# Reuse and Components

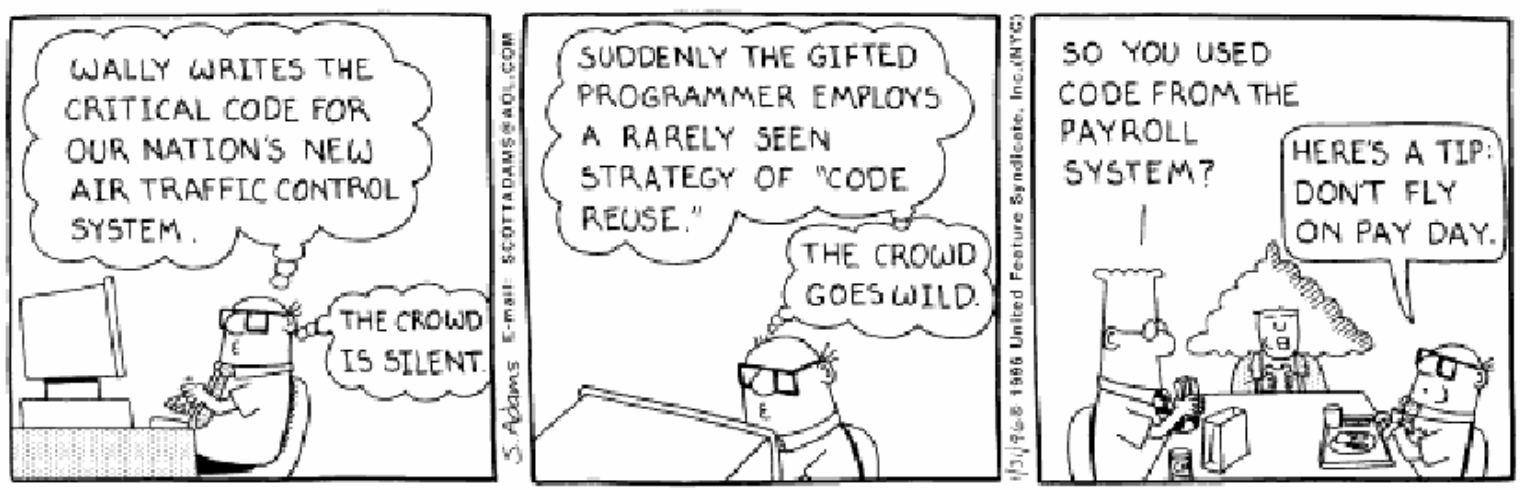
Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

# Reuse in Software Engineering






- 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, Arine 5, veered off flight path, broke up and exploded less that 40 seconds into its maiden flight
- Inquiry board report July 1996
  - Sequence of events: nominal behaviour up to H0+36; 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
- 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 inadequante

# Examples of Types of reuse

- **Application system reuse**: the whole of an application system may be reused by incorporating it without change into other systems
- **Component reuse**: components of an application ranging in size from sub-systems to single objects may be reused
- **Object and function reuse**: Software components that implement a single function, such as a mathematical function or an object class, may be reused

# Benefits of Software Reuse

- **Increased dependability**: Reused software, which has been tried and tested in working systems, should be more dependable than new software because its design and implementation faults have already been found and fixed.
- **Reduced process risk**: The cost of existing software is already known, while the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as subsystems are reused.
- **Effective use of specialists**: Instead doing the same work over and over, these application specialists can develop reusable software that encapsulates their knowledge.
- **Standards compliance**: Some standards, such as user interface standards, can be implemented as a set of standard reusable components. For example, if menus in a user interface are implemented using reusable using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users are less likely to make mistakes when presented with a familiar interface.
- **Accelerated development**: Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time should be reduced.

# Problems with reuse

- **Increased maintenance costs**: If the source code of a reused software system or component is not available the maintenance costs may be increased because the reused elements of the system may become increasingly incompatible with system changes.
- **Lack of tool support**: CASE toolsets may not support development with reuse. It may be difficult or impossible to integrate these tools with a component library. The software process assumed by these tools may not take reuse into account.
- **Not-invented-here syndrome**: Some software engineers prefer to rewrite components because they believe they can improve on them. This ids partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.
- **Creating and maintaining a component library**: Populating a reusable component library and ensuring the software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature.
- **Finding, understanding and adapting reusable components**: Software components have to be discovered in a library understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they will make include a component search as part of their normal development process.

