ENCE 688R Civil Information Systems

Software Design and Development

Mark Austin

E-mail: austin@isr.umd.edu

Department of Civil and Environmental Engineering, University of Maryland, College Park

Lecture Topics

Part 1: Problem Solving with Computers

- Orchestration of good design solutions.
- Strategies for problem solving and dealing with system complexity.

Part 2: Implementation

- High-level problem solving procedure.
- Writing and running the software code.
- Compiling and running the program.
- Languages that are both compiled and interpreted.

Part 3: Program Development in Java

- Flowchart for development of Java programs.
- Strengths and weaknesses of Java.
- Integrated development environments (IDEs)

Part 1. Problem Solving with Computers

Problem Solving with Computers

Orchestration of Good Design Solutions

Generally speaking, a good (system or software) design provides (MIT, 2002):

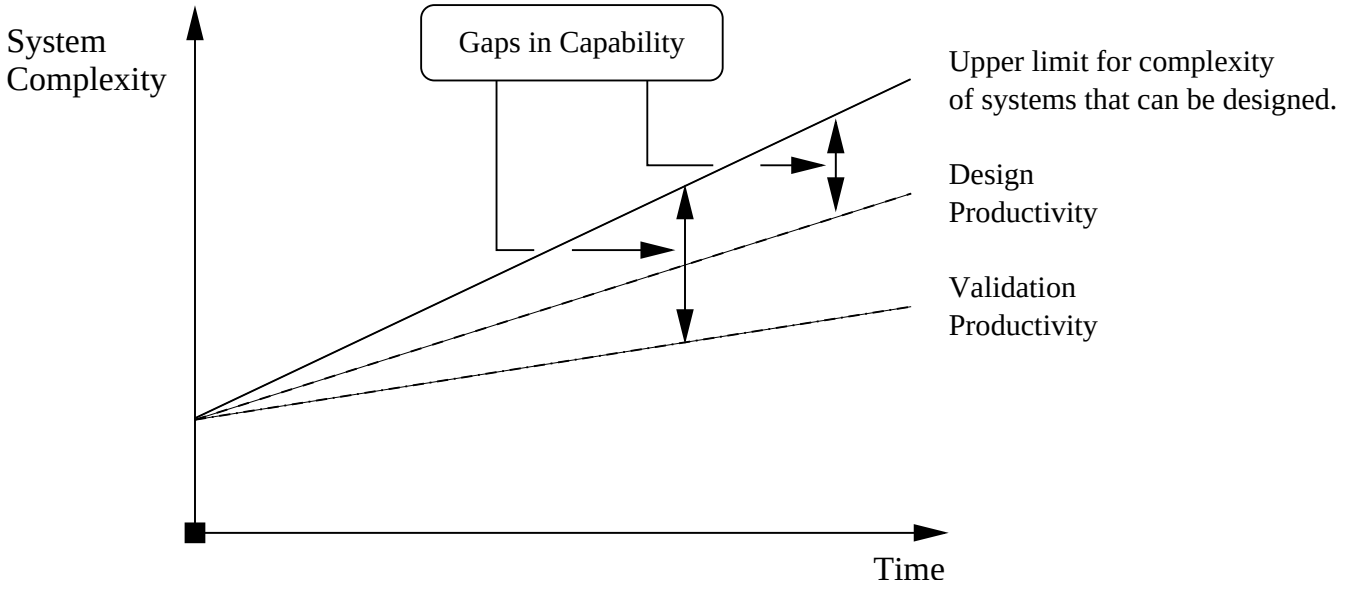
1. Bang for the buck – minimal mechanism; maximal function (i.e., a good, balance of functionality, performance and economics),
2. Reliable operation in a wide range of environments, and
3. Ease of accommodation for future technical improvements.

Problem Solving with Computers

Complexity of Systems and Software versus Time

Future engineering systems will be more complex than today. Designers will need to be more productive ...

... just to keep the duration and economics of design development in check.

