PC204

Lecture 5
Programming Methodologies

Programming Paradigms

- Software Engineering
- Exploratory Programming

Software Engineering

- Requirements
- Specification
- Design
- Coding
- Verification
- Debugging
- Documentation
- Dissemination
- Maintenance
- *Enhancement*

Why Doesn’t It Work (for us)?

- Fuzzy requirements
- The most important phase is often is least well defined, especially in a research environment

Exploratory Programming

- Faster feedback loop
- Standard *components*
- Reusable *components*
- Rapid Application Development (RAD)

Methodologies

- Functional decomposition
- Structured programming
- Modular programming
- Object-oriented programming
- Generic programming
- Extreme programming
- Agile programming

What's the Difference?

- Methodologies may be applied for any programming language
- Some languages are easier (or harder) to use with some methodologies
- The outward appearance of a program is frequently determined by the language, but the methodology may be discerned from code organization

Evaluation Criteria

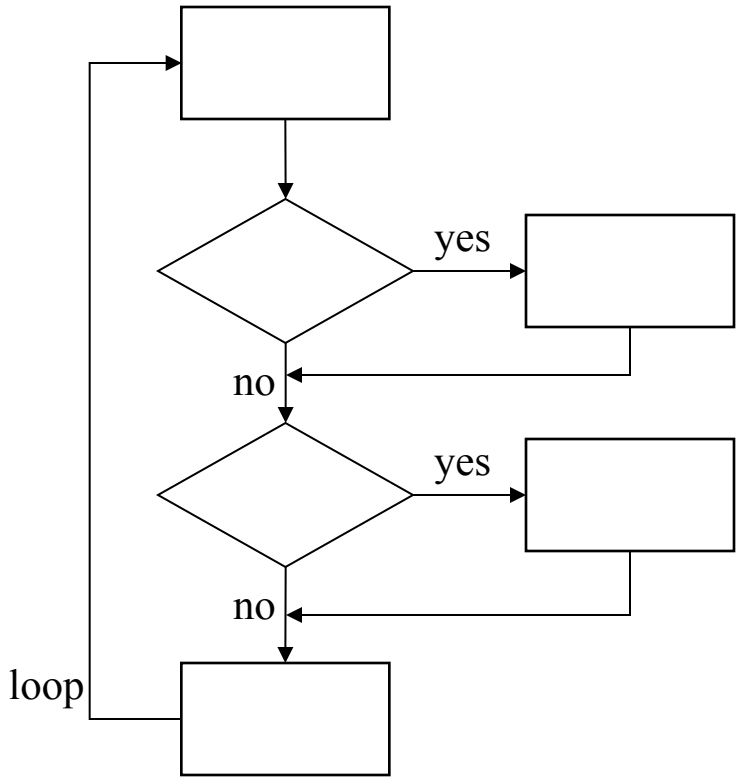
- Correctness
- Maintainability
- Flexibility
- Reusability

Functional Decomposition

- Divide problem into phases
- Flowchart diagrams
- “Input, compute, output”
- Algorithm for each phase
- Fortran

- Correctness - okay
- Maintainability - okay
- Flexibility - limited
- Reusability - limited

