Development Methodologies

Prof. Dr. Josef M. Joller
jjoller@hsr.ch

REUSABILITY, PORTABILITY, AND INTEROPERABILITY

Overview

- Reuse concepts
- Impediments to reuse
- Reuse case studies
- Objects and reuse
- Reuse during the design and implementation phases
- Reuse and maintenance
- Portability
- Techniques for achieving portability
- Interoperability

Reuse Concepts

Two types of reuse

- Accidental reuse
  - First, product is built
  - Then, parts put into part database for reuse
- Planned reuse
  - First, reusable parts are constructed
  - Then, products are built using these parts

Why reuse?

Minor Reason

- It is expensive to design, implement, test, and document software
- Only 15% of new code serves an original purpose (average)
- Reuse of parts saves
  - Design costs
  - Implementation costs
  - Testing costs
  - Documentation costs

Major Reason

- Maintenance

→ **Maintenance consumes two-thirds of software cost**

Impediments to Reuse

Not invented here (NIH) syndrome

Concerns about faults in potentially reusable routines

Storage–retrieval

Cost of reuse!

Reuse Rate

- Theoretical maximum is 85%
- What can we get in practice?
- Consider case studies (1975 through 2000)

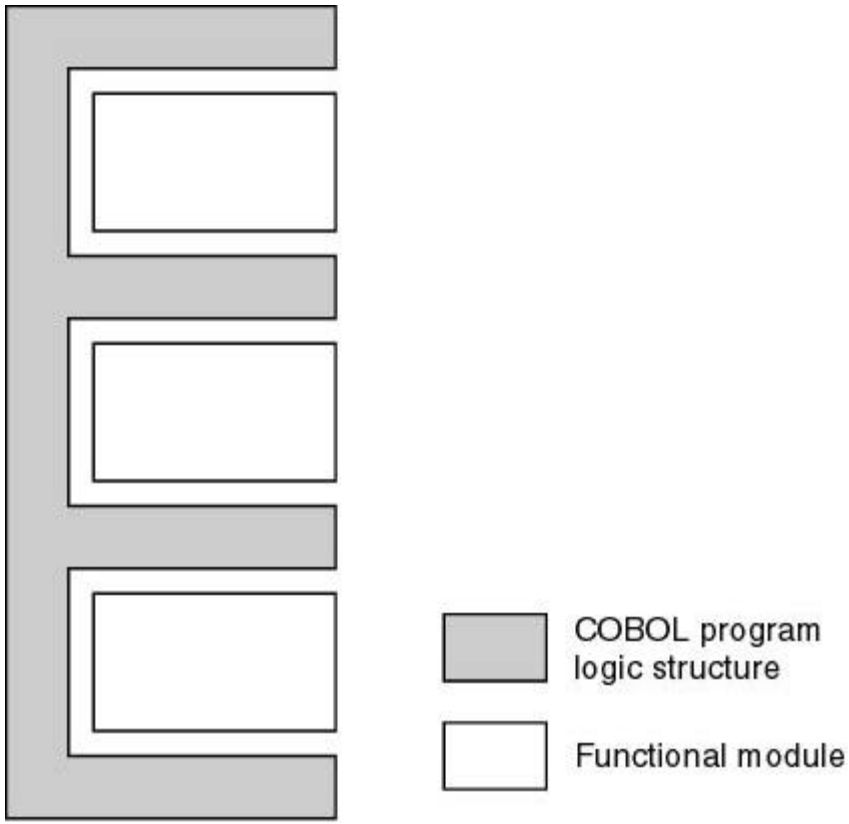
Raytheon Missile Systems Division

Data-processing software

Planned reuse of

- Designs
  - 6 code templates
- COBOL code
  - 3200 reusable modules

Reuse rate 60% (1976–1982)



GTE Data Services

Data-processing software

Strongly encouraged by management

- Cash incentives when module was accepted for reuse
- Cash incentive when module was reused

Accidental reuse of

- Modules

Reuse rate

- (1988) 15% reused code, $1.5 million saved
- (est. 1989) 20% reused code
- (est. 1993) 50% reused code

European Space Agency

Ariane 5 rocket blew up 37 seconds after lift-off

Cost: $500 million

Reason: attempt to convert 64-bit integer into 16-bit unsigned integer, without Ada exception **handler**

