



A Tutorial on Software Reuse in Safety-Critical Systems

Leanna Rierson
(Leanna.Rierson@faa.gov)
June 6, 2001

1



Outline

- What is Reuse?
- Pros/Cons of Reuse
- Reuse Myths
- Why Reuse Isn't Used Much
- 7 Concepts Relevant to Reuse
- Successful Reuse – Pulling It All Together
- FAA Activities Related To Reuse
- Summary

2



What is Reuse?

3



What is Reuse?

- A hot buzzword?
- The newest silver bullet?
- Something greatly desired, but ever so elusive?
- Real and practical?

“If you ask five programmers what reuse is, you’ll get eight answers” (Steve Adolf)

4



What is Reuse? (cont)

- Software Reuse != Software Salvaging (Adolf)
 - Software reuse is software that is designed to be reused
 - Software salvaging is using software that was not designed for reuse
- We do a lot of “salvaging” in the aviation world today, but we want to do more “reusing”

5



What is Reuse? (cont)

- A process of implementing or updating software systems using existing software assets. (Sodhi)
 - Assets can be software components, objects, software requirement analysis and design models, domain architecture, database schema, code documentation, manuals, standards, test scenarios, and plans.
 - Software reuse may occur within a software system, across similar systems, or in widely different systems.
- Software reuse is the process of creating software systems from existing software assets, rather than building software systems from scratch. (Krueger)

6



What is Reuse? (cont)

- Goal of reuse: To use as much software data as possible from previous development efforts in order to reduce time, cost, and risks associated with re-development.

“Reuse is a bet on the future” (Williams)

7



PROS/CONS of REUSE

8



Potential Benefits of Reuse

- Meeting business needs (addressing the software crisis)
- Higher productivity
- Increased quality
- Quicker time to market
- Better use of resources
- Helps with system complexity issues



“Systematic reuse has the highest payback of any technology since software began.” (Williams)

9



Potential Risks of Reuse

- It requires more upfront investment
- It is a bit of a gamble on the future
- It can end up costing more, if not done properly
- It can induce errors, if not done properly
- **It must be used cautiously in safety-critical domains**



10



Reuse Myths



11



Reuse Myths

- Reuse is quick, easy, simple, & free.
- Buying components means no building.
- Components equal reuse.
- Reuse is just code.
- Maintenance is not building, therefore reuse does not apply.
- Increase productivity means loss of jobs.
- Reuse means everyone must do the same thing.

12



Lack of Reuse Utilization

13



Why Reuse Has Not Been Utilized Much

- It isn't taught in schools
- We have the "not invented here" attitude
- Cost is believed to be prohibitive
- Time constraints
- Culture
- Lack of experience
- Lack of tools
- Not understanding what reuse really is

14



7 Concepts Related to Reuse

15



7 Concepts Relevant to Reuse

- Planning for Reuse
- Domain Engineering
- Software Components
- Object-Oriented Technology
- Portability
- Commercial-off-the-shelf (COTS) Software
- Product Service History

16



1. Planning for Reuse

“If you don’t know where you are going,
any road will lead you there.” (eastern saying)

17



Planning for Reuse

- Reuse doesn’t just happen.
- Reuse must be well planned.
- Reuse must be well managed.

18



Reifer's 10 Steps To "Reuse Adoption"

- Define the company vision/strategy
- Determine company's current reuse status
- Establish an operation concept for the company
- Develop a company business plan
- Focus early efforts on company infrastructure
- Make an initial success
- Try the ideas before they are solidified
- Strive for a success image
- Iterate & refine the process based on results

19



McConnell's Keys to Success in Reuse



- Take advantage of personnel continuity between old & new programs
- Do not overestimate your savings
- Secure long-term, high-level management commitment to a reuse program
- Make reuse an integral part of the development process
- Establish a separate reuse group
- Focus on small, sharp, domain-specific components.
- Focus design efforts on abstraction & modularity.

20



Things To Be Addressed In Planning

- Reifer's Steps
- McConnell's Keys
- Safety
- Software/Software and Software/Hardware Integration
- Portability
- Maintenance
- Re-Verification

21



2. Domain Engineering



22



What is Domain Engineering?