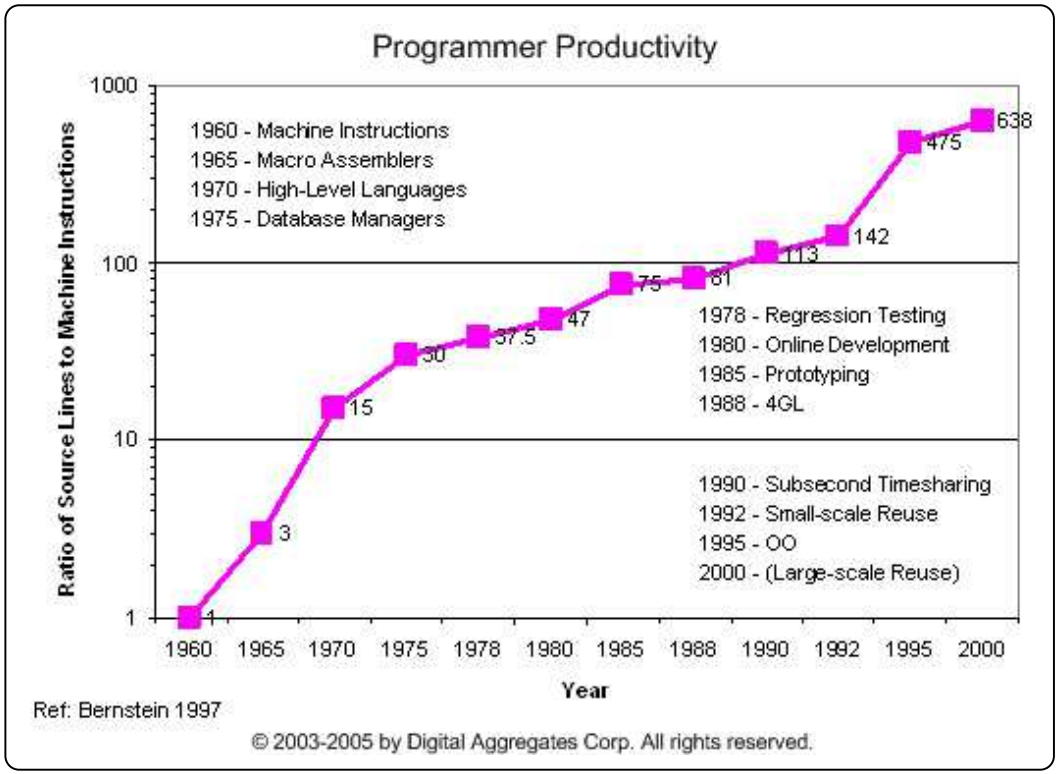


Problem Solving with Computers

Evolution of Abstractions in Software Development

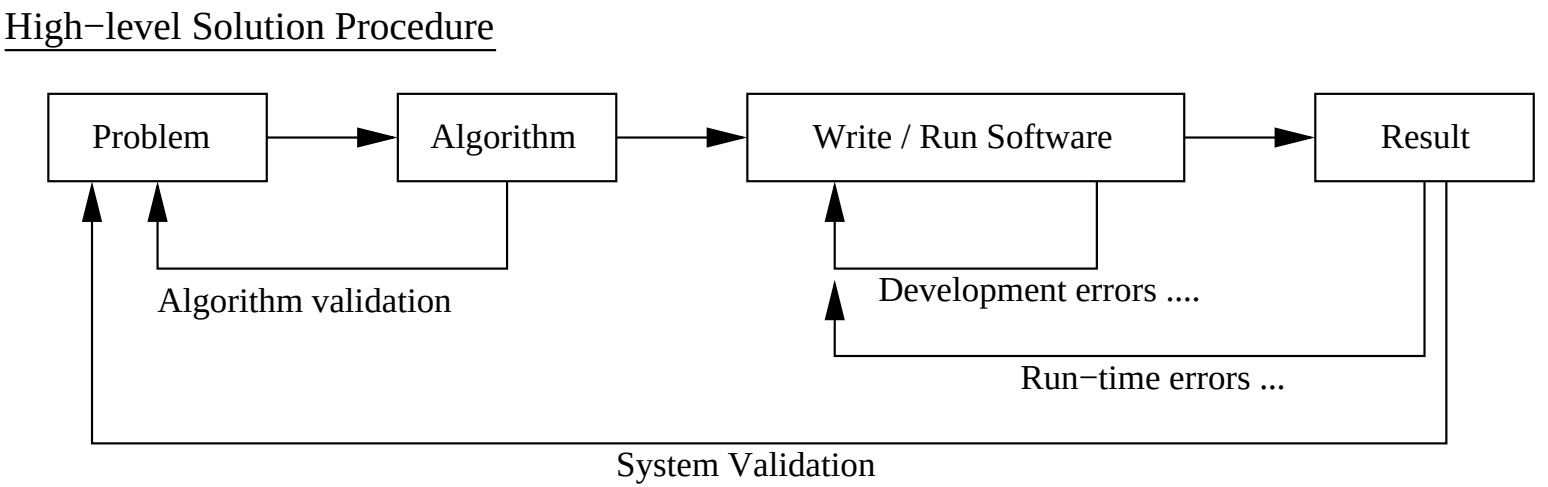
The pathway forward can be found by looking to the past, where ...

... major increases in designer productivity have nearly always been accompanied by new methods for solving problems at higher levels of abstraction.