# Planning Reuse: Key Factors

- The development schedule for the software
- The expected software lifetime
- The background, skills and experience of the development team
- The criticality of the software and its non-functional requirements
- The application domain
- The platform on which the system will run

# Approaches Supporting Software Reuse

- Design Patterns
- Component-based Development
- Application Frameworks
- Legacy system wrapping
- Service-oriented systems
- Application product lines
- COTS (Commercial-Off-The-Shelf) integration
- Configurable vertical applications
- Program libraries
- Program generators
- Aspect-oriented software development

# Types of Reuse

- Knowledge reuse
  - Artificial reuse
  - Pattern reuse
- Software reuse
  - Code reuse
  - Inheritance reuse
  - Template reuse
  - Components
  - 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 comparators of quality and knowledge
- **Minuses**: overheads, constraints on innovation (coder versus manager)

# Reuse of Knowledge: Patterns

- "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."
- "Patters and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience."
- Reuse of publicly documented approaches to solving problems (e.g., class diagrams)
- Origin of Patterns:
  - Ward Cunningham & Kent Beck: pattern language for OO novice programmers, 1987
  - Jim Coplien: C++ programming idioms, 1991
  - E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, 1994
- Plusses: 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 behaviour
- **Pluses**: takes advantage of existing behaviour, 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, etc.)
- **Pluses**: increase consistency and quality, decrease data entry time
- **Minuses**: needs to be simple, easy to use, consistent among groups

# Types of Software Reuse: Component

- Analogy to electronic circuits: software "plug-ins"
- Reuse of prebuilt, fully encapsulated "components"; typically self-sufficient and provide only one concept (high cohesion)
- ***A Definition of Component: "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)
- **Pluses**: greater scope for reuse, common platforms (e.g., JVM) more widespread, third party component development
- **Minuses**: development time, genericity, need large libraries to be useful

# Types of Software Reuse: Framework

- 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, implement easy 80%
- cf. software architecture, product line architectures, domain component reuse, domain specific programming,…

# Reuse: The Success Stories

- EDS (Electronic Data Systems) early 1990s
- Smalltalk programmers given the same specifications and test suites as earlier team who produced PL/1 system
- PL/1
  - 265,000 SLOC (Source Line of Code)
  - 152 staff months
  - 19 months to develop
- Smalltalk
  - 22 SLOC
  - 10 staff months
  - 3.5 months to develop

# Reuse: The Success Stories

- NASA SEL (Software Engineering Laboratory)
- Routinely 75% or higher reuse
- Development time reduce by 90%
  - 58,000 hours for application development
  - Recently, with reuse, reduced to approx. 6,000

# Whatever Happened to Reuse?

- EDS experience not repeated
  - Common arguments: wrong language, "not invented here" syndrome, lack of management support
  - Numbers inflated?
- NASA success hard to transfer: domain and economics
  - Specialised problem domain (spacecraft flight dynamics)
  - Very high initial investment
    - 36,000 hours of domain analysis
    - 40,000 hours to develop components
    - Design of reusable asses library starts 1992
    - First application using library in 1995
    - SEL estimates library development costs recouped by 4th mission

# Reuse Pitfalls

- **Underestimating** the difficulty of reuse
  - Software must be more general
  - Similarities among projects often small
  - Much of what is reused is already provided by OS
  - Universe in constant flux (hardware, software, environment, requirements, expectations, etc.)
- Having or setting **unrealistic** expectations
  - OO reuse overly "hyped"
  - "Software is not built from Lego™ blocks" – Alexander Ran
  - Reuse domain may be unrealistic
  - Expectations for reuse outstrip skills of developers
- **Not investing** in reuse
  - Reuse costs: time and money in development, analysis, design, implementation, testing,…
- Being **too focused** on code reuse
  - Focus on code reuse as end, not means
  - "Lines of code reused" metric meaningless
  - Design reuse often neglected in favour of code reuse
  - Too little abstraction at framework level
- Generalising after the fact
  - Design often migrate from general to specific during development
  - System not designed with reuse in mind (cf. code reuse versus code salvage)
- Allowing too many connections
  - Coupling unavoidable, but must be very low to permit reuse
  - Circular dependencies also problematic – where to break the chains?

# Difficulties with Component Development

- **Economic**
  - Small business do not have long term stability and freedom required
- Where is the third party component **market**?
  - Effort in (re)using components
  - Cross-platform and cross-vendor compatibility
  - Many common concepts, few common components
  - Some success: user interfaces, data management, thread management, data sharing between applications
  - Most successful: GUIs and data handling (e.g., Abstract Data Types)

# Components in Java

- JavaBeans
  - Visual composition of components
  - Builder introspection of Bean features (properties, methods, events)
  - Composition of Beans into applets, applications or other Beans
- ADTs: **java.util.*** library
- GUIs: The Java Foundation Classes (JFC)
  - History: AWT ("Abstract Window Toolkit", 1995), JFC (1997 – Swing)
  - All Components are JavaBeans
  - Lightweight UI framework
    - Peerless emulation versus layered ("peer") toolkit model
    - Cross platform (no native code)
  - Pluggable look and feel
  - No framework lock-in ("easily" compatible with 3rd party ccomponents)
  - Subclasses are fully customisable and extendible (according to Sun)

# Reading/Activity

- William N. Robinson and Hang G. Woo, *Finding Reusable UML Sequence Diagrams Automatically*. In IEEE Software, September/October 2004.
- Tiffany Winn and Paul Calder, Is This a Pattern?. In IEEE Software, January/February 2002.

# Summary

- Many types 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
  - Component development requires significant time and effort
  - As does component reuse
  - Compnent reuse has been successful for interfaces and data handling
- Employing reuse requires management
- Java (potentially) supports cross-platform component reuse
  - JFC and java.util.* classes exemplify this