- Domain is a group or family of related systems. All systems in that domain share a set of capabilities and/or data. (Sodhi)
- Two Sides of Reuse:
 - Domain engineering → Developing for reuse
 - Reuse engineering → Developing with reuse
- Domain engineering is a developing field
 - it is still relatively immature

23



Some Concepts of Domain Engineering

- Knowledge reuse
- Repositories of components
- Reuse of architectural domain knowledge
- Reuse of software designs and patterns
- Reduction of “cognitive distance”

Cognitive distance is the intellectual effort required to take a software system from one stage of development to another (Girardi/Ibrahim)

24



3. Software Components



25



What Is A Component?

1. *Prewritten elements of software with clear functionality and well-defined interface. (Rhodes)*

2. *An atomic software element that can be reused or used in conjunction with other components; ideally, it should work without modification and without the engineer needing to know the content and internal function of the component. However, the interface, functionality, pre-conditions, and post-conditions performance characteristics and required supporting elements must be well known. (Lattanze)*

26



What Is A Component? (cont)

- 8110.RSC → Reusable software component (RSC) is the software code and its supporting DO-178B documentation being considered for reuse. It forms a portion of the software that will be implemented by the integrator/applicant.

27



Examples of Components

- Real-time operating systems
- Software libraries
- Loading software
- Communication protocol stacks

A component is a piece of software and/or data that can be “chunked” by itself.

28



Key Properties of a Software Component



- The component may be used by other program elements.
- The users and developers of the software component do not need to know each other. (Meyer)

29



3 Attributes of Software Components

- It is reusable
- It has clear functionality
 - Single purpose
 - Encapsulates related functions
 - Properly sized
- It has well-defined interfaces
 - E.g., consistent syntax, logical design, predictable behavior, & consistent method of error handling.
 - Complete, consistent, & cohesive interfaces

30



8 Qualities of Software Components (Meyer)

- Careful specification of functionality & interface
- Correctness – works as specified
- Robustness – doesn't fail if used properly
- Ease of identification
- Ease of learning
- Wide-spectrum of coverage
- Consistency
- Generality – useful for multiple environments

31



Component Library



- Contains software components/assets to be reused throughout a company
- Library should:
 - Provide seamless access to authorized users
 - Be searchable & browsable
 - Be integrated into the engineering environment

32



Component Library (cont)

- Items in the library should contain:
 - **Design narratives** – an overview of the component
 - All data that supports the component (e.g., plans, requirements, design, verification records, etc.)
 - **Design rationale** – detailed explanation of design decisions



33



Component Library (cont)

- **Design Rationale**
 - Communicates the design decisions and can help users determine if it meets their needs
 - **Internal Design Rationale** – describes internal interaction within the component
 - **External Design Rationale** – describes interaction of the component with the outside world

34



Component Library (cont)

- Aspects of Libraries to be Considered:
 - Format of components & assets entered into the library should be useful & consistent
 - Best utilization of search capabilities
 - Library management, operation, & maintenance

35



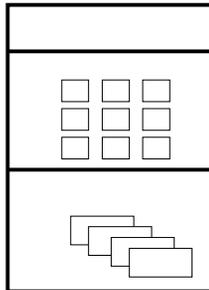
Components & Safety – Items to Consider

- Planning
- Traceability of requirements
- Re-verification
- Interface documents
- Partitioning/protection
- Artifacts
- Maintenance
- Unused code

36



4. Object-Oriented Technology

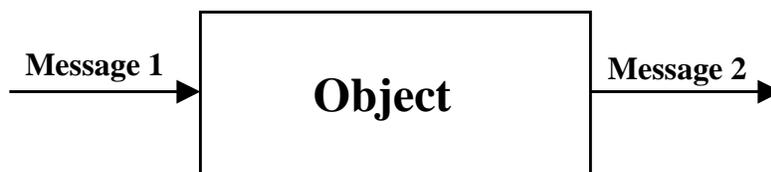


37



OOT Overview

- IEEE Definition of OOT: “A software development technique in which a system or component is expressed in terms of objects and connections between those objects”
- Centered around “objects” and “classes”



38

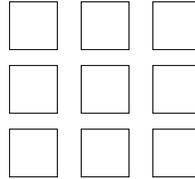


OOT Overview (2/7)

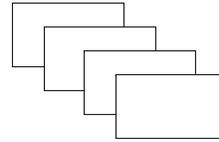
- Definition of Class: “a set of objects that share a common structure and a common behavior” (Booch)

