Advanced Programming Software Engineering Lecture 2

Phil Gross

23 Oct 2002

Remotivating

- CS teaches how to write ideal software
- In the real world, software is usually late, overbudget, and broken
- Software lasts much longer than either hardware or employees
- The real world is a harsh environment, and software is fundamentally brittle

Case Study: Ariane 501

- Next-generation launch vehicle
- Successor to the Ariane 4
- Prestige project for ESA
- Maiden flight: June 4th 1996

A Part of the System

- Inertial Reference System
  - □ What's my position, velocity, and acceleration?
- Critical, obviously
  - □ Dual redundant
- Calibrated on launch pad
- Largely carried over from Ariane 4
  - □ recalibration routine allowed to continue running for 40 sec after launch

The Problems

- Recal routine never used after launch, but still active
- One step in recal converted floating point value of horizontal velocity to integer
- Ada automatically throws an exception if data conversion is out of bounds
- If exception not handled, IRS returned diagnostic data instead of position/velocity info

The Situation

- Perfect launch
- Starts flying much faster than Ariane 4
- Horizontal component goes out of bounds for integer conversion
- Both IRSs switch to diagnostic mode
- Control system interprets diagnostic data as very weird orientation
- And attempts to correct it…

Ariane 501 Go Boom

- 150+ feet high
- 25 tons of hydrogen
- 130 tons of liquid oxygen
- Over 500 tons of solid propellant
- Failure at altitude of 2.5 miles
- Ten years and $7,000,000,000

Postmortem

- Recal routine had no business being active after launch
- Horizontal velocity parameter conversion was deliberately allowed to be unchecked
- Q: Who's to blame?
  - A: No one, of course. "Mistakes were made"

At Least It Was Pretty

Reuse Specification Error

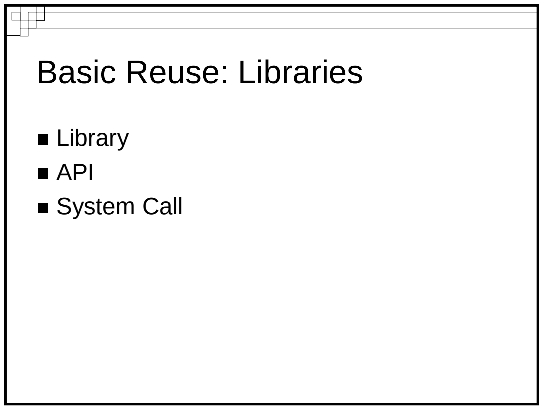
- Horizontal bias needed to fit into 16 bits
- Documented somewhere
- Not in the code
- Software had never been tested with actual flight parameters
  - Problem easily reproduced in test environment after the fact

Things to Think About Early

- Reuse
- Portability
- Interoperability
- Scalability
- Your future self will thank you

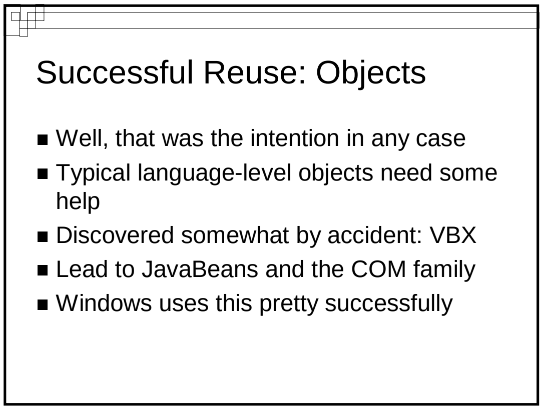
Impediments to Reuse

- Lack of trust / NIH
- Logistics of reuse
- Loss of knowledge base
- Mismatch of features (Kangaroos)



