What is CASE?

- CASE: Computer Aided Software Engineering
  (Compare CAD - Computer Aided Design)
- All phases of system development projects are included:
  - Analysis, Design, Project planning
  - Documentation
  - Coding
  - Testing
  - Version and configuration control
  - Management and verification/follow up
- IEEE’s definition of “Software Engineering”:
  "A systematic approach to developing, using, maintaining and liquidating systems"

What is an environment?

- Programming Environment
  - An environment for the implementation phase
    (Programming-in-the-Small)
  - Editing, compiling, debugging, testing
- Program Development Environment
  - An environment which (possibly) should support all phases:
    - Design of requirements specifications and functional specifications
    - Managing configurations, versions, product(s)
      (Programming-in-the-Large)
    - Support for project management and planning, etc.
      (Programming-in-the-Many)

Four main groups of programming environments

- Language oriented environments
- Structure oriented environments
- Tool box environments
- Method based environments (Analysis/Design)

Language oriented environments
(or single-language environments)

- Tools (sometimes including operating systems) especially adapted to using a certain language
- Often strong integration between tools

Examples of single-language environments:

- Medley (InterLisp) for the Lisp language
- Cedar for the Mesa/Cedar languages
- Smalltalk for Smalltalk
- Rational-Ada for Ada
- The Mathematica environment for the Mathematica language
- The ObjectMath Environment for mathematical models in ObjectMath/Mathematica
- C++ environments:
  (ObjectCenter, Borland C++, etc...)
Characteristic 1: Support for explorative program development, i.e. prototyping

- Incremental, interactive
- Strong connections between application program and development environment
- Often monolithic system in a single address space
- Strong integration between tools (Editor, debugger, compiler, cross reference tool)

Characteristic 2: Syntactic and Semantic information available for navigation tools (browsers)

- Examples:
  - Diana trees with semantic attributes in the Rational environment
  - The Medley Masterscope cross-reference database
  - Symbol table information
- Uses of browsers:
  - System maintenance
  - Changes in unknown software systems
  - Explorative program development

Characteristic 3: Programming-in-the-Large (only some systems)

- Most common high level languages lack support for programming-in-the-large
  - Solution: use a special description language for this purpose
  - The user specifies a System Model over the components of this system
- Examples of single-language environments which also give some support for programming-in-the-large:
  - Cedar: introduced the first system modelling language (which later was used in Apollo’s DSEE product)
  - Rational: Subsystems (Versions of sub-systems, checking modules in and out)
Usage of single language environments

- Previously:
  - Development of research prototypes
  - Development of strongly interactive systems
- Now also used industrially
  - **The Rational environment:**
    (Handles over 1 million lines of code)
    (Used by Nobel Tech (previously Philips) in Stockholm in projects with 100 programmers)

Structure-oriented program development environments

- These environment can be seen as a further development of integrated single-language environments
- Syntax-oriented editing
  (Incremental immediate checking of syntax and semantics)
- Generation of tools from formal specifications

Examples of structure oriented environments

- The Cornell Program Synthesizer (from Cornell Univ)
  => The Synthesizer Generator (from Cornell)
  => The Synthesizer Generator (from Grammatech Inc.)
- GANDALF, Aloe (Carnegie-Mellon Univ)
- DICE (Linköping Univ)
  (Distributed Incremental Compiling Environment)
- Pecan Field (1984 Brown Univ, Steve Reiss)
- Mjölnor/ORM (Lund Univ)
- PSG (Darmstadt Univ.)
- Mathematica (Wolfram Research, 1989)
Struct. Orient. Env: Generation of tools

- Formal specifications of languages and tools are centered around abstract syntax trees
- Generation of tools such as: structure editor, prettyprinter, etc. (Gandalf, DICE, Mjølner/ORM, etc...)
- Generation of static semantic checking (= type checking), code generation, etc.
  (From attribute grammar: Synthesizer generator)
  (From Object-oriented attribute grammars: Mjølner/ORM)

Usage of Structure-Oriented environments

- So far mostly used by computer scientists and by beginners in teaching programming (this concerns the structure editing aspect)
- Problems for industrial application:
  - Efficient implementation
  - Large databases are needed for integrated environments
  - Common interchangeable program representation
- Several of these problems are now overcome, e.g. through efficient object-oriented data bases
- Examples proving the large-scale industrial systems are possible: Rational Ada

Tool box environments

- Such environments are composed of a number of largely independent tools which are run under some standard operating system
  Example: compiler, editor, linker, debugger, version manager, cross reference analyzer, etc...
- The development environment has usually little control over how the tools are used:
The user has to keep track of many details manually, and try to avoid mistakes.

Examples of toolbox environments

- Some environments:
  - Unix toolbox (Make, SCCS, compilers, debuggers)
  - DEC VMS VAX-set (CMS, etc.)
  - PCTE (Portable Common Tool ENVIRONMENT, prototype)
- Somewhat more integrated:
  - SUN NSE (Network Software Environment
  - SUN TeamWare (Simplified NSE, used in this course)
  - Apollo DSEE (Domain Software Engineering Environment)
Current Properties of Toolbox environments

- Easy to add new tools
- Portable, adapted to standard operating systems
- Communication between tools through untyped ASCII files (as in Unix)
- No incremental handling
- Problems storing typed structured objects

Usage of Toolbox environments:

- Dominates industrial program development, because of:
  - Portability
  - Backwards compatibility
  - Many users /large volumes
- Toolbox environments are evolving in the direction of:
  - Better integration between tools, similar to structure- and language oriented environments
  - Generation of tools as in structure-oriented environments

Method based environments

- Supports a certain program development methodology
- Two classes of such environments:
  - Support for a certain development phase
  - Support managing the development process

Environments supporting a development methodology

- For example, Upper-CASE environments, supporting the development of:
  - Requirements specifications
  - Design specifications
  - Functional specifications
- Semi-formal methods, e.g:
  - Objectory, Excelerator, etc...
  - SADT, SDL, PSL/PSA, etc...
- Formal methods, specification languages:
  - Petri nets
  - State machines, automata
  - VDM
  - Refine
  - Anna
  - Z, Lotus, etc...
Support for development methods:
- Methods for specification, design, validation and verification, reuse, etc...
- Most existing such CASE-tools:
  - Graphical editors for drawing design schemata
  - Organizing designs as a hierarchy of abstraction levels
  - Perform some consistency checking
  - Derivation of cross reference information

Support for the development process
- Support for handling the product:
  - Procedures and standards for version- and configuration management
  - Tools for version-, configuration- and "release" management
- Managing the development process
  - Project planning and management
  - Scheduling work and people
  - Estimation of costs and resource usage
- Support for product maintenance
  - Different models:
    - Process model
    - Contracts model

Comments regarding method based environments
- Current isolated tools only solve parts of the problem
  - Better integration and connection to the software product being developed is needed
- Better understanding of the semantics (= precise meaning) of different methods is desirable
  - Currently, a large number of Ad-Hoc methods are methods are available
- Automatic generation of programs from specifications is desirable
- Method based environments currently have a weak theoretical foundation compared to language- and structure-oriented environments
  - Often a very simple-minded operational control flow
  - Managing activities instead of objects
  - (although several object-oriented method based environments, e.g. Objectory, have now appeared)