On-board computers crashed, so did rocket

Conversion was unnecessary

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

European Space Agency (contd)

Cause of 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
- Software was used, unchanged and untested, on Ariane 5
- But, the assumptions for the Ariane 4 no longer held

Lesson

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

Reuse During Design and Implementation

Design reuse

- Library or toolkit
  - Make or buy
- Framework
  - Domain specific
  - Expensive to develop & maintain
- Design pattern
  - Common technique
- Software architecture
  - Common technique

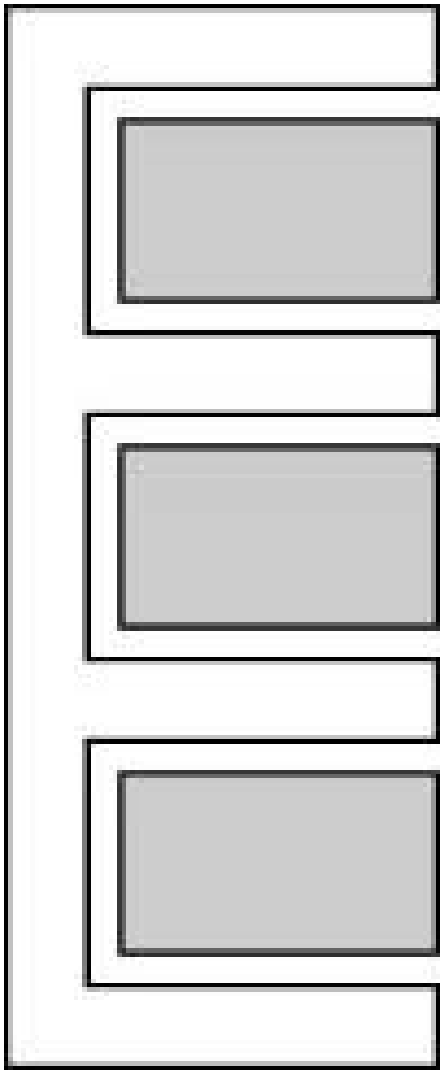
Library or Toolkit

Set of reusable routines

Examples:

- Scientific software
- GUI class library or toolkit

The user is responsible for the control logic (white in figure)



Application Framework

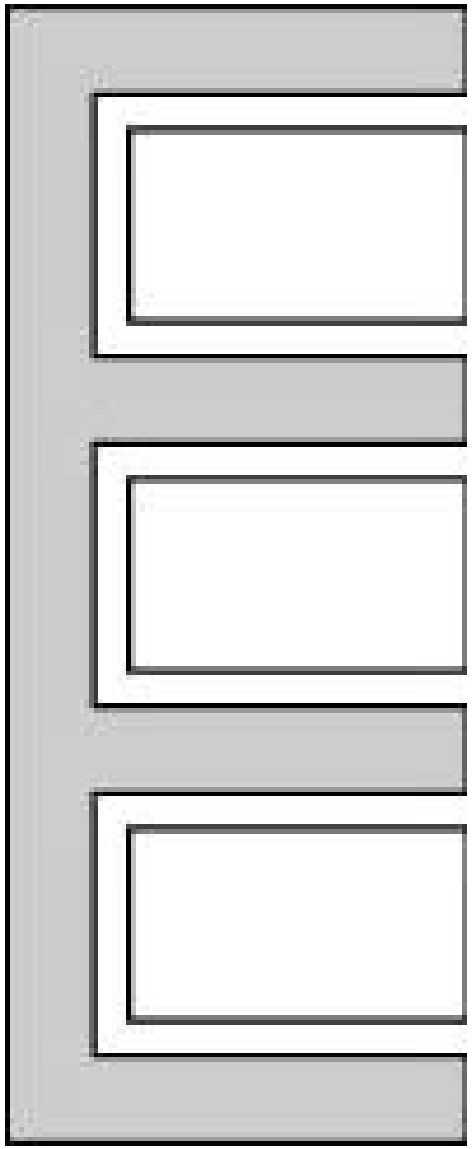
Control logic of the design

"Hot spots" (white in figure)