Problem Solving with Computers

High-Level Problem Solving Procedure



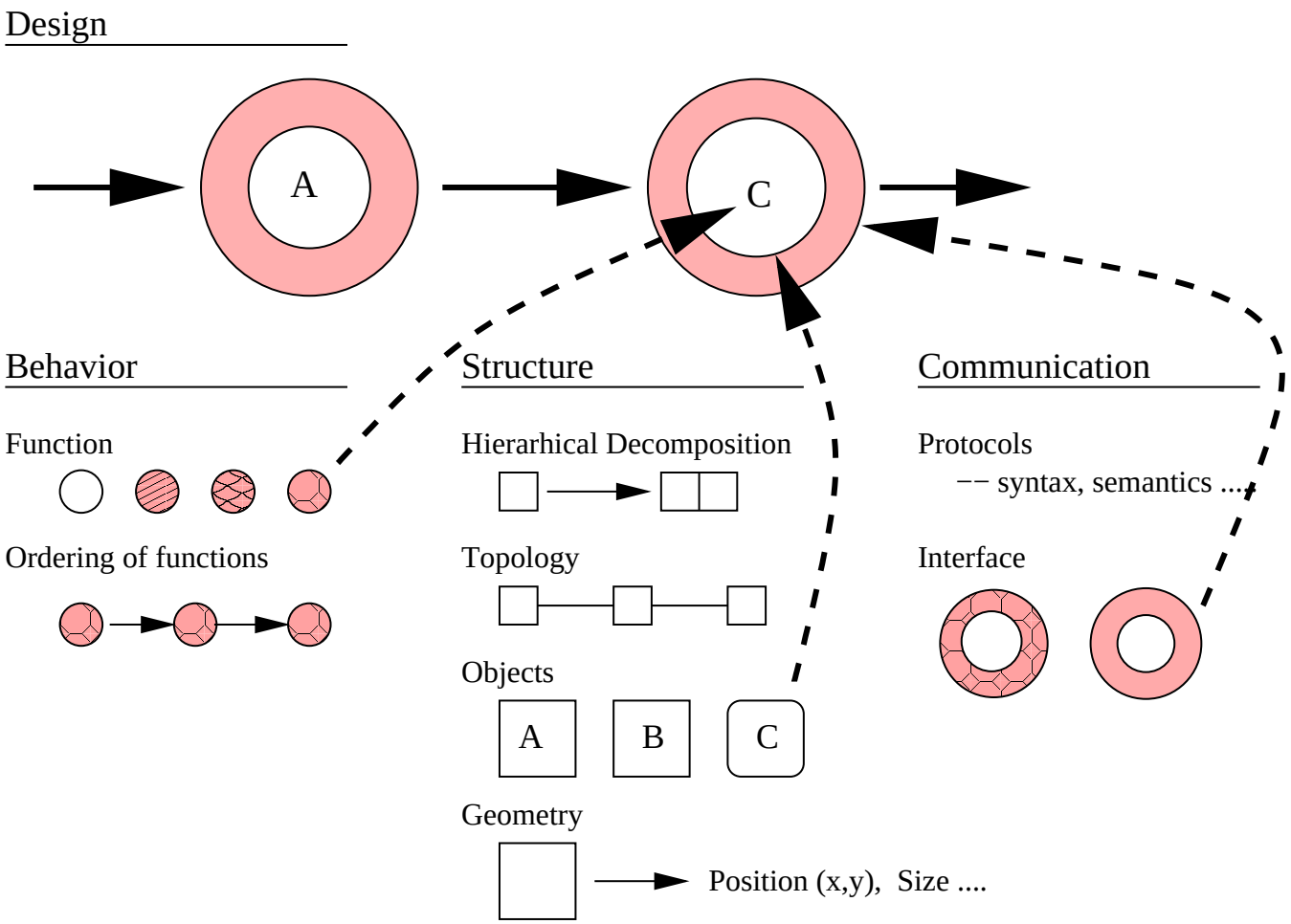
Computer programming is all about ...

... learning how to translate an algorithm into a set of instructions that a computer can understand.

Machine code, assembly language, high-level languages (e.g., Fortran), object-oriented programming (e.g., Java), scripting languages (e.g., Python).

Problem Solving with Computers

Simplify Software Design through Separation of Concerns



Problem Solving with Computers

Getting Started

1. **Develop Model of System Context**

   What is the context within which the system will operate?

2. **Operations Concept.**

   What is the required system functionality?

   What will the system do in response to external stimuli?

3. **Requirements.**

   What are the system inputs and outputs?

   What requirements are needed to ensure that the system will operate as planned?

Remark. Usecase diagrams are a good way of capturing fragments of required system functionality.

Problem Solving with Computers

Creating a Behavior Model...

1. **Identify top-level functionality**

   What are the top-level functions?

   Define inputs and outputs for each top-level functions.

   In what order will execution of the top-level functions occur?

   Trace inputs to outputs through network of connected functions.

2. **Identify sub-tasks within each top-level function**

   Goal is to simplify models of funtionality by decomposing high-level functions into networks of lower-level functionality.

