CHAPTER 8

REUSABILITY AND PORTABILITY

Overview

- Reuse concepts
- Impediments to reuse
- Reuse case studies
- Objects and reuse
- Reuse during design and implementation
- Reuse and postdelivery maintenance
- Portability
- Why portability?
- Techniques for achieving portability

8.1 Reuse Concepts

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

The Two Types of Reuse

- Opportunistic (accidental) reuse
  - First, the product is built
  - Then, parts are put into the part database for reuse

- Systematic (deliberate) reuse
  - First, reusable parts are constructed
  - Then, 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

- 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

- Why do so few organizations employ reuse?
8.2 Impediments to Reuse

- Not invented here (NIH) syndrome
- Concerns about faults in potentially reusable routines
- Storage–retrieval issues

8.3 Reuse Case Studies

- The first case study took place between 1976 and 1982
- Reuse mechanism used for COBOL design
  ‣ Identical to what we use today for object-oriented application frameworks

8.3.1 Raytheon Missile Systems Division

- Data-processing software
- Systematic reuse of
  ‣ Designs
  ‣ 6 code templates
  ‣ COBOL code
  ‣ 3200 reusable modules

Raytheon Missile Systems Division (contd).

- Reuse rate of 60% was obtained
- Frameworks ("COBOL program logic structures") were reused
- Paragraphs were filled in by functional modules
- Design and coding were quicker

Impediments to Reuse (contd)

- 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
- Legal issues (contract software only)
- Lack of source code for COTS components
- The first four impediments can be overcome

Raytheon Missile Systems Division (contd).

- By 1983, there was a 50% increase in productivity
  ‣ Logic structures had been reused over 5500 times
  ‣ About 60% of code consisted of functional modules
- Raytheon hoped that maintenance costs would be reduced 60 to 80%
- Unfortunately, the division was closed before the data could be obtained
8.3.2 European Space Agency

- Ariane 5 rocket blew up 37 seconds after lift-off
  - Cost: $500 million
- Reason: An attempt was made to convert a 64-bit integer into a 16-bit unsigned integer
  - The Ada exception handler was omitted
- The on-board computers crashed, and so did the rocket

The Conversion was Unnecessary

- Computations on the inertial reference system can stop 9 seconds before lift-off
- But if there is a subsequent hold in the countdown, it takes several hours to reset the inertial reference system
- Computations therefore continue 50 seconds into the flight

The Cause of the Problem

- Ten years before, it was mathematically proven that overflow was impossible — on the Ariane 4
- Because of performance constraints, conversions that could not lead to overflow were left unprotected
- The software was used, unchanged and untested, on the Ariane 5
  - However, the assumptions for the Ariane 4 did not hold for the Ariane 5

European Space Agency (contd)

- Lesson:
  - Software developed in one context needs to be retested when integrated into another context

8.4 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

Objects and Reuse (contd)

- Claim of CS/D:
  - The next 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
8.5 Reuse During Design and Implementation

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

8.5.1 Design Reuse

- Opportunistic reuse of designs is common when an organization develops software in only one application domain

Library or Toolkit

- A set of reusable routines
  - Examples:
    - Scientific software
    - GUI class library or toolkit
  - The user is responsible for the control logic (white in figure)

8.5.2 Application Framework

- A framework incorporates the control logic of the design
- The user inserts application-specific routines in the "hot spots" (white in figure)

8.5.2 Application Framework

- Faster than reusing a toolkit
  - More of the design is reused
  - The logic is usually harder to design than the operations
- Example:
  - IBM's Websphere
    - Formerly: e-Components, San Francisco
    - Utilizes Enterprise JavaBeans (classes that provide services for clients distributed throughout a network)

8.5.3 Design Patterns

- A pattern is a solution to a general design problem
  - In the form of a set of interacting classes
- The classes need to be customized (white in figure)
We want a program that will generate widgets that can run under different operating systems.

Abstract classes and abstract (virtual) methods

The interfaces between the client and the generator are abstract

The application program is uncoupled from the specific operating system.