Faster than reusing toolkit

More of design is reused

The logic is usually harder to design than the operations

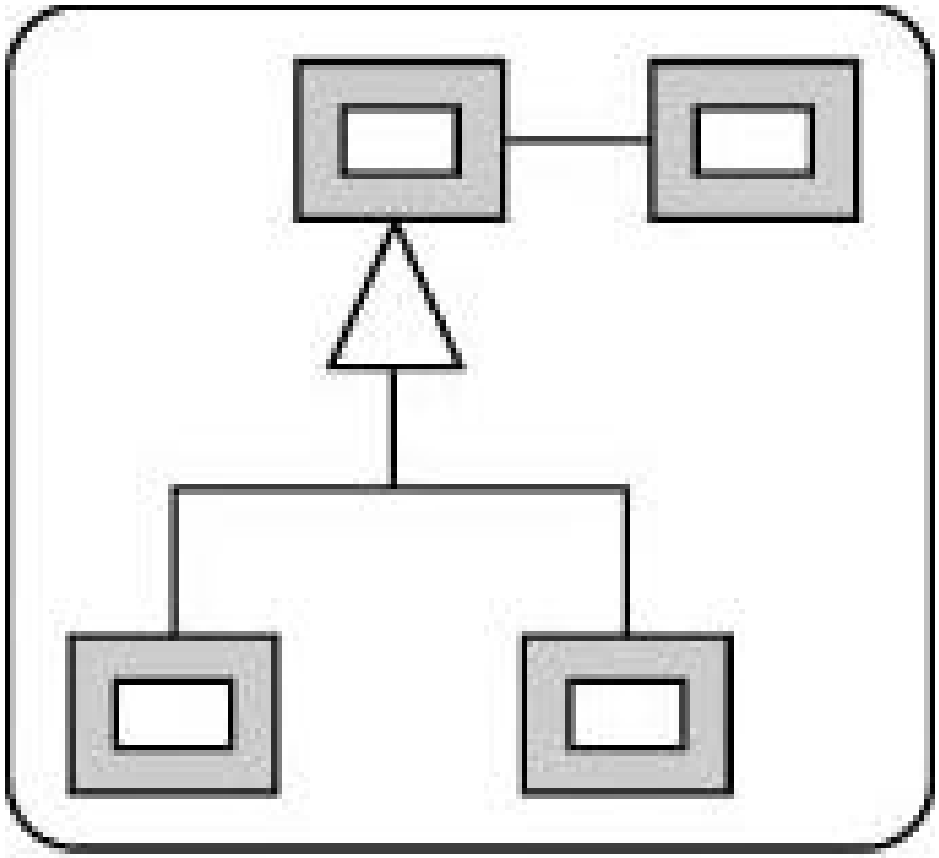


Design Pattern

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)



Reuse of Software Architecture

Architecture reuse can lead to large-scale reuse

One mechanism:

- Software product lines

Case study:

- Hewlett-Packard printers (1995 to 1998)
  - Person-hours to develop firmware decreased by a factor of 4
  - Time to develop firmware decreased by factor of 3
  - Reuse has increased to over 70% of components

Objects and productivity

Reuse achieved

- *Not* via modules with functional cohesion,
- but via objects (informational cohesion) [classes]

In general

- Software costs have decreased
- Overall quality has improved
- Large products are essentially collection of smaller products

Difficulties and Problems

Learning curve

- Particularly noticeable with GUI

Problems with inheritance

- New subclass does not affect its superclass
- But, any change to a superclass affects all its subclasses
- Subclasses low in the inheritance tree can be huge (inherited attributes)

Polymorphism and dynamic binding

- Maintenance problems were already discussed

Reuse - Why Portability?

Difficulties hampering portability

- One-off software
- Hardware incompatibility
- Lifetimes of software, hardware
- Multiple copy software

Portability saves money!

Portable system software

- Isolate implementation-dependent pieces
  - UNIX kernel, device-drivers
- Levels of abstraction
  - Graphics

Portability Strategies

Portable application software

- Use popular language
- Use popular operating system
- Adhere to language standards
- Avoid numerical incompatibilities
- Excellent documentation

Future Trends in Interoperability

.NET (not yet), JAVA and CORBA are currently the "big's"

Other interoperability products may succeed, such as Java EnterpriseBeans (EJB's)

- Session / Transaction,
- Entity / DB,
- Message-Driven / Messaging

Web-based applications need to be integrated into client–server products

- XML (Extensible Markup Language) will probably play a major role
- XML Protocol (SOAP, Web Services) based services