BCT 2308 SOFTWARE DEVELOPMENT TOOLS AND ENVIRONMENTS

CHAPTER 2:

Classification of Software Development Tools and Environments

Chapter Objectives

- By the end of this chapter, the learner should be able to:
  - Distinguish various dimensions along which tools can be classified.
  - Describe the major trends in (collections of) software tools.
  - Describe the role of tools in the software development process.

Introduction

- Software development is generally supported by tools, ranging from tools supporting a single activity to integrated environments supporting a complete development process.

Introduction

- The demand for software grows faster than the increase in software development productivity and available manpower.
- The result is an ever-increasing shortage of personnel.

Introduction

- One of the most obvious routes to pursue is automation itself.
- Software developers may use the computer as a tool in the production of software.

Introduction

- In the past, all sorts of things were automated, save software development itself.
- Software developers have long been accustomed to employ the computer as a tool for the implementation of software.

Introduction

- To this end, programmers have a vast array of tools at their disposal, such as compilers, linkers and loaders.
- Also during testing, tools like test drivers and test harnesses have been used for a long time.

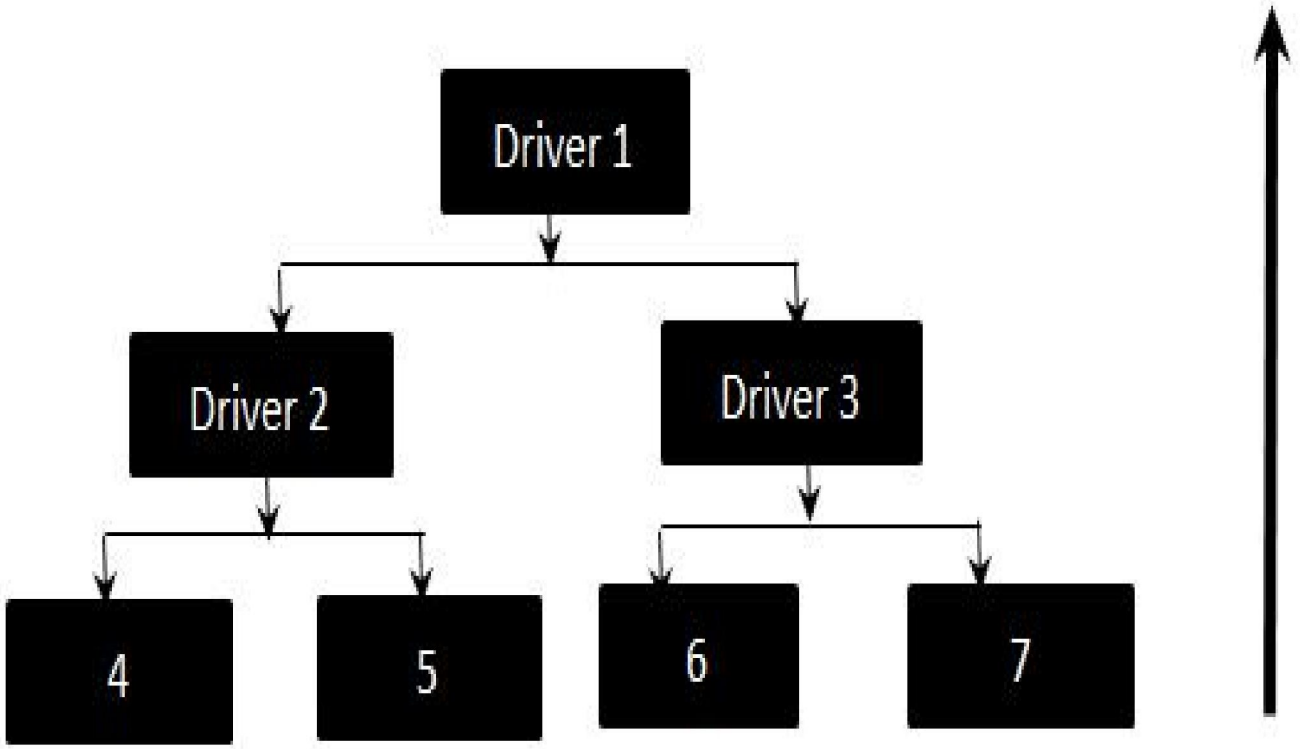
What is Test Driver?

- Test Drivers are used during Bottom-up integration testing in order to simulate the behaviour of the upper level modules that are not yet integrated.

Test Drivers

- Test Drivers are the modules that act as temporary replacement for a calling module and give the same output as that of the actual product.
- Drivers are also used when the software needs to interact with an external system and are usually complex than stubs.