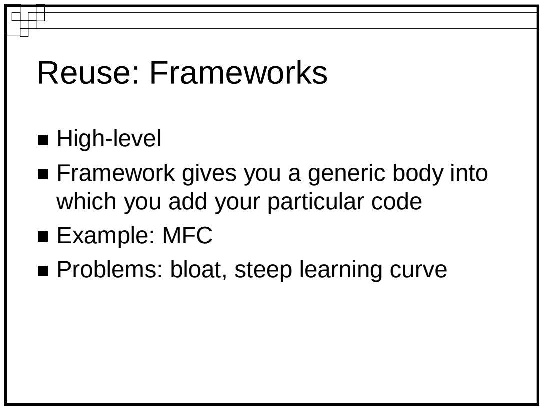
Basic Reuse: Libraries

- Library
- API
- System Call



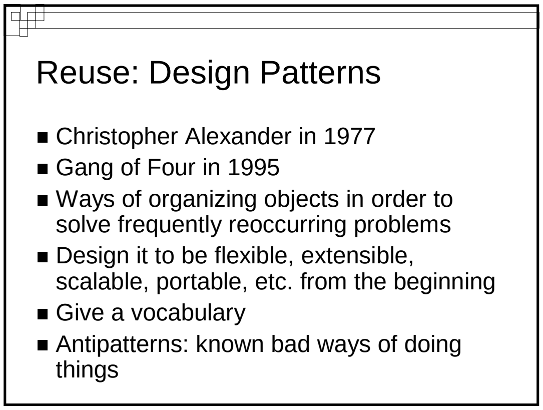
Successful Reuse: Objects

- Well, that was the intention in any case
- Typical language-level objects need some help
- Discovered somewhat by accident: VBX
- Lead to JavaBeans and the COM family
- Windows uses this pretty successfully



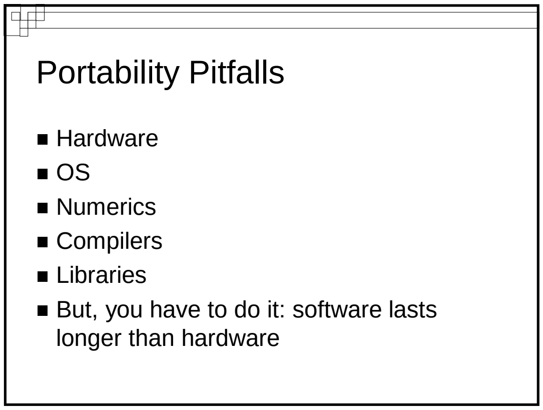
Reuse: Frameworks

- High-level
- Framework gives you a generic body into which you add your particular code
- Example: MFC
- Problems: bloat, steep learning curve



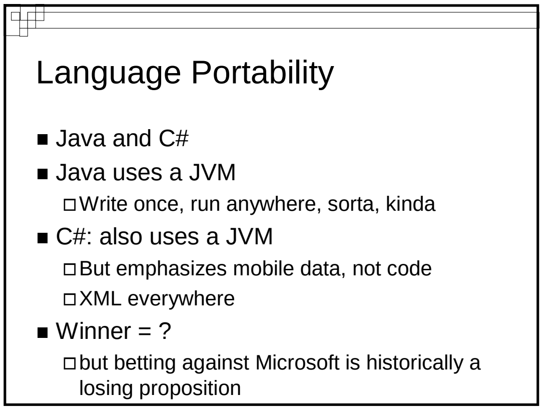
Reuse: Design Patterns

- Christopher Alexander in 1977
- Gang of Four in 1995
- Ways of organizing objects in order to solve frequently reoccurring problems
- Design it to be flexible, extensible, scalable, portable, etc. from the beginning
- Give a vocabulary
- Antipatterns: known bad ways of doing things



Portability Pitfalls

- Hardware
- OS
- Numerics
- Compilers
- Libraries
- But, you have to do it: software lasts longer than hardware



Language Portability

- Java and C#
- Java uses a JVM
  - Write once, run anywhere, sorta, kinda
- C#: also uses a JVM
  - But emphasizes mobile data, not code
  - XML everywhere
- Winner = ?
  - but betting against Microsoft is historically a losing proposition

Interoperability

- COM, CORBA, EJB, Web Services
- define abstract services
- Allow programs in any language to access services in any language in any location
- Object-ish

Scalability

- Just keep it in mind
  - Familiarity with patterns can help
- Don’t worry about scaling beyond abilities of machine
  - Avoid unnecessary barriers
  - Plus maybe graceful overload handling
- From single connection, to forking processes, to threads, to thread pool