Functional Decomposition

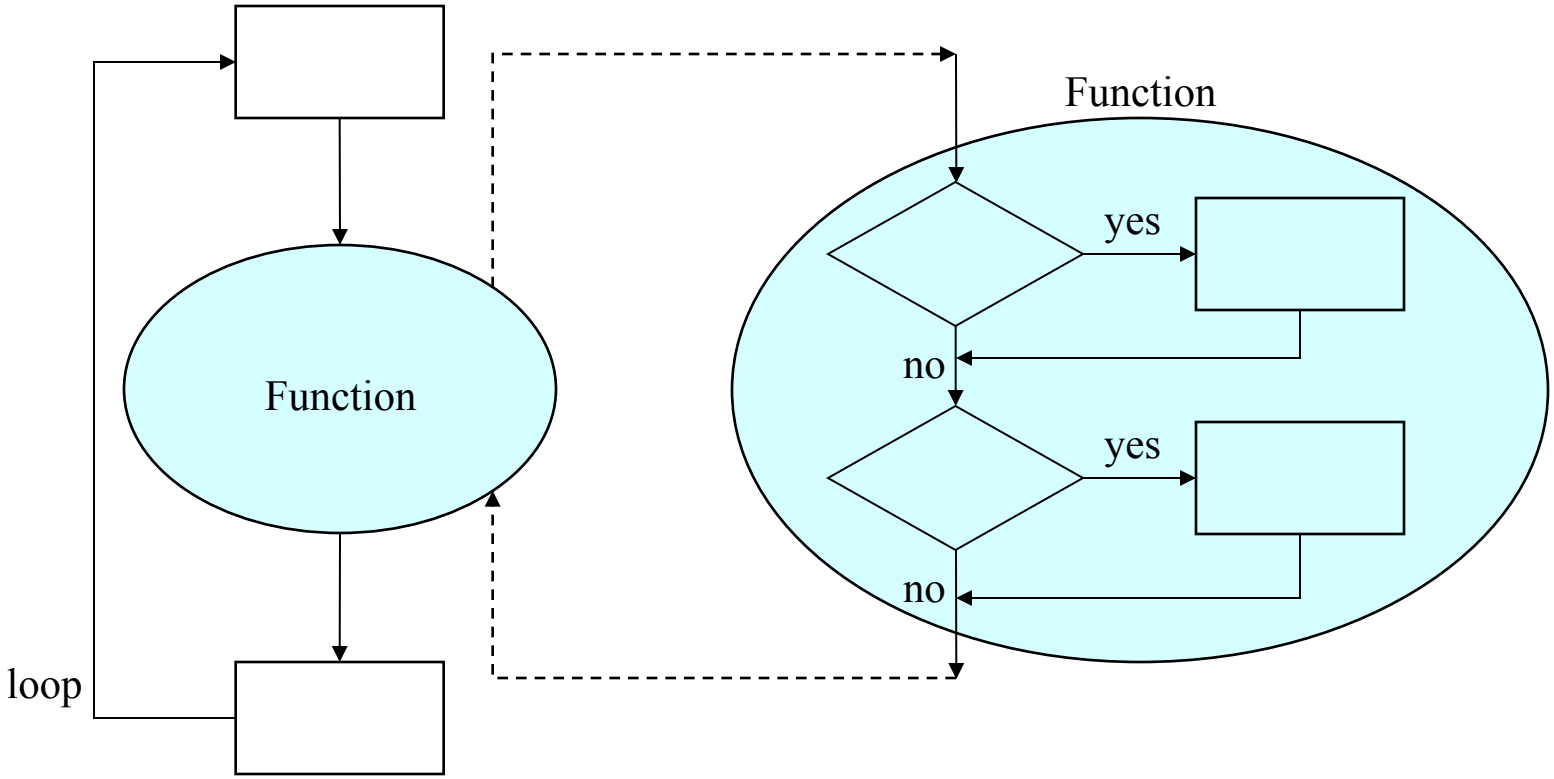


Structured Programming

- Local organization
- No “go to”s
- Use functions
- Characterized as “just indentations” by unbelievers
- Fortran, C, Pascal

- Correctness - okay
- Maintainability - better
  - more readable code
- Flexibility - okay
  - simpler to reorganize
- Reusability - better
  - reuse functions