3. **Identify opportunities for concurrent behaviors**

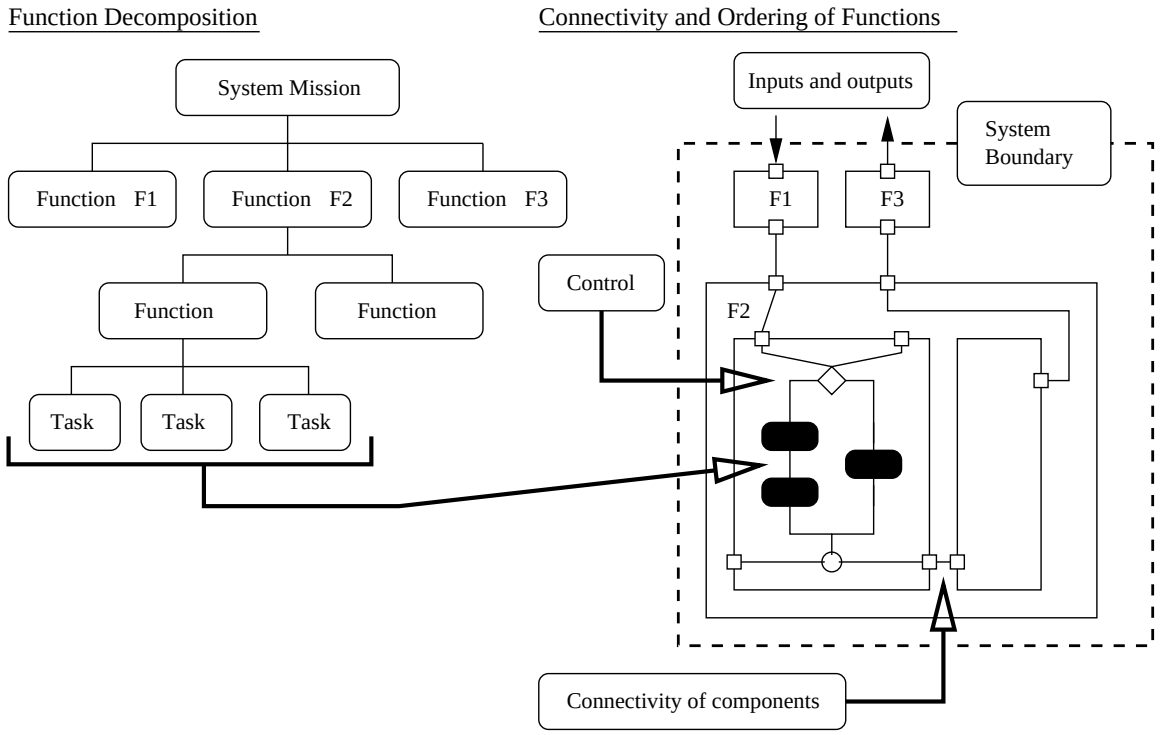
4. **Insert low-level functionalities**

Note. Several views of behavior may be required to obtain a complete picture of overall behavior.

Abstractions for Modeling System Behavior

Functional Decomposition

System behavior defined through decomposition and ordering (control) of functions.



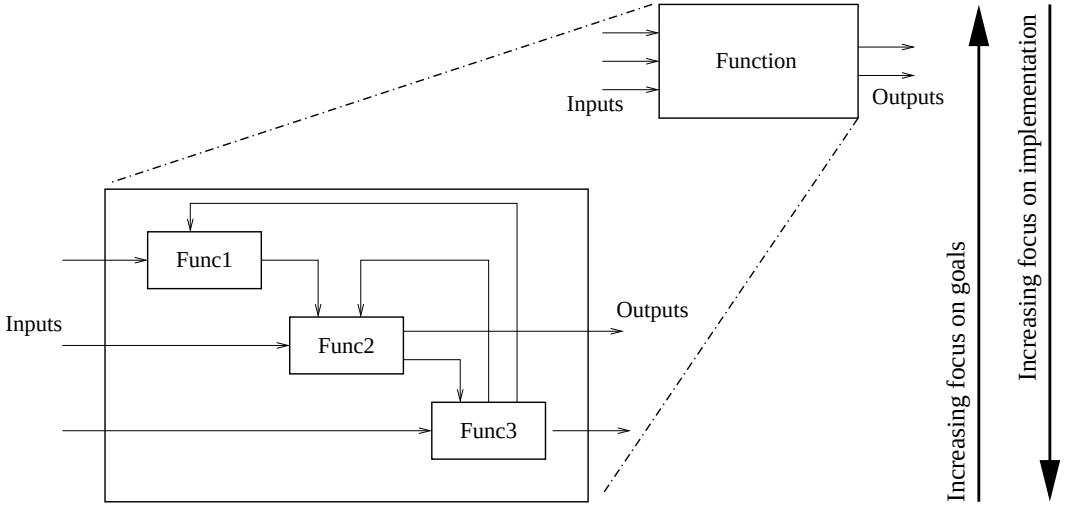
Note. The functional decomposition hierarchy says nothing about inputs and outputs.