Class Name

Attributes:



Operations:



39



OOT Overview (3/7)

7 Major Principles

Typical

- abstraction
- modularity
- concurrency
- persistence

Unique to OOT

- ****encapsulation**
- ****hierarchy**
- ****typing**

40



OOT Overview (4/7)

- **Abstraction:** *Helps to address complexity by providing crisply defined boundaries.*
- **Modularity:** *The process of partitioning a program into logically separated and defined components that possess defined interactions and limited access to data.*
- **Concurrency:** *Process of carrying out several events simultaneously.*
- **Persistence:** *Property of an object through which its existence transcends time and/or space.*

41



OOT Overview (5/7)

- **Encapsulation:**
 - The mechanism that binds together code and the data it manipulates
 - Keeps code and data safe from outside interference and misuse
 - Generally achieved through *information hiding*

42



OOT Overview (6/7)

- **Hierarchy:** The ordering of abstractions.
 - Examples of hierarchy: *single inheritance* and *multiple inheritance*
 - Sub-class “inherits” all of the existing attributes and operations of the original class, called the “parent” or “superclass”

43



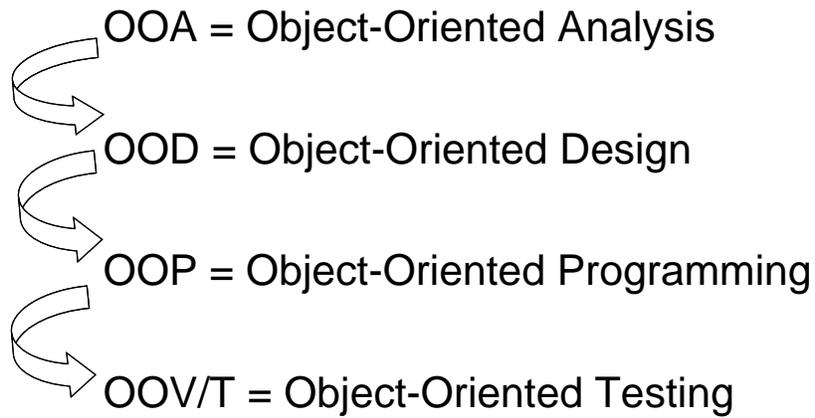
OOT Overview (7/7)

- **Typing:** Enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways
 - *Polymorphism* is a concept closely related to typing.
 - *Polymorphism* comes from the Greek meaning “many forms.”

44



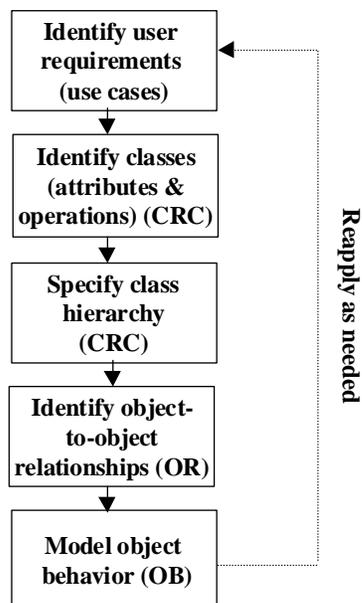
OOT Methodology



45



OOA



46



OOD

- Blueprint for software construction.
- Four layers of design are usually defined:
 - subsystem layer,
 - class and object layer,
 - message layer, and
 - responsibilities layer.

47



OOP

- Examples: SmallTalk, Java, C++, Ada 95
- C++ starting to be used in airborne avionics
- Some concerns: dynamic memory allocation, multiple inheritance, virtual base classes, run-time identification, templates, exceptions, and namespaces are deleted

48



OOV/T

- Process of detecting errors and verifying correctness of the OOA, OOD, and OOP. OOV/T
- Includes reviews, analyses, and tests of the software design and implementation

49



OOV/T

- OOV/T requires slightly different strategies and tactics than the traditional structured approach.
 - Because of inheritance, encapsulation, and polymorphism.
- Most developers use a “design for testability” approach to begin addressing any verification/test issues early in the program.