Structured Programming

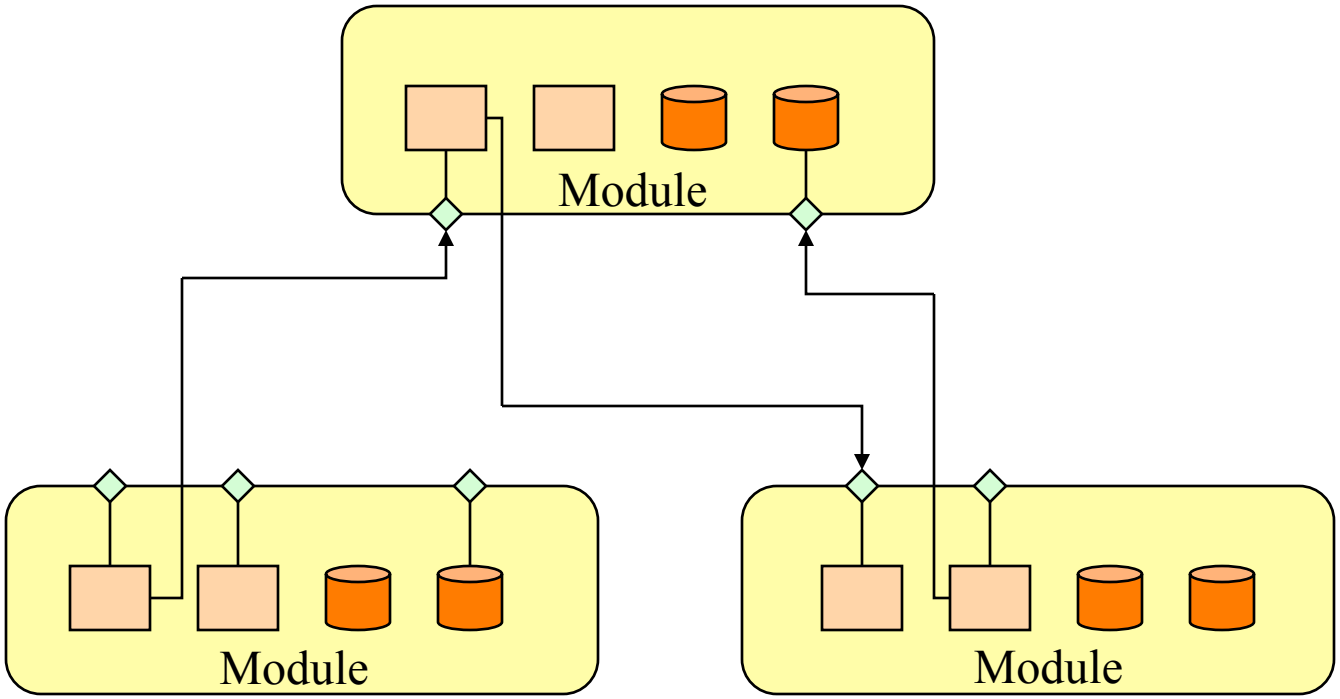


Modular Programming

- Group related data and functions together
- Module functions operate on module data
- Interface *vs*. Implementation
- Data abstraction and data encapsulation
- C, Algol, Ada

- Correctness - good
  - module-based testing
- Maintainability - good
  - localized changes
- Flexibility - better
- Reusability - better
  - reuse entire modules