Abstractions for Modeling System Behavior

Decomposition. Decomposition is the process of ...

... breaking the design at a given level of the hierarchy into components that can be designed and verified almost independently.

Decomposition of System Functionality



Note. Details of implementation are addressed in the lower levels of functional decomposition.

Bottom-Up Software Development

The strategy of bottom-up design ...

> **... starts with low-level procedures, modules, and subprogram library routines, and tries to combine them into higher-level entities.**

A key benefit of bottom-up design is its use of already implemented code.

For example, software libraries for,

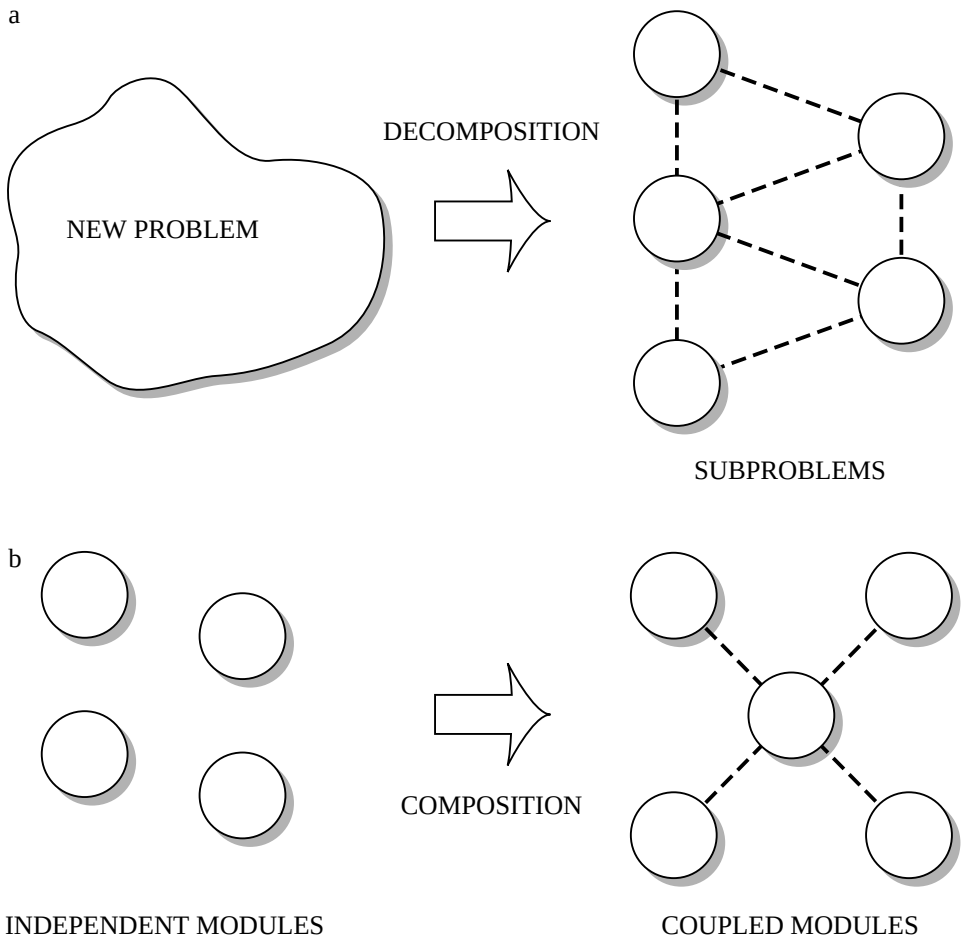
- Graphical user interface development.
- Numerical analysis.
- Distributed computing over networks.

In this class,

- Working with Java Collections.
- Network and graph-based engineering analysis.

Bottom-Up Software Development

Top-Down and Bottom-Up Design



Top-Down and Bottom-Up Development

Advantages/Disadvantages of Top-Down Decomposition

- Can customize a design to provide what is needed and no more.
- Decomposition simplifies development – lower-level (sub-system) development may only require input from a single discipline.
- Start from scratch implies slow time-to-market.

Advantages/Disadvantages of Bottom-up Development

- Reuse of components enables fast time-to-market.
- Reuse of components improves quality because components will have already been tested.
- Design may contain (many) features that are not needed.

Abstractions for Modeling System Behavior

Program Control $\rightarrow$ System Behavior

Behavior models coordinate a set of what we will call steps.