Driver Flow Diagram



Test Drivers

- The above diagrams clearly states that Modules 4, 5, 6 and 7 are available for integration, whereas, above modules are still under development that cannot be integrated at this point of time.

Test Drivers

- Hence, drivers are used to test the modules.
- The order of Integration will be:
  - 4,2
  - 5,2
  - 6,3
  - 7,3
  - 2,1
  - 3,1

Test harness

- In software testing, a test harness or automated test framework is a collection of software and test data configured to test a program unit by running it under varying conditions and monitoring its behavior and outputs.
- It has two main parts:
  - The test execution engine and
  - The test script repository.

Test harness

- Test harnesses allow for the automation of tests.
- They can call functions with supplied parameters and print out and compare the results to the desired value.
- The test harness is a hook to the developed code, which can be tested using an automation framework.

Test harness

- A test harness should allow specific tests to run (this helps in optimizing), orchestrate a runtime environment, and provide a capability to analyze results.
- The typical objectives of a test harness are to:
  - Automate the testing process.
  - Execute test suites of test cases.
  - Generate associated test reports.

Test Cases

- Defined as a sequence of steps to test the correct behavior of a functionality/feature of an application.
- A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Test Cases

- A test case is a list of the conditions or issues of what the tester want to test in a software.
- Test case helps to come up with test data.
- A test case has an input description, Test sequence and an expected behavior.

Software Tools

- The use of software tools may have a positive effect on both the productivity of the people involved and the quality of the product being developed.
- Tools may support checking conformance to standards.

Software Tools

- Tools may help to quantify the degree of testing.
- Tools may support progress tracking.

Computer Aided Software Engineering -CASE

- The application of tools in the software development process is referred to as Computer Aided Software Engineering (CASE).
- Apart from the traditional implementation and test tools, CASE has a relatively short history.

CASE

- The first tools to support design activities appeared in the early 1980s.
- Today, the number of CASE products is overwhelming.
- As the number of available CASE products proliferates, it becomes expedient to classify them.

Classifying Tools

- Software tools can be compared and classified along many dimensions.
- When faced with the task of selecting a particular tool, one can use this list of criteria to consider the available alternatives.

Classifying Tools

- One way of doing so is according to the breadth of support they offer.
- Some products support a specific task in the software development process.
- Others support the entire software process.
- The former are called tools, the latter environments.

Criteria

- Interface format.
- Interaction Mode.
- Level of formality
- Dependency on phase of life cycle
- Dependency on application or method

Criteria

- Degree of standardization
- Programming Language dependency
- Static Versus dynamic tools
- Development tools versus End product component.

Interface Format

- Software engineers interact with computer tools in several ways.
- The goal of making the use of the computer as friendly and intuitive as possible is relevant and helps the software engineer feel at ease while using the environment.
- This reduces chance of error.

Interface Format

- Computer technology has evolved in the direction of enabling software tools to support increasingly sophisticated formats of HCI.
- The evolution of computer technology has allowed human-computer interfaces to become increasingly more friendly.

Interface Format

- Initially human-computer interaction was mainly textual.
- Later, the availability of low-cost graphics terminals and the introduction of pointing devices, such as the mouse, made it possible to provide graphical interfaces.

Interface Format

- Recent technologies such as multimedia interfaces and hypertext have the potential of making human-computer interaction even more sophisticated and productive.
- Multimedia interfaces, color, and hypertext make HCI even more sophisticated and productive.

Interface Format

- These new technologies are expected to have an impact on the structure of the tools that support cooperative group work during the entire life cycle.
- Text-oriented graphical tools each have their roles.

Interface Format

- Often, it is useful to support access to both notations for the same underlying model.
- Support for viewing a high-level design and selectively exploding the details of particular modules help in quickly browsing through large details.

Interaction Mode

- Tools may be categorized as:
  - Batch-oriented or
  - Interactive.

Interaction Medium

- Batch-oriented tools support the application of wholesale operations to a collection of documents.
- E.g. batch editing tools such as grep and sed in the Unix environment enable the programmer to search for all occurrences of a certain keyword or variable name in a set of files.

Interaction Medium

- This is usually much more convenient and less prone to error than using an interactive text editor for the same task.
- Interactive tools are more convenient in other situations where the immediate feedback from applying changes is helpful.

Interaction Medium

- E.g. in program debugging, it make useful to make a change and see the result of the change immediately.
- It is convenient when tools support both interactive and batch modes of operation.

Interaction Medium

- E.g. in debugging, sometimes we may want to find an instance when a particular value is assigned to a variable.
- In this case, interaction with a debugger is useful.
- Other times, we may want to find all occurrences of an assignment to a variable.