Modular Programming

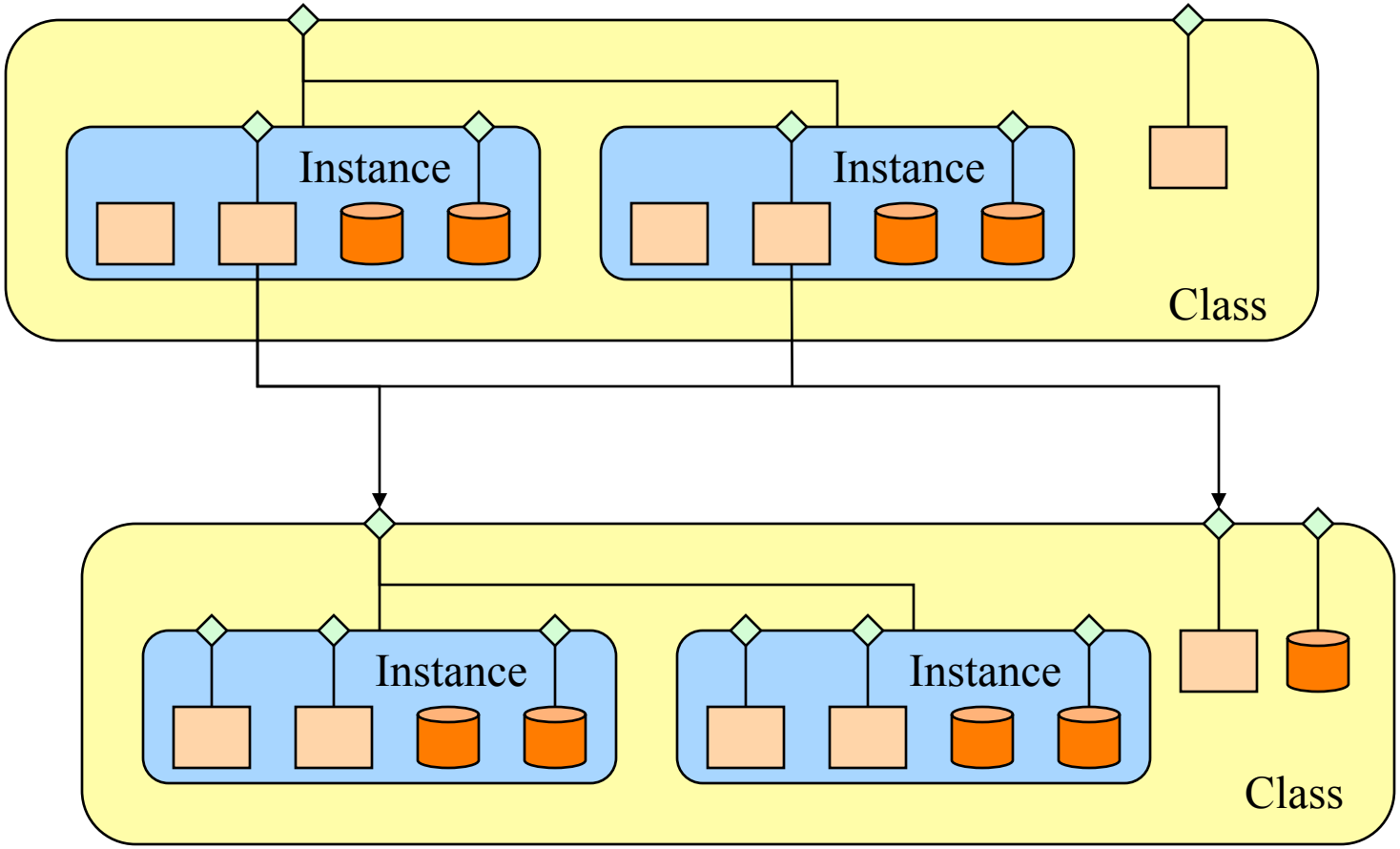


Object-oriented Programming

- Formalize convention of always passing module data structures to module functions
- Class: definition of data and the functions that operate on them
- Object: data created from class definition
- Smalltalk, C++, Java

- Correctness - good
- Maintainability - better
  - guaranteed internal consistency
- Flexibility - better
- Reusability - better
  - reuse concepts

Object-oriented Programming



Generic Programming

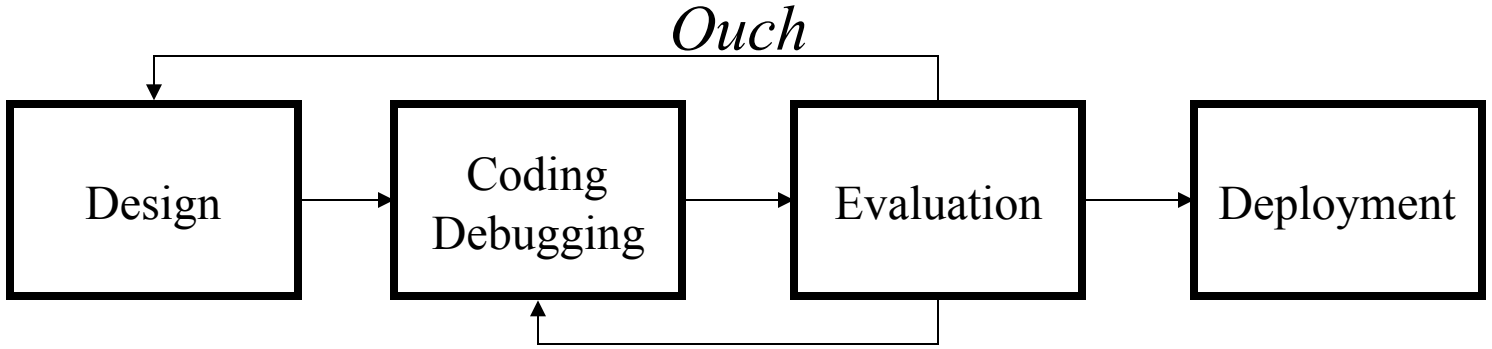
- Back to algorithms
- Objects that share the same interface can be generically manipulated
- Toolkits of objects and algorithms

What Is A Good Design?

- Too few classes, and code is limited in flexibility and reusability
  - lacks cohesion
- Too many classes, and code is more difficult to verify and maintain
  - too much coupling
- Design is still an art form

Design *vs*. Coding

- Coding from a design is much simpler than "hacking" because most of the hard work has been done
- Sometimes you have to hack to find the right design



Object-oriented Programming

- Languages that support object-oriented programming have built-in concept of class
- In addition to design, object-oriented programming also features inheritance
- Base class defines behavior; derived class defines new or redefines base behavior
- Simplifies code reuse

Reference Material

- Object Oriented Design with Applications, Grady Booch
- Object-oriented Software Construction, Bertrand Meyer
- Software Tools, Brian Kernighan, *et al*.