50



How OO Supports Reuse

- OOT helps to break complex systems into manageable pieces
- It's easier to implement OO design into code (using OO languages)
- OO model-based approach supports use of development tools

51



Safety Concerns of OOT

- ***Dead/Deactivated Code***
- ***Dynamic Binding/Dispatch***
- ***Encapsulation***
- ***Inheritance***
- ***Polymorphism***



52



5. Portability



53



Portability

- Goal of portability is transporting software to new platforms and/or environments with minimal adaptation.
- Portability is a desirable attribute for most software intended for reuse.

54



Portability Design Strategies

- Identify the minimum necessary set of environmental requirements & assumptions.
- Eliminate all unnecessary assumptions throughout the design.

55



Portability Design Strategies (cont)

- Identify specific environment interface required.
For each interface, either:
 - Encapsulate the interface completely in a suitable module, package, object, etc; or
 - Identify a suitable standard for the interface, which is expected to be available in most target environments.
- Anticipate the need to provide a software layer to “bridge the gap” for environments which don’t meet the interface assumptions.

56



Technical Considerations of Portability

- Classification
 - Classify complete applications according to their environmental interfaces & requirements
- Specification of portability requirements must be effective
 - **how much** portability is needed
 - **what kind of environments** will be used
 - **what costs** can be accepted to achieve portability

57



Technical Considerations of Portability (cont)

- Measurement
 - Ways to measure portability-based cost & effectiveness
- Design
 - Portability has significant impact on the design process
- Cultural Adaptation
 - Adapting to the conventions of new environments & users

58



Technical Considerations of Portability (cont)

- Verification & Validation
 - Verification activities, such as reviews, analysis, & testing are needed to ensure correctness in all applications & implementations.
- Common Problems With Portability
 - OS inconsistencies
 - Different compiler options/effects
 - Incompatible libraries
 - Run-time problems
 - Underestimation of integration effort
 - Architectural inconsistency

59

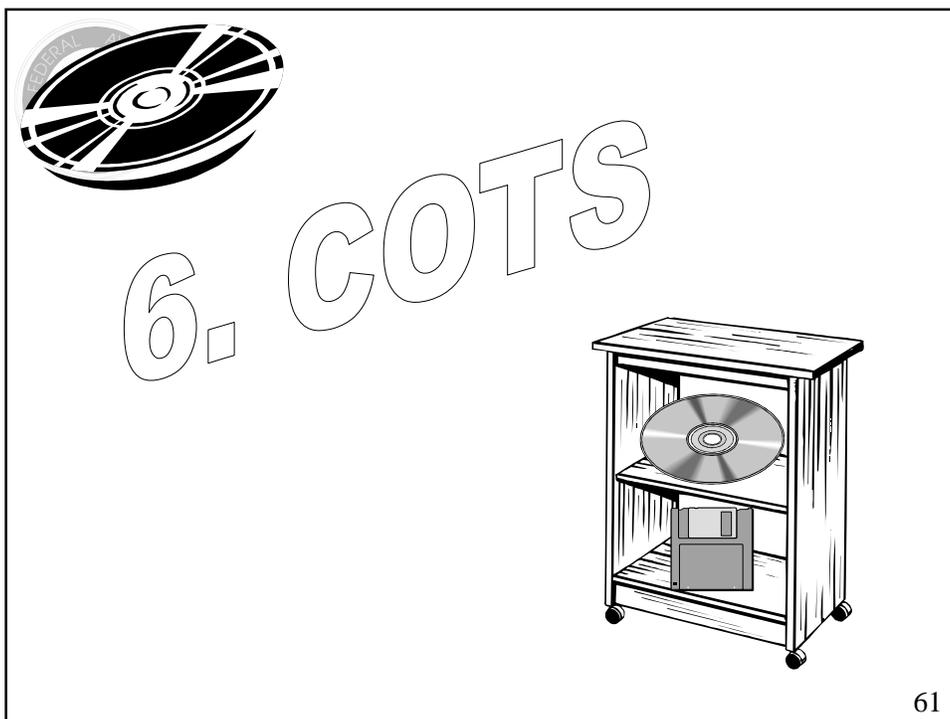


Real-Time Issues for Portability

- Timing
- Memory Allocation
- Memory Deallocation
- Dynamic Task Creation
- Scheduling Control
- Synchronization & Communication
- Events & Input/Output
- File Access