Level of formality

- Software development involves the production of several documents.
- Each document is written in some language whose syntax and semantics can be defined in a more or less formal way.
- In principle, suitable processors can be associated with any language.

Level of formality

- However, the functionality they may provide is highly dependent on the level of formality of the language.
- Typically, a compiler can be built only for a language whose syntax and semantics are defined formally, while an editor can be built for any language.

Level of formality

- Similarly, one may build a specialized graphical editor for data flow diagrams, UML since their composition rules (their syntax) are defined precisely.

Dependency on phase of life cycle

- Most software tools only help in some specific activity that is limited to a certain phase of the software life cycle.
- Therefore tools can be classified on the basis of the phase they are intended to support.

Dependency on the phase of life cycle

- E.g. there are tools for writing requirements specifications, tools for specifying module interfaces, tools for editing code, and tools for debugging

Dependency on phase of life cycle

- Other tools, such as text editors may be used in different activities.
- Some environments integrate tools, with the goal of supporting a natural and smooth transition through the software development phases, according to the selected model for software development life cycle.

Degree of standardization

- Standardizing an item, a life cycle method or a tool enhances its applicability and on the other hand freezes its evolution.
- The more a method or a language is standardized, the more software producers are willing to invest in developing support for it

Degree of standardization

- The stability of a method or language guarantees:
  - The longevity of its support tools, and this in turn, guarantees the return on investments.
  - A supply of people who are familiar with it and portability of their knowledge from one project to the next.

Degree of Standardization

- This is what happens if a development effort is based on an industry wide operating system or development methodology.
- Using a proprietary operating system reduces the supply of candidates and adds overhead for training.

Degree of standardization

- In practice, standardization is a matter of degree.
- Most developers add extensions to a base-level standard.

Programming Language dependency

- Some tools are associated with a particular programming language.
- These tools are called *monolingual ;*a compiler is an example.
- Other tools are language independent and are called *polylingual*.

Programming Language dependency

- E.g. conventional text editor or a word processor can be used as an editor for any programming or specification language, but there are specialized graphical editors for UML diagrams, or DFDs.

Programming Language dependency

- An operating system like UNIX, which provides a large variety of tools in support of software development in different programming languages.
- Similarly, conventional linkers are polylingual, since they support the linking of object modules derived from compilation of programs written in different source languages.

Programming Language dependency

- However, there are linkers that are specialized to particular programming languages such as Ada or Modula-2.
- The Java Development Kit (JDK) is a monolingual tool kit.

Programming Language dependency

- Generally, monolingual tools provide more specialized help and support, but a polylingual environment is more general and open.
- A user's experience from a polylingual environment is more portable.

Static Vs Dynamic Tools

- Some tools neither perform nor require execution of the object they operate on, be it a program or a specification document.
- Static tools neither perform nor require execution of the object they operate on.

Static Vs Dynamic Tools

- Static tools are applied to such an object to:
  - Create it.
  - Modify it.
  - Verify its consistency with respect to some rule, or even measure some static properties such as length, or detect the presence of certain constructs.

Static Vs Dynamic Tools

- An example of a static tool is a parser of a programming language, which checks the syntactic correctness of a program.
- Also, a type checker is static.
- Type checker for a typed language is a tool that evaluates whether the variables appearing in a program are manipulated consistently with their declared type.

Static Vs Dynamic Tools

- Dynamic tools require execution of the object e.g. Interpreters.
- A Perl interpreter, a statechart simulator are other dynamic tools.
- Static tools support the analysis of models and artifacts while dynamic tools support the simulation of models and artifacts.

Development Versus End-product components

- Development tools are tools that support the development of end products, but do not become part of them once the product is complete.
- When the product is complete nothing of the tools remains in the application that is released.

Development Versus End-product components

- Examples of development tools are:
  - Project management tools
  - Software specification simulators
  - Test case generators
  - Debuggers.

Development Versus End-product components

- End-Product component tools are kits of the software components that can be included in and become part of the final product.
- These components are usually provided as a run-time support library.

Development Versus End-product components

- Runtime is when a program is running (or being executed).
- That is, when you start a program running in a computer, it is runtime for that program.
- In some programming languages, certain reusable programs or "routines" are built and packaged as a "runtime library."
- These routines can be linked to and used by any program when it is running.

Development Versus End-product components

- Examples of end-product component tools are:
  - Window managers, which provide run-time libraries to support the human computer interaction.
  - Libraries of mathematical routines to be linked to an application and a set of macros for the development of specialized spreadsheets.

Dependency on application or method

- Application area affects the following:
  - Relevance of several software qualities,
  - The selection of appropriate design techniques and methods, and
  - More generally the choice of life cycle model.

