Computer-Aided Software Engineering

**MAJOR TOPICS**

- Objectives ................................................................. 76
- Pre-Test Questions ......................................................... 76
- Introduction ................................................................. 77
- Selecting a CASE Tool ................................................... 78
- Summary ................................................................. 81
- Post-Test Questions ....................................................... 81
OBJECTIVES
At the completion of this chapter, you will be able to:

- Identify the role of CASE tools in the software development process.
- Identify the criteria for selecting a CASE tool.
- Identify the benefits and limitations of CASE tools.
- Install and use Argo/UML.

PRE-TEST QUESTIONS
The answers to these questions are in Appendix A at the end of this manual.

1. What is Computer-Aided Software Engineering?
   ..................................................................................................................................
   ..................................................................................................................................

2. What are the criteria should one consider in a CASE tool?
   ..................................................................................................................................
   ..................................................................................................................................

INTRODUCTION

Computer-Aided Software Engineering (CASE) is the integration of software-based modeling tools into the software development process. Analysis and design methodologies and modeling notations were developed to formalize the software engineering process; CASE tools automate that process by assisting in each step. Some types of CASE tools are analysis and design tools, automated code generation tools, and software testing tools. Analysis and design tools aid in the creation of diagrams and project documentation. Automated code generation assists in the implementation phases. Testing tools lead to a more thorough evaluation of an application. Table 6-1 provides a chronological list of CASE tool development.

<table>
<thead>
<tr>
<th>Time Period</th>
<th>CASE Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early 1980's</td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>Analysis and design diagramming</td>
</tr>
<tr>
<td>Mid-1980's</td>
<td>Analysis and design validation</td>
</tr>
<tr>
<td>Late 1980's</td>
<td>Automated code generation</td>
</tr>
<tr>
<td>Early 1990's</td>
<td>User interface reusability</td>
</tr>
</tbody>
</table>

Early CASE tools were used to create project documentation and to assist in the creation of analysis and design diagrams. Later, CASE tools incorporated a type of intelligence that assisted in validating designs and ensuring conformity of diagrams. By the late 1980's, CASE tools were being used to automatically generate code, based on design diagrams. In the early 1990's, CASE tools had evolved into user-friendly interfaces that could be used on multiple projects, but still contained the aspects that had made previous CASE tools useful.
SELECTING A CASE TOOL

A wide variety of CASE tools exist to support different development environments. Choosing the best tool for a project can be a challenging exercise. The criteria used to select CASE tools include methodology, flexibility, collaboration, and diagram validation.

Methodology

Although a wide variety of software development methodologies exist, common themes exist between all object-oriented methodologies. Most variations occur in their processes, procedures, and notations. It is vital that the CASE tools selected support the methodology followed by the project.

When using UML and the Unified Process as a methodology for software development, a traditional process should be followed. This process consists of moving from high-level models, through abstract models, and finally to the full specification model. High-level models are general descriptions of the project, while abstract models include definitions of the resources required and ensure the soundness system design. Finally, the full specification model will completely describe the software and provide explicit directives for the software design. Any CASE tool that is under consideration, must support this methodology.

Flexibility

It is impossible to foresee every project need before development has begun. For this reason, choose tools that allow customization. The tool should allow custom documentation and should provide support for multiple programming languages. The best tools allow construction of custom schemas that can be used to generate code in several programming languages. Even with a detailed methodology, unforeseen problems and needs may arise throughout software development. Thus, the CASE tool cannot force users to use a fixed design.
Common CASE tool flexibilities include the ability to modify the target operating system, language used, modeling language, size of the software, and the entire process. Developers should not have to change the entire design of the software simply because specific parts of the CASE tool must be modified. Experienced development teams are flexible enough to deal with unforeseen problems and changes. The tools used by these developers should be just as flexible.

**Collaboration**

One of the principal purposes of any software development methodology is to facilitate collaboration. It is vital that CASE tools allow collaboration among developers; multiple developers should be able to work on the project simultaneously. The tool should also support distribution across multiple computers and multiple work areas.

Configuration management and version control software is critical to the development process. It allows multiple developers to work on the same code simultaneously. With it, engineers can manage multiple versions of a software segment, which targets multiple platforms or customers. Engineers can also control the status of code between versions and from one iteration to the next.

There are two parts to collaboration, both of which can heavily influence the selection of the CASE tool. The first is the prerequisites of the design. The second is the knowledge of the organization producing the code.

Sometimes the programmers on a team may have different visions of what the project should accomplish. Having common project prerequisites will allow cleaner collaboration within the team.

The actual organization that will be producing the code is also an important consideration when choosing a CASE tool. The development team can range in maturity, experience, and knowledge. These factors can heavily influence the CASE tool needed. A highly evolved CASE tool might be too intimidating for some inexperienced teams, while a simpler CASE tool could limit the efforts of a very experienced software organization.
Diagram validation

Although diagrams are used to simplify the design of a complex system, diagrams themselves can become fairly complex. Tools are available to validate analysis and design diagrams. These tools help diagrams conform to the modeling syntax, and ensure consistency across sets of diagrams.

Exercise 6-1: Using Argo/ UML

Argo/UML, a UML CASE tool, supports use-case, class, state, activity and collaboration diagrams. Argo/UML is written in Java and can be used to generate Java source code, and it will be used to complete many of the exercises in this course. In this exercise, you will install and explore Argo/UML.

1. Download and install a Java2-compliant virtual machine software, such as the Sun Java2 Runtime Environment. You can download the Java2 Runtime Environment from the following URL:
3. If you are using Microsoft Windows, double-click the argouml070.jar file to launch Argo/UML. If you are using another system or you experience difficulty with this method, create a batch file or shell script to execute Argo/UML. The file should contain one line that uses the Java virtual machine to run the uci.uml.Main class. Be sure to include the argouml070.jar and xml4j.jar files in the class path. Use the following example as a guide:
   java -classpath %CLASSPATH%;argouml070.jar;xml4j.jar uci.uml.Main
   You may need to include the full paths to the Java interpreter and .jar files.
4. Execute the batch file or shell script you created in Step 3 to start using Argo/UM L.

5. Open the projects included on the supplemental disk, and browse the diagrams included in the projects.

**SUMMARY**

Computer-Aided Software Engineering (CASE) is the integration of software-based tools into the software development process. These tools began appearing in the early 1980's. Today, CASE tools are used by software developers to create project documentation, analysis and design diagrams, to perform analysis and design validation, and to create program code from design diagrams. The criteria for selecting a CASE tool include the methodology it supports, the flexibility it affords developers, the degree to which it facilitates collaboration between developers, and its ability to validate analysis and design.

**POST-TEST QUESTIONS**

The answers to these questions are in Appendix A at the end of this manual.

1. What were early 1980's CASE tools used for?

2. What were mid 1980's CASE tools used for?

3. What were late 1980's CASE tools used for?