Engineering Software Development

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688R, Spring Semester 2023

February 6, 2023

Overview

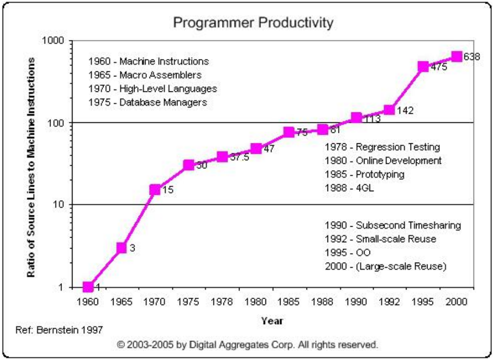
Part 1

Quick Review

Pathway to Improved Programmer Productivity

Pathway Forward

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



Evolution of Computer Languages

Computer Languages. Formal description – precise grammar – for how a problem can be solved.

Evolution. It takes about a decade for significant advances in computing to occur:

Capability	1970s	1980s	1990s
Users	Specialists	Individuals	Groups
Usage	Numerical computations	Desktop computing	E-mail, web, file transfer.
Interaction	Type at keyboard	Screen and mouse	audio/voice.
Languages	Fortran, C	MATLAB	HTML, Java

Popular Computer Languages

Tend to be designed for a specific set of purposes:

- FORTRAN (1950s – today). Stands for formula translation.
- C (early 1970s – today). New operating systems.
- C++ (early 1970s – today). Object-oriented version of C.
- MATLAB (mid 1980s – today). Stands for matrix laboratory.
- Python (early 1990s – today). A great scripting language.
- HTML (1990s – today). Layout of web-page content.
- Java (1994 – today). Object-Oriented language for network-based computing.
- XML (late 1990s – today). Description of data on the Web.

Problem Solving with Computers

Problem Solving with Computers

Develop Model of System Context:

- What is the context within which the software will operate?

Operations Concept:

- What is the required system functionality?
- What are the system inputs and outputs?
- What will the system do in response to external stimuli?

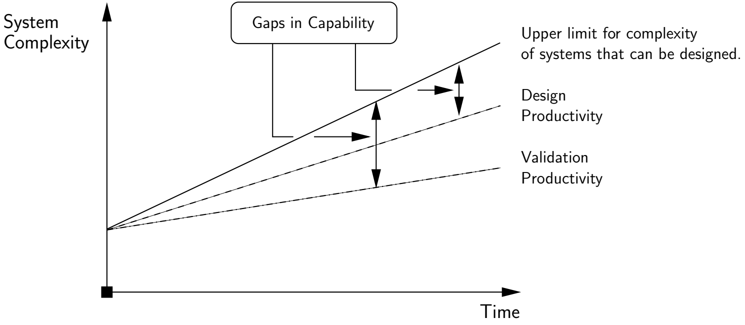
Requirements:

- What requirements are needed to ensure that the system will operate as planned?
- How will the software be written, tested, maintained?

Strategies for Handling Complexity

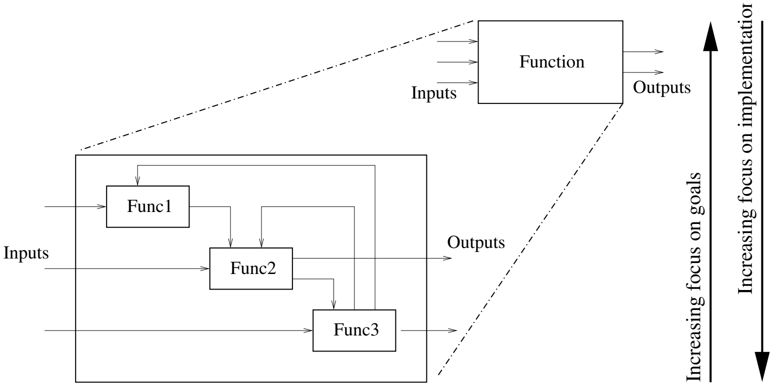
Productivity Concerns

System designers and software developers need to find ways of being more productive, just to keep the duration and economics of design development in check.



