CS 315 - Oct 14, 2015

Chapter 8: Reusability and Portability

- Reuse concepts
    - Reuse is the use of components of one product to facilitate the development of a different product with different functionality
    - Two Types
        - **Opportunistic** (accidental)
            - First, the product is built
            - Then, the parts are put into the part database for reuse
        - **Systematic** (deliberate)
            - First, reusable parts are constructed
            - Then, products are built using these parts
    - Why?
        - To get products to the market faster
            - There is no need to design, implement, test, and document a reused component
        - On average, only 15% of new code serves an original purpose
            - In principle, 85% could be standardized and reused
            - In practice, reuse rates of no more than 40% are achieved
    - Impediments to Reuse
        - Not invented her (NIH) syndrome
        - Concerns about faults in potentially reusable routines
        - Storage-retrieval issues
        - Cost of reuse
            - The cost of making an item reusable
            - The cost of reusing an item
                - More mature (3 on the CMM scale) start to reuse components
            - The cost of defined and implementing a reuse process
        - Legal issue (contract software only)
        - Lack of source code for COTS components
    - 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

  - Claim of CS/D
    - The next best type of module has information cohesion
    - The is an object (an instance of a class)
    - An object comprises both data and action
    - The promotes reuse

- Reuse during Design and Implementation
  - Various types of design reuse can be achieved
    - Some can be carried forward into implementation
    - Opportunistic reuse of designs is common when organization develops software in only one application domain

  - Library or Toolkit
    - A set of reusable routines
    - The user is responsible for the control logic

  - Application Frameworks
    - A framework incorporates the control logic of the design
    - The user inserts application-specific routines in the "hot spots"
    - Faster than reusing a toolkit
      - More of the design is reused
      - The logic is usually harder to design than the operations

  - Design Patterns
    - A pattern is a solution to a general problem
      - In the form of a set of interacting classes

    - The classes need to be customized
      - Wrapper and Adapter patterns

    - If a design pattern is reused, then its implementation can also probably be reused
    - Patterns can interact with other patterns

  - Software Architecture
    - An architecture consisting of
      - A toolkit
      - A framework
      - Multiple design patterns

- Reuse of Software Architecture

            - Architecture reuse can lead to large-scale reuse
            - One mechanism
                - Software product lines

            - Architecture Patterns
                - Another way of achieving architectural reuse
                - Example: the model-view-controller (MVC) architecture pattern
                    - Can be viewed as an extension to GUIs
                    - Input-processing-output architecture

    - Reuse and Post-Delivery Maintenance
        - Reuse impacts maintenance more than development
        - Assumptions
            - 30% of entire product reused unchanged
            - 10% reused changed

- Portability Concepts
    - Have two products, P and P'
        - Functionally equivalent
        - Much easier to convert P into P' than to write P' from scratch

    - Impediments to Portability
        - Hardware
        - OS
        - Numerical/Memory
        - Compiler
        - Language

    - Why?
        - Is there any point in porting software?
            - Incompatibilites
            - One-off software
            - Selling company-specific software may give a competitor a huge advantage

        - On the contrary, portability is **essential**
            - Good software lasts 15 years or more
            - Hardware is changed every 4 years

        - Upwardly compatible harware 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 constucts of a popular high-level language
    - Isolate implementation-dependent pieces
        - Example: Unix kernal, device drivers
    - Utilize levels of abstraction
        - Example: Graphical display routines
- Portable Application Software
    - Use a popular programming language
    - Use a popular operating system
    - Adhere strictly to language standards
    - Avoid numerical incompatibilities
    - Document meticulously
    - File formats are often operating system-dependent
    - Porting structured data
        - Construct a sequential (unstructured) file and port it
        - Reconstruct the structured file on the target machine
        - The may be nontrivial for complex database models