UML

- History
- Use case diagrams
- Class diagrams
- Sequence diagrams
- State diagrams

UML History

- Need to draw pictures
  - Every guru has his own style
- “The three amigos”
  - Grady Booch, James Rumbaugh, Ivar Jacobson
  - “The three egos”
- Rational
  - The Microsoft of Software Engineering

Use Case Diagrams

- Neither Janak nor I like these much
- The idea is necessary
  - Classic SoftE disaster: system is built and runs perfectly. Unfortunately, it’s the wrong system.
- Idiotic little stick-figure diagrams are not

Typical Use Case

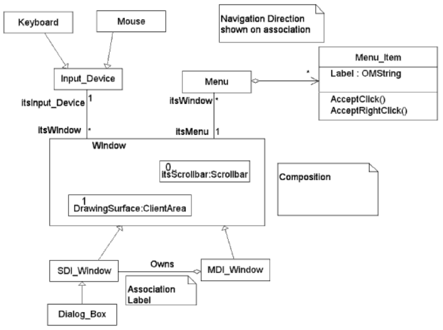
- Subway Routing
  - Touch here
  - User touches
  - Map and “touch destination”
  - User touches times square
  - Highlight times square
  - Route is calculated
  - Route shown with transfer highlighted
  - Wait 30 sec
  - Ask if should stay up
  - Otherwise reset

Class Diagram

- The “guts”of UML
- Show static class relationships
  - □ Generalization = inheritance
  - □ Classes, Attributes, and Operations
- *Dramatis Personae* for your program

Relationships

- Association = “has a”
- Have multiplicities
  - □ And, by extension, mandatory/optional
- Can also have role name
- Navigability
- Constraints/contracts
- Composition



Sequence Diagrams

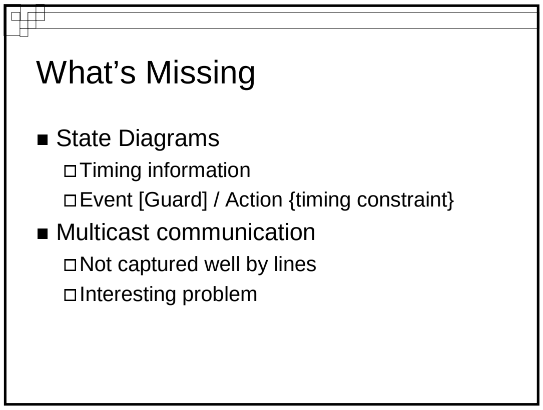
- Show lifetime of objects
- And their interaction
- “lifelines” arranged vertically
- Same info as collaboration diagram
  - □ Has sequence annotations on 2D diagram

State Diagrams

- States, transitions between them
- Long running actions happen within states
- Fast, uninterruptable actions transition between states
- Transition labels: *Event [Guard] / Action*

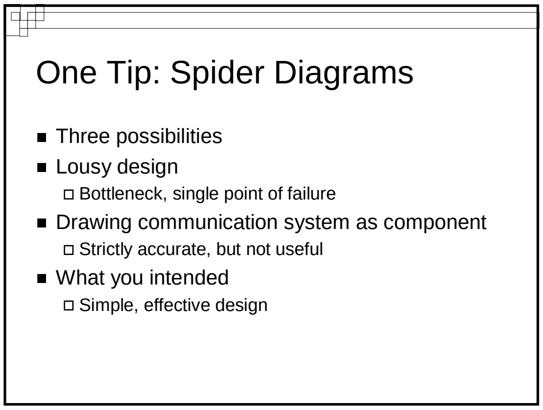
Other UML Diagrams

- Component/Deployment
  - □ What pieces are running where
- Activity Diagram
  - □ Fancy flow chart
- Non-UML
  - □ Architecture diagrams
  - □ Components and connectors



What’s Missing

- State Diagrams
  - □ Timing information
  - □ Event [Guard] / Action {timing constraint}
- Multicast communication
  - □ Not captured well by lines
  - □ Interesting problem



One Tip: Spider Diagrams

- Three possibilities
- Lousy design
  - □ Bottleneck, single point of failure
- Drawing communication system as component
  - □ Strictly accurate, but not useful
- What you intended
  - □ Simple, effective design