Strategies for Handling Complexity

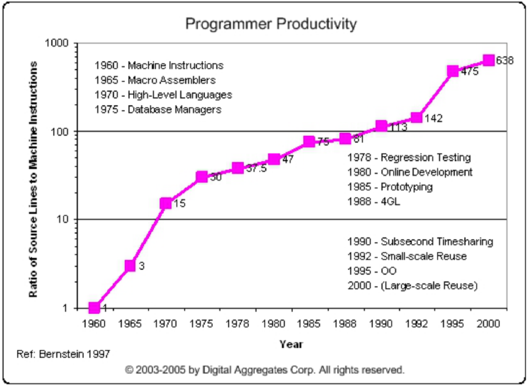
Simplify models of funtionality by decomposing high-level functions into networks of lower-level functionality:

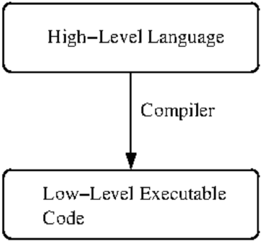


Strategies for Handling Complexity

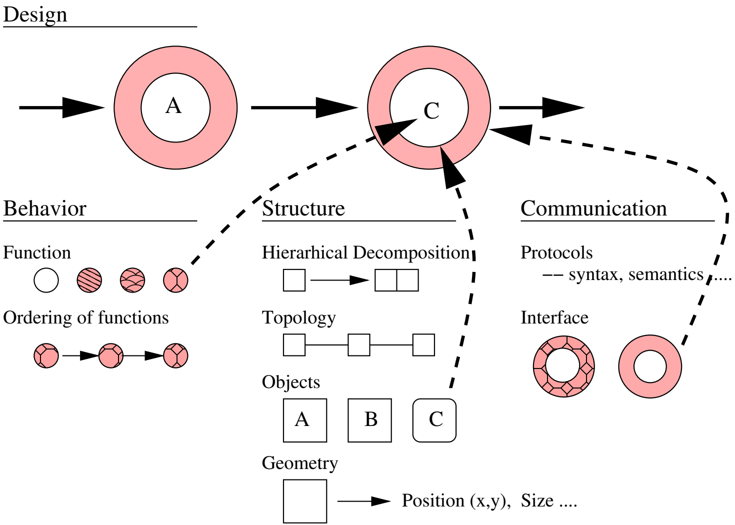
Create High-Level Description of Solution:

Increasing System Complexity: Software programmers need to find ways to solve problems at high levels of abstraction.





Separation of Concerns



Separation of Concerns

Models of System Structure:

- Specify how a system (including software) will solve a problem.
- Includes development of functional hierarchies and network structures.

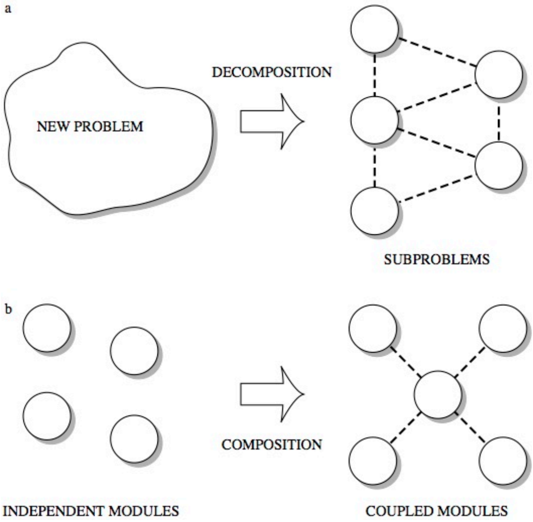
Models of System Behavior:

- Specify what the system (including software) will do.
- Includes top-level functionality, inputs and outputs, order of function execution.

Models of System Communication:

- Specification for how subsystems will communicate.
- Includes specification of interfaces and protocols for communication.

Top-Down and Bottom-Up Design



Top-Down and Bottom-Up Design

Top-Down Development:

- Can customize a design to provide what is needed and no more.
- Start from scratch implies slow time-to-market.

