Reusability and Portability

Xiaojun Qi

Reuse Concepts

• Reuse is the use of components of one product to facilitate the development of a different product with different functionality

• Two types of reuse
  – Opportunistic (accidental) reuse
    • The product is built and parts are put into the part database for reuse
  – Systematic (deliberate) reuse
    • Reusable parts are constructed and products are built using these parts

• Why reuse?
  – To get products to the market faster
    • There is no need to design, implement, test, and document a reused component

Impediments to Reuse

1. Not invented here (NIH) syndrome: Would rather rewrite a routine from scratch
2. Concerns about faults in potentially reusable routines
3. Storage–retrieval issues
4. Cost of reuse
   – The cost of making an item reusable
   – The cost of reusing the item
   – The cost of defining and implementing a reuse process
5. Legal issues (contract software only)
6. Lack of source code for COTS components

The first four impediments can be overcome

Objects and Reuse

• Claim of CS/D
  – An ideal module has functional cohesion
  – Problem: The data on which the module operates ➔ We cannot reuse a module unless the data are identical

• The best type of module has informational cohesion ➔ This is an object (an instance of a class)

• An object comprises both data and action ➔ This promotes reuse

Reuse During Design and Implementation

• Various types of design reuse can be achieved
  – Some can be carried forward into implementation

• Design Reuse
  – Opportunistic reuse of designs is common when an organization develops software in only one application domain

Design Reuse

• Library or Toolkit: A set of reusable routines
• Application Frameworks: A framework incorporates the control logic of the design
• Design Patterns: A pattern is a solution to a general design problem
  – In the form of a set of interacting classes
• Software Architecture: Encompasses a wide variety of design issues, including:
  – Organization in terms of components
  – How those components interact
• Component-Based Software Engineering: Construct a standard collection of reusable components.
Reuse and Maintenance

- Reuse impacts maintenance more than development
  - Savings during maintenance are nearly 18 percent
  - Savings during development are about 9.3 percent

Portability

- A product is portable if it is significantly easier to modify the product as a whole to run it on another compile-hardware-operating system configuration than recode it from scratch.
- Difficulties of Achieving Portability
  - Hardware Incompatibilities
  - Operating System Incompatibilities
  - Numerical Software Incompatibilities: Differences in word size can affect accuracy
  - Compiler Incompatibilities

Why Portability?

- Portability is essential
  - Good software lasts 15 years or more
  - Hardware is changed every 4 years
- Upwardly compatible hardware works
  - But it may not be cost effective
- Portability can lead to increased profits
  - Multiple copy software
  - Documentation (especially manuals) must also be portable

Techniques for Achieving Portability

- Obvious technique
  - Use standard constructs of a popular high-level language
- Portable System Software:
  - Isolate implementation-dependent pieces
    - Example: UNIX kernel, device-drivers
  - Utilize levels of abstraction
    - Example: Graphical display routines

Techniques for Achieving Portability (Cont.)

- Portable Application Software
  - Use a popular programming language
  - Use a popular operating system
  - Adhere strictly to language standards
  - Avoid numerical incompatibilities
  - Document meticulously

Techniques for Achieving Portability (Cont.)

- Portable Data
  - Porting structured data
    - Construct a sequential (unstructured) file and port it
    - Reconstruct the structured file on the target machine
    - This may be nontrivial for complex database models