Such a specification requires that at least two questions be answered for each step:

1. When should each step be taken?
2. When are the inputs to each step determined?

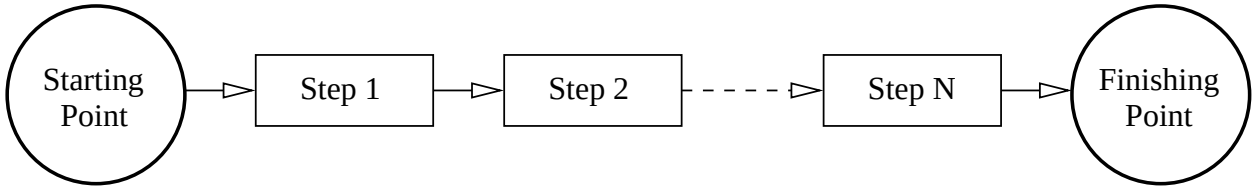
Abstractions that allow for the ordering of functions include:

- Sequence constructs,
- Branching constructs,
- Repetition/looping constructs,
- Concurrency constructs.

Abstractions for Modeling System Behavior

Sequencing of Steps in an Algorithm

Which functions must precede or succeed others?



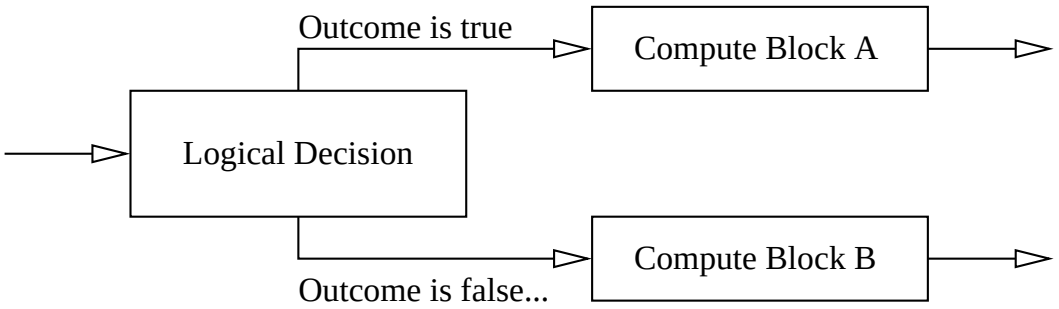
The textual/pseudocode counterpart is:

```
Starting Point
   Step 1.
   Step 2.
   Step 3.
   ......
   Step N.
Finishing Point
```

Abstractions for Modeling System Behavior

Selection Constructs

Capture choices between functions



Languages need to support decision making through ...

... the implementation of relational and logical expressions.

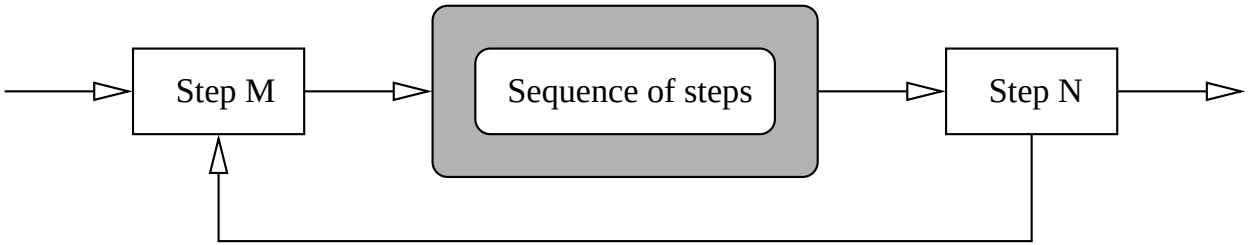
For example ...

```
Question: Is 4 greater than 3?
Expression: 4 > 3 ... evaluates to ... true.

Question: Is 4 equal to 3?
Expression: 4 == 3 ... evaluates to ... false.
```

Abstractions for Modeling System Behavior

Repetition/Looping Constructs



Repitition constructs want to know:

- Which functions can be repeated as a block?