Encompasses a wide variety of design issues, including:
- Organization in terms of components
- How those components interact

An architecture consisting of
- A toolkit
- A framework, and
- Three design patterns

Architecture reuse can lead to large-scale reuse

One mechanism:
- Software product lines

Case study:
- Firmware for Hewlett-Packard printers (1995-98)
  - Person-hours to develop firmware decreased by a factor of 4
  - Time to develop firmware decreased by a factor of 3
  - Reuse increased to over 70% of components

Reuse impacts maintenance more than development

Assumptions
- 30% of entire product reused unchanged
- 10% reused changed
### Results

<table>
<thead>
<tr>
<th>Activity</th>
<th>Percentage of Total Cost over Product Lifetime</th>
<th>Percentage Savings over Product Lifetime due to Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>33%</td>
<td>9.3%</td>
</tr>
<tr>
<td>Postdelivery maintenance</td>
<td>67%</td>
<td>17.9%</td>
</tr>
</tbody>
</table>

- Savings during maintenance are nearly 18%
- Savings during development are about 9.3%

### 8.7 Portability

- Product P
  - Compiled by compiler C₁, then runs on machine M₁ under operating system O₁
- Need product P', functionally equivalent to P
  - Compiled by compiler C₂, then runs on machine M₂ under operating system O₂
- P is portable if it is cheaper to convert P into P' than to write P' from scratch

### 8.7.1 Hardware Incompatibilities

- Storage media incompatibilities
  - Example: Zip vs. DAT
- Character code incompatibilities
  - Example: EBCDIC vs. ASCII
- Word size

### 8.7.2 Operating System Incompatibilities

- Job control languages (JCL) can be vastly different
  - Syntactic differences
- Virtual memory vs. overlays

### 8.7.3 Numerical Software Incompatibilities

- Differences in word size can affect accuracy
- No problems with
  - Java
  - Ada
8.7.4 Compiler Incompatibilities

- FORTRAN standard is not enforced
- COBOL standard permits subsets, supersets
- ANSI C standard (1989)
  - Most compilers use the pcc front end
  - The lint processor aids portability
- ANSI C++ standard (1998)

Language Incompatibilities (contd)

- Ada standard — the only successful language standard
  - First enforced legally (via trademarking)
  - Then by economic forces
- Java is still evolving
  - Sun copyrighted the name to ensure standardization

8.8 Why Portability?

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

Why Portability? (contd)

- On the contrary, 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

8.9 Techniques for Achieving Portability

- Obvious technique
  - Use standard constructs of a popular high-level language
- But how is a portable operating system to be written?

8.9.1 Portable System Software

- Isolate implementation-dependent pieces
  - Example: UNIX kernel, device-drivers
- Utilize levels of abstraction
  - Example: Graphical display routines
8.9.2 Portable Application Software

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

8.9.3 Portable Data

- 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
  - This may be nontrivial for complex database models

Strengths of and Impediments to Reuse and Portability

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Impediments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuse</td>
<td>NIR syndrome (Section 8.1.2)</td>
</tr>
<tr>
<td></td>
<td>Potential quality issues (Section 8.2.2)</td>
</tr>
<tr>
<td></td>
<td>Retrieval issues (Section 8.2)</td>
</tr>
<tr>
<td></td>
<td>Cost of making a component reusable (non-trivial) (Section 8.2)</td>
</tr>
<tr>
<td></td>
<td>High costs of providing a reusable component (Section 8.2)</td>
</tr>
<tr>
<td></td>
<td>Lack of source code for COTS components (Section 8.2)</td>
</tr>
</tbody>
</table>

Portability

- Software has to be ported to new hardware every 4 years or so (Section 8.8)
- More copies of COTS software can be sold (Section 8.8)

- Potential incompatibilities:
  - Hardware (Section 8.7.1)
  - Operating systems (Section 8.7.2)
  - Numerical software (Section 8.7.3)
  - Compilers (Section 8.7.4)
  - Data formats (Section 8.5.3)