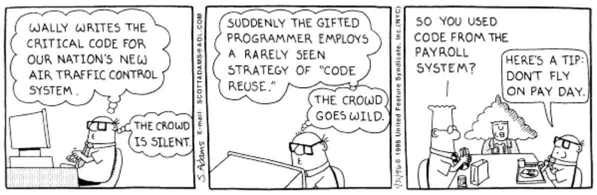
Reuse and Components (1)

CS3 / SEOC1

Note 12



Perils of Reuse
(cf. Ariane 5...)

Software Engineering with Objects and Components

Software Engineering is concerned with processes, techniques and tools which enable us to build "good" systems.

Object-Orientation is a (methodology, technique, process, suite of design and programming languages and tools) with which we may build good systems.

Components are "units of reuse and replacement".

ESA's Ariane 5: Flight 501

- June 4th 1996, Ariane 5, veered off flight path, broke up and exploded less than 40 seconds into its maiden flight
- Inquiry board report July 1996
  - sequence of events: nominal behaviour up to H0+36 seconds; failure of backup Inertial Reference System (SRI), followed immediately by failure of active system; boosters and main engine swivel to extreme position, causing abrupt veer; launcher (correctly) self-destructs
  - both SRI's recovered and analysed
  - active SRI had failed due to software exception (out-of-range error), On Board Computer had interpreted diagnostic report as navigational data

Ariane 5: (root?) error causes

- "hardware failure" mentality – reliance on backups
- alignment function (Ariane 4) obsolete in Ariane 5
- consequences of reuse not sufficiently explored
- exception handling incomplete (decision and justification obscured from external review)
- V & V inadequate
- cooperation amongst Ariane 5 partners inadequate

Types of Reuse

- Knowledge reuse
  - artifact reuse
  - pattern reuse
- Software reuse
  - code reuse
  - inheritance reuse
  - template reuse
  - component reuse
  - framework reuse

Reuse of Knowledge: Artifact Reuse

- reuse of use cases, standards, design guidelines, domain specific-knowledge
- *Pluses:* consistency between projects, reduced management burden, global comparitors of quality and knowledge
- *Minuses:* overheads, constraints on innovation (coder versus manager)

Reuse of Knowledge: Patterns

- reuse of publically documented approaches to solving problems (e.g. class diagrams of typically 1–5 classes)
- *"A pattern is a named nugget of insight that conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns."*
- See links from SEOC1 lecture log for more info on patterns

Patterns (cont.)

- Origin of Patterns:
  - Ward Cunningham & Kent Beck: pattern languages for OO novice programmers – 1987
  - Jim Coplien: C++ programming idioms – 1991
  - "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) publish *Design Patterns: Elements of Reusable Object-Oriented Software* – 1994
- *Pluses:* long life-span, applicable beyond current programming languages, applicable beyond OO?
- *Minuses:* no immediate solution, no actual code, knowledge hard to capture/reuse

Types of Software Reuse: Code Reuse

- reuse of (visible) source code – *code reuse* versus *code salvage*
- *Pluses:* reduces written code, reduces development and maintenance costs
- *Minuses:* can increase coupling, substantial initial investment

Types of Software Reuse: Inheritance

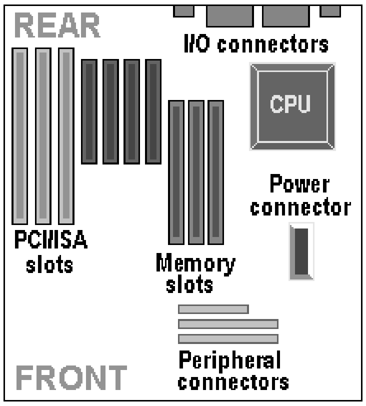
- using inheritance to reuse code behavior
- *Pluses:* takes advantage of existing behavior, decrease development time and cost
- *Minuses:* can conflict with component reuse, can lead to fragile class hierarchy – difficult to maintain and enhance

Types of Software Reuse: Template Reuse

- reuse of common data format/layout (e.g. document templates, web-page templates)
- *Pluses:* increase consistency and quality, decrease data entry time
- *Minuses:* needs to be simple, easy to use, consistent amongst group

Further Software Reuse: Component Based Development

- analogy to electronic circuits: software "plug-ins"



Intel ATX Motherboard (launched in 1996)

Further Software Reuse: Component Reuse

- reuse of prebuilt, fully encapsulated "components"; typically self-sufficient and provide only 1 concept (high cohesion)
- *Pluses:* greater scope for reuse, common platforms (e.g. JVM) more widespread, 3rd party component development
- *Minuses:* development time, genericity, need large libraries to be useful

Components: A Definition

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

ECOOP '96. (European Workshop on Component-Oriented Programming)

Further Software Reuse (2) Framework Reuse

- collection of basic functionality of common technical or business domain (generic "circuit boards" for components)
- *Pluses:* supports CBD, can account for 80% of code
- *Minuses:* substantial complexity, leading to long learning process; platform specific; framework compatibility issues leading to vendor specificity; implements *easy* 80%
- *cf. software architecture, product line architectures, domain component reuse, domain specific programming, ...*

Summary

- many kinds of reuse – of both knowledge and software
- each has pluses and minuses
- component reuse is a form of software reuse
  - encapsulation
  - high cohesion
  - specified interfaces
  - explicit context dependencies
- Does this work...?
- *... tune in next lecture*