Abstractions for Modeling System Behavior

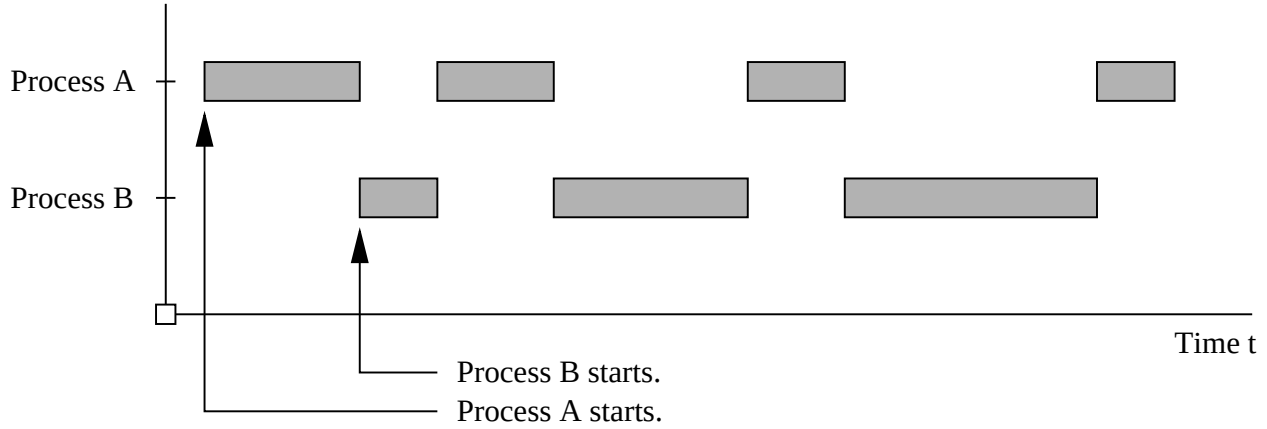
Ordering of Functions: Concurrency

Most real-world scenarios involve concurrent activities in one form or another.

The key challenge lies in the ...

> **... sequencing and coordination of activities to maximize a system's measures of effectiveness (e.g., production).**

Example 1. Running multiple threads of execution on one processor.



Implementation

Implementation

Writing the Program Source Code

When you write the source code for a computer program, all you are doing is ...

... using text to fill-in the details of programming templates.

While the basic problem solving strategy will be language-independent, the syntax details will vary from one language to another, e.g.,

```
Branching Construct in Java                Branching Construct in Matlab
=========================================================================

if ( i < 3 ) {                             if i < 3,
  .... do something ....                      .... do something ....
} else {                                   else
  .... do something else ....                 .... do something else ....
}                                          end;

=========================================================================
```

Implementation

Interpreted Programming Languages

In an **interpreted** computer program, ...

> **... high-level statements are read one by one, and translated and executed on the fly (i.e., as the program is running).**

Examples

- HTML and XML.
- Visual Basic and Javascript.

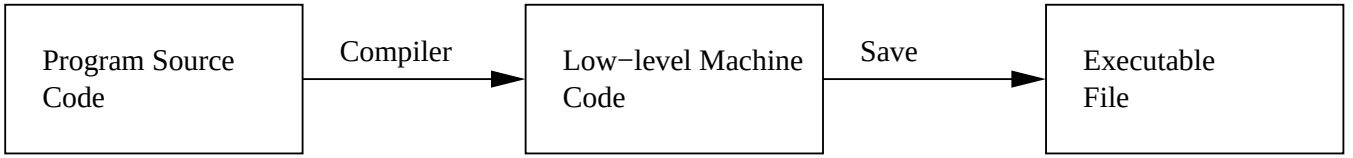
Scripting languages such as Tcl/Tk and Perl are interpreted, as are application programs written in the MATLAB programming language.

Implementation

Compiling the Program Source Code

A compiler translates the computer program source code into ...

... lower level (e.g., machine code) instructions.



For example, ...

... high-level programming constructs (e.g., evaluation of logical expressions, loops, and functions) are translated into equivalent low-level constructs that a machine can work with.

Examples. C and C++.

Implementation

Benefits of Compiled Code

- Compiled programs generally run faster than interpreted ones. This is because ...

  ... an interpreter must analyze each statement in the program each time it is executed and then perform the desired action,

  whereas the compiled code just performs the action within a fixed context determined by the compilation.

Benefits of Interpreted Code

- With an interpreted language you can do things that cannot be done in a compiled language. For example, interpreted programs can ...

  ... modify themselves by adding or changing functions at runtime.

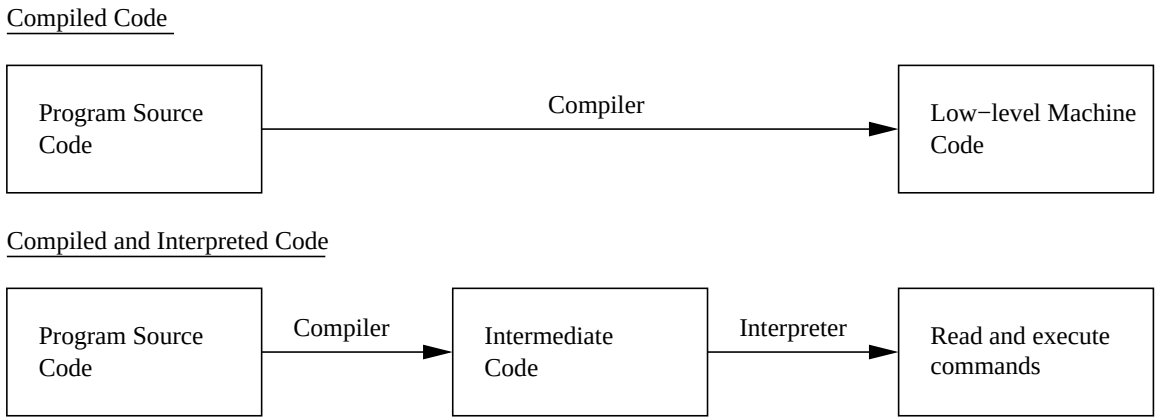
- Also, it is usually easier to develop applications in an interpreted environment because you don't have to recompile your application each time you want to test a small section.