Dependency on application or method

- For instance, structured Analysis and Design was mainly developed for conventional data-processing applications.
- For these applications, the friendliness of user interfaces is an important quality, because their expected end users have little or no computer background.

Dependency on application or method

- Some methods have their own associated, specifically designed, tools and thus affect the features provided by the tools.
- For example, stepwise refinement is best supported by syntax-directed editors.

Dependency on application or method

- Most software development methods can be made more effective by the availability of appropriate supporting tools.
- However, tools alone should not be counted on.
- Tools may enhance a method, but are not a substitute for the method.

Dependency on application or method

- Tools by themselves do not provide any magic solution.
- Software qualities do not improve by just using tools without attention to their meaning and purpose within the methodology.

Single user versus multiuser Tools

- Some tools support one user at a time, while others are capable of supporting multiple users at once.
- Single-user tools are used for personal activities, whereas multiuser tools may be used for groups.

Single user versus multiuser Tools

- Multiuser tools must support the locking of objects to prevent inadvertent interference among multiple users.
- As the use of networking has increased, with personal computers now shared on networks, even tools on personal computers can be multiuser.

Workbenches and Toolkits

- In between tools and environments it is useful to identify CASE products that support a limited set of activities, such as those which comprise the analysis and design stages.
- Such a coherent set of tools with a limited scope is referred to as a workbench.

Classifying Environments

- Environments can be further classified according to the mechanism that ties together the individual tools that make up the environment.
- In a toolkit, tools are generally not well integrated.

Classifying Environments

- The support offered is independent of a specific programming language or development paradigm.
- A tool kit merely offers a set of useful building blocks.

Classifying Environments

- A language-centered environment contains tools specifically suited for the support of software development in a specific programming language.
- Such an environment may be hand-crafted or generated from a grammatical description of the language.

Classifying Environments

- In the grammatical description of the language case, the environment tends to focus on the manipulation of program structures.
- The essence of integrated and process-centered environments is the sharing of information between the tools that make up the environment.

Classifying Environments

- Integrated environments focus on the resulting product.
- The heart of an integrated environment is a data repository, containing a wealth of information on the product to be developed, from requirements up to running code.

Classifying Environments

- Process-centered environments focus on sharing a description of the software development process.

Toolkits

- With a toolkit, developers are supported by a rather loosely-coupled collection of tools, each of which serves a specific, well-defined, task.
- The analogy with a carpenter – His/her toolkit contains hammers, screwdrivers, a saw, and the like.

Toolkits

- These tools each serve a specific task.
- However, they are not ‘integrated’ in the way a drill and its attachments are.
- The prime example of a toolkit environment is UNIX.

Toolkits

- UNIX may be viewed as a general support environment, not aimed at one specific programming language, development method, or process model.

Language-Centered Environments

- Nowadays, most software is developed interactively, changes are made interactively, and programs are tested and executed interactively.

Language Centered Environments

- Much research in the area of language-centered environments is aimed at developing a collection of useful, user-friendly, effective tools for this type of activity.

Language-Centered Environments

- Since most of these environments focus on supporting programming tasks, this type of environment is often called a programming environment.
- To emphasize their graphic capabilities to manipulate program constructs, they are sometimes called visual programming environments.

Language-Centered Environments

- Environments that are built around a specific programming language exploit the fact that a program entails more than a mere sequence of characters.

Language-Centered Environments

- Programs have a clear structure.
- This structure can be used to make the editing process more effective, to handle debugging in a structured way, and the like.

Language-Centered Environments

- Knowledge of properties of the objects to be manipulated can be built into the tools and subsequently used by these tools.
- Well-known early examples of language-centered environments are Interlisp and the Smalltalk-80 environment.

Language-Centered Environments

- Present-day language-centered environments generally come with a host of components that considerably ease software development.
- Examples of such environments include Microsoft Studio .NET and Eclipse.

Language-Centered Environments

- The support offered ranges from a set of API's for generating user interfaces (such as Swing), to facilities for handling persistence (EJB) or create web applications (Ajax).
- The richness of features comes with a price: a rather long learning curve.

Process-Centered Environments

- In a process-centered software engineering environment (PSEE), a description of the software development process is shared by the tools that make up the environment.

Process-Centered Environments

- Developments in process-centered environments are closely tied to developments in process modeling, and vice versa.
- For example, the kinds of description used in process modeling (state transition diagrams, Petri nets, and the like) are also the formalisms used in PSEEs.

The End

CAT 1:

Date: 30th October 2015

Time: 1 pm

Scope: Chapter 1 & 2