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.

This Class:

- Extensive use of software libraries (e.g., collections).

Modeling

System Behavior

Abstractions for Modeling System Behavior

Program Control → System Behavior:

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

Two questions for each step:

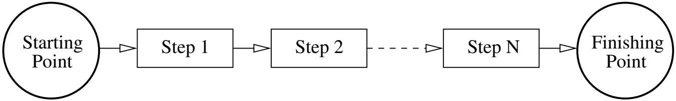
- When should each step be taken?
- When are the inputs to each step determined?

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?

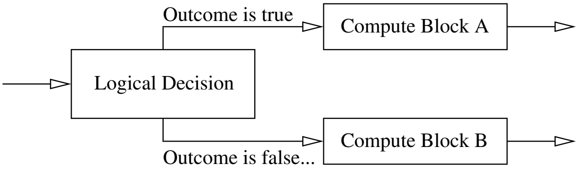


The textual/pseudocode counterpart is:

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

Abstractions for Modeling System Behavior

Selection Constructs: Capture choices between functions



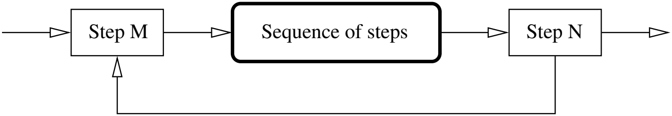
Languages need to support evaluation of relational and logical expressions.

```
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:

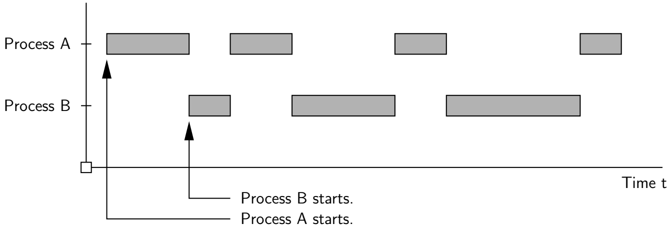
- Which functions can be repeated as a block?

Abstractions for Modeling System Behavior

Ordering of Functions: Concurrency

Most real-world scenarios involve concurrent activities. The key challenge is sequencing and coordination of activities to maximize a system's performance.

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



Interpreted and Compiled Languages

Interpreted Programming Languages

Interpreted Programming Languages:

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

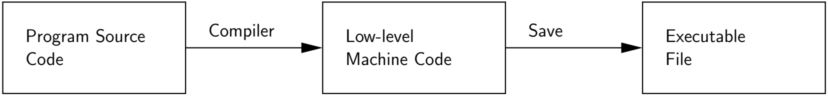
Examples:

- HTML and XML.
- Visual Basic and Javascript.

Scripting languages such as Tcl/Tk and Perl are interpreted. Python and Java are both interpreted and compiled.

Compiling the Program Source Code

A compiler translates the computer program source code into lower level (e.g., machine code) instructions.



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++.

Benefits of Compiled and Interpreted Code

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.

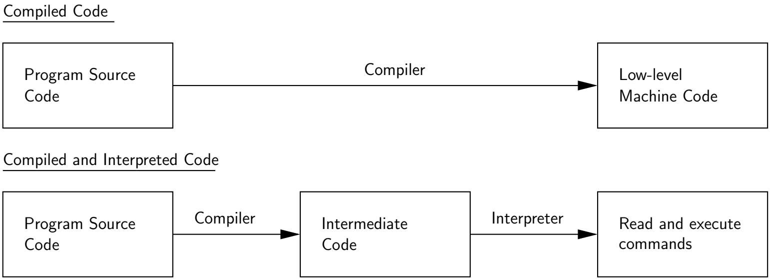
Benefits of Interpreted Code:

- Interpreted programs can modify themselves by adding or changing functions at runtime.
- Cycles of application development are usually faster than with compiled code because you don't have to recompile your application each time you want to test a small section.

Compiled and Interpreted

Modern Interpreter Systems

Transform source code into a lower-level intermediate format. Interpreter then executes commands.



Examples: MATLAB, Java and Python.