Implementation

Code that is both Compiled and Interpreted

Most modern interpreter systems also perform some form of compilation – that is, ...

... they take the source code and transform it into a lower-level intermediate format. An interpreter then executes commands in the intermediate format.

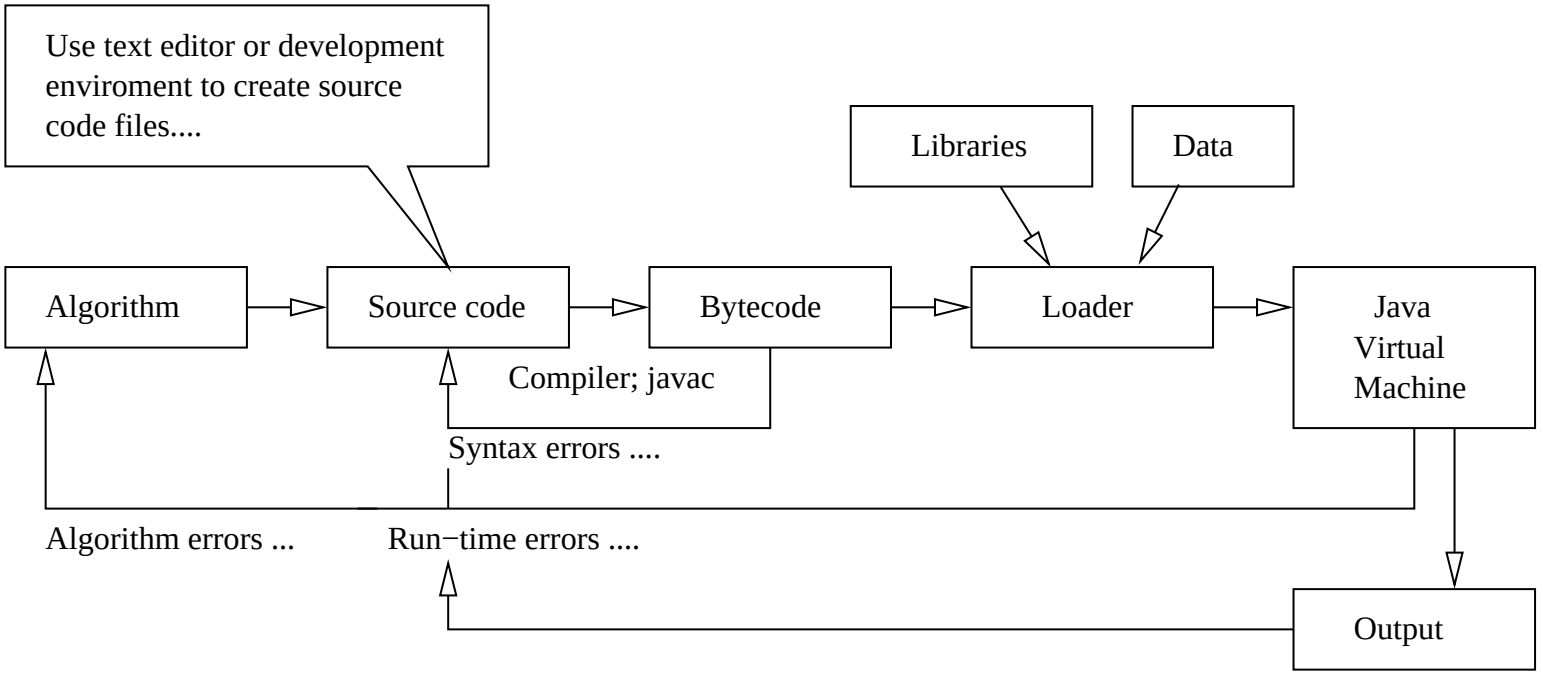


Examples. MATLAB, Java and Python.

Program Development with Java

Program Development with Java

Flowchart for Software Development in Java



Program Development with Java

Strengths of Java

1. Java is both a compiled and interpreted language. Java source code is **compiled** into a **bytecode format**.
2. Bytecodes are the lowest possible instruction format that remain **architecture neutral**. As a result, the bytecode can travel across the Internet and execute on any computer that has a Java Virtual Machine.
3. Java is an **object-oriented language**. Implementation details are made efficient by exploiting the **relationship among objects**.

Weaknesses of Java

1. There's a lot to learn, especially if you want to become really skilled at developing software in Java.

Integrated Development Environments for Java

Eclipse is an integrated software development tool (or IDE) for Java Software Development.