process requiring the synthesis of many disciplines.

Software Development

- From modeling and design to code generation, project management, testing, deployment, change management and beyond, tools have become an essential part of managing that complexity.

Software Development Environment

- Software Development Environment refers to a complete integrated set of hardware and associated software tools that are used by the software project team to develop an application.
- It is a collection of related tools.

Software Development Tools and Environment

- The tools and environments all aim at automating some of the activities that are involved in software engineering.
- The generic term for this field of study is Com*puter Aided Software Engineering* (CASE).

Software Development Tools and Environment

- Sometimes, tools are strictly necessary to support a given activity.
- In other cases, they simply help in making some jobs easier or more efficient.
- E.g. programming languages and their compilers or interpreters are indispensable for executing programs. Without them, software would not exist.

Software Development Tools and Environments

- A simple word processor to edit the specifications is useful that may increase the productivity and reliability of the whole production process compared to manually documenting specifications using pen and paper.

Historical Evolution of Tools and Environments

- In the early years of computing, constructing software was essentially a matter of programming.
- Thus the programming language was the major tool used by software developers.
- Associated with it were other tools used to facilitate programming, mainly, the compiler or the assembler, depending on the programming language.

Historical Evolution of Tools and Environments

- When compiler was the only tool, the result of coding consisted of mainly punched cards.
- Later, with the advent of third-generation computer systems with time-sharing, interactive terminals and on-line disk storage, interactive editors supported a much easier way of creating and manipulating programs.

Historical Evolution of Tools and Environments

- The major breakthrough introduced by these tools was the capability of correcting errors online.
- Previously, even a simple typing error required deletion of the current card, re-punching a new one and resubmission of the entire card deck.

Historical Evolution of Tools and Environments

- On-line storage of programs allowed several versions of the same program to stored into files, thus giving rise to the possibility of computer-supported configuration management, because all software was directly under computer control.

Historical Evolution of Tools and Environments

- A major improvement in program editing has been obtained quite recently by recognizing that programs are not just pure text.
- Programs have structure, defined by the syntax of the programming language.

Historical Evolution of Tools and Environments

- This recognition led to syntax-directed editors, which are especially helpful to novice programmers, because they enforce syntactic correctness as the program is being edited.

Historical Evolution of Tools and Environments

- Currently, programs may be created by expanding predefined templates in accordance with the grammar of a programming language, and any program manipulation constrains the portions of program that may be legally replaced by others to those that match the same syntactic template.

Historical Evolution of Tools and Environments

- Another area receiving increasing attention is program verification and debugging.
- Symbolic debuggers, which aid in analyzing program execution states in terms of source code identifiers, are the most widely used tools in this area.

Historical Evolution of Tools and Environments

- Advanced testing support tools such as test case generators, and data and control-flow analyzers are tools for verifying formal program correctness, are now available.

Historical Evolution of Tools and Environments

- The more complex the programming activity became, the more the tools were developed to support it and the more sophisticated those tools became.
- The need for the integration of tools arose naturally, the better to cope with the activity of software construction, which is in itself an integrated set of several sub-activities.

Historical Evolution of Tools and Environments

- This led to the notion of an *integrated project support environments* (IPSE), as a collection of tools and related methods with which the software engineer interacts during the entire life cycle of a product and across several products.

Historical Evolution of Tools and Environments

- Modularization has been recognized as vital in the construction of large systems, even when the techniques to exploit it are not well understood.
- Earlier examples of modular software were integration of mathematical routines into scientific applications and the linking of assembly language routines to object code compiled from possibly high-level languages.

Historical Evolution of Tools and Environments

- The linker is the tool that makes this combination possible.
- Integration may be viewed as an internal quality of the environment.

Historical Evolution of Tools and Environments

- This means that different tools or tool fragments cooperate easily.
- For example, no complex format conversions are needed from that output of tool 1 to the input of tool 2, if tool 1 and tool 2 are used in sequence.

Historical Evolution of Tools and Environments

- Integration may also be viewed as an external quality that may be appreciated by the environment's users.
- For example, in the debugging activity, first, one needs a debugger to monitor program execution and to locate errors.

Historical Evolution of Tools and Environments

- Once the errors have been located, they must be removed by editing, recompiling, and re-linking the program are supported by separate, nonintegrated tools, the programmer has to quit the debugger before invoking the editor, then after correction, recompilation, and re-linking, debugging resumes from scratch.

Historical Evolution of Tools and Environments

- The process is painful when repeated several times.
- By contrast, if tools are integrated and can run concurrently, the editor can be invoked from the debugger, which, based on the kind and amount of editing, can cause the minimum amount of recompiling to take place.

Historical Evolution of Tools and Environments

- In such integrated environment, switching from inspection of the execution state to program editing and back could be done quite naturally and effectively.

The end of Chapter 1