60



COTS Software Definitions

- **RTCA/DO-178B:** Commercially available applications sold by vendors through public catalog listings. COTS software is not intended to be customized or enhanced. Contract-negotiated software developed for a specific application is not COTS software.
- **FAA Research Report:** Any software product that is not developed within a given company for a specific application for that company. In particular, information regarding the software product's development and fabrication is not known or not available to the user of the COTS product. (Krodel)

62



Two Classes of COTS

- **Class 1** – Integrity Unknown
 - No access to software life cycle data
 - Level D
 - FAA Notice 8110.82 (now 8110.92)
- **Class 2** – COTS with Integrity
 - Developed using DO-178B
 - Software life cycle data exists
 - Potentially to Level A
 - FAA Draft Notice 8110.RSC

63





Product Service History Definition

- A contiguous period of time during which the software is operated within a known environment, and during which successive failures are recorded. (DO-178B)

65



Acceptability Depends On

- Configuration management of the software
- Effectiveness of problem reporting
- Stability and maturity of the software
- Relevance of product service history environment
- Actual error rates and product service history
- Impact of modifications

66



Attributes To Be Evaluated

- Service duration length
- Change control during service
- Proposed use versus service use
- Proposed environment to service environment
- Number of significant mods during service
 - Hardware mods & software mods
- Error detection capability
- Error reporting capability
- Number of in-service errors
- Amount/quality of service history data available and reviewed

67



Service History Conclusions

- Currently, it is hard to make a case for product service history.
- FAA is sponsoring research in this area.

68



Successful Reuse - Pulling It All Together



69



Characteristics of Organizations with Highest Reuse

- Use a product-line approach
- Utilize an architecture which standardizes interfaces and data formats
- Use common software architecture across product lines
- Implement a design for manufacturing approach
- Use domain engineering
- Have a defined software reuse process
- Management understands reuse issues.

70



Characteristics of Organizations with Highest Reuse (cont)

- Have software reuse advocate(s) in senior management
- Employ state-of-the-art reuse tools and methods
- Reuse more than just code (e.g., requirements and design)
- Trace end-user requirements to the components which support them

Rine/Sonnerman

71



Golden Rule of Reuse

- Before you can reuse something, you need to:
 - Find it
 - Know what it does
 - Know how to reuse it

~ Tracz ~

72



REBOOT – Reuse Maturity Model (RMM)

- **REBOOT = REuse Based on Object Oriented Techniques**
- **Implements 5 Levels Like the SEI Capability Maturity Model**



73



REBOOT – Reuse Maturity Model (RMM) (cont)

- **Level 1 – Initial or Chaotic**
 - No planned reuse
 - Only unintentional reuse occurring
- **Level 2 – Repeatable**
 - Project-to-project reuse
 - Limited scope
 - No overall reuse strategy

74



REBOOT – Reuse Maturity Model (RMM) (cont)

- **Level 3 – Defined**
 - Defined company-wide reuse strategy
 - Defined processes allow for reuse across the company
 - Company-wide reuse library
 - Each project is evaluated for reuse potential in accordance with the company's reuse strategy

75



REBOOT – Reuse Maturity Model (RMM) (cont)

- **Level 4 – Managed**
 - Reuse processes and reusable assets of the company are controlled and understood in detail.
- **Level 5 – Optimized**
 - Quantitative feedback
 - Continuously improve reuse processes & assets
 - Innovative ideas are evaluated and applied

76



REBOOT – Reuse Maturity Model (RMM) (cont)

- **Key Reuse Areas**
 - Reuse commitment
 - Project management
 - Asset management
 - Metrics
 - Development process

77



**FAA Reuse-
Related Activities**



78



FAA Reuse-Related Activities

- Notice 8110.Reuse
- Notice 8110.RSC
- COTS Research Project
- OO Research Project
- Service History Research Project
- Integrated Modular Avionics Team
- Plan for Tool Qual Reuse Policy
- SC-190 Activities

79



80



Summary

- There are many things to consider in reuse
- There are many techniques/tools to help
- The following issues must be addressed to be successful:
 - Top level management support
 - Modified development process
 - Overcoming non-technical inhibitors
 - Create an incentive program
 - Establish reuse measurements
 - Develop reuse guidelines
 - Focus on a single domain

81



Summary (cont)

- Safety must be a priority
- FAA has several initiatives underway to enable reuse

82