



**Australian Government**  
**Department of Defence**  
Defence Science and  
Technology Organisation

# Synthesising Complex Software Systems of Systems from Existing Software Components

*Stephane Collignon*  
*Professor Stephen Cook*

# Contents

- Introduction
  - System-of-Systems (SoS) definition
  - The Commercial-Off-The-Shelf (COTS) software products as SoS components
- SOS Software Inherits COTS Issues
  - Analysis of problem roots
  - Examples of mitigation strategies
  - Structural costs of implementing these strategies
- Candidate Solutions
  - Adaptation of the system environment
  - COTS products modification
  - Use of Open Source Software (OSS)
- An Optimal Solution
  - Application Programming Interfaces (APIs)
  - Principal decision elements
  - Examples of use of Open Sources Resources (OSR)
- Conclusion

# Introduction

- SOS definition (Maier, 1998):
  - Comprise components that are designed as independent entities
  - Display operational and managerial independence of the components
  - Tend to evolve with time through the asynchronous upgrading of components
  - Tend to be very large, e.g. air traffic control, military command and control systems

# Introduction

- The Commercial-Off-The-Shelf software (COTS) products as SoS components
  - Marketed as black boxes with no visibility of internal structure
  - Contain “hidden” code that may affect the component behaviour, thus the SoS behaviour;
  - Often designed as standalone products and as such not the ideal building block for SoS
- We shall now focus on the detail of these issues.

# SOS Software Inherits COTS Issues

- Analysis of problem roots
  - Defence market too small for software developers (Ciufu, 2003 ~ 0,1%)
  - Therefore COTS products traditionally developed for cost-sensitive market that conflicts with the imperatives of the professional environment
- Next slide gives an overview of these differences.

# SOS Software Inherits COTS Issues

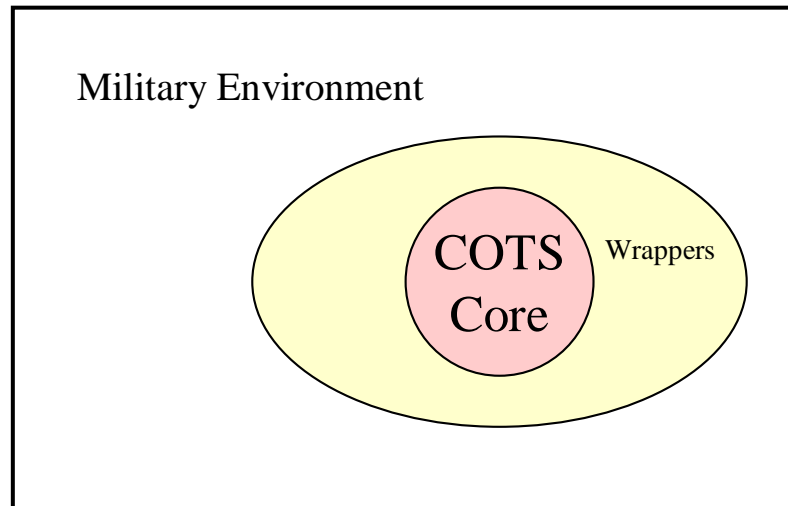
Software Attribute	Characteristics of COTS software in a Domestic Environment	Characteristics of COTS software in a Professional Environment	Alignment
Cost	Important	Secondary	Yes
Reliability	Secondary	Vital	No
Support	Restricted to 'How To Use'	Ability to get more than 'How To Use' is Essential	No
Information on Package Internals (I/O, interfaces..)	Rarely available	Essential	No
Availability of Package Functions through APIs	Rarely available	Essential	No
Code Integrity and Trust Level	Irrelevant	Primary	No
Duration of support	Warranty only, at most a few years	Life-Cycle Critical	No

- Hidden code More than Excel'97 Flight simulator: What are Easter Eggs?  
In terms of computers, Easter Eggs have nothing to do with chocolate. They are in fact hidden code within applications and operating systems. This code has been put in by the programmer just for fun or as a "calling card". Easter Eggs can take many forms, including scrolling credits lists, hidden games, pictures, and bizarre occurrences.  
<http://www.dhodrien.pwp.blueyonder.co.uk/Eggs/eggs.html>

- COTS is not comprehensibly testable
  - Undocumented code
  - Little documentation on internal structure
  - Internal modules not separately testable
  - No visibility of design process and process products and hence not able to judge code integrity
- Some mitigations strategies exist to improve these shortfalls

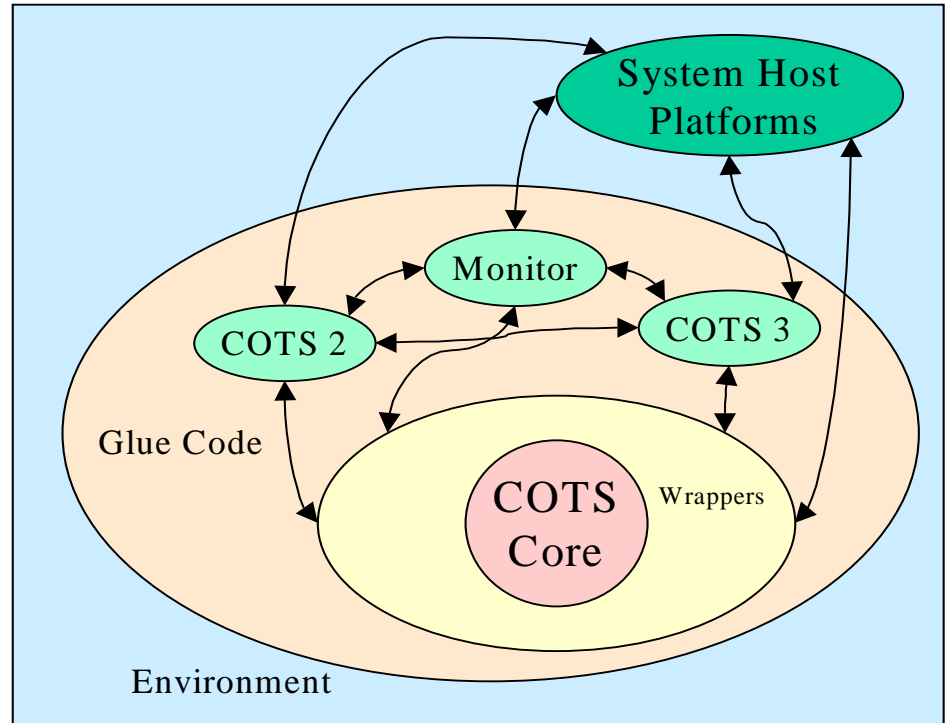
# Examples of Mitigation Strategies

- Reverse-engineering
- Wrappers implementation (White, 2000)



# Examples of Mitigation Strategies

- Addition of Glue-code, to synchronize the COTS packages
- Incorporates monitoring tools and wrappers



**A SoS product synthesised from COTS products.**

# Examples of Mitigation Strategies

- Costs associated with the implementation of the strategies:
  - Deep knowledge of the operating system of the computer host
  - Enhanced trouble-shooting skills
  - Reverse-engineering skills
  - Knowledge of low-level language
  - Software engineering skills, to perform risk evaluation and
  - Advanced software developmental and coding skills to implement wrappers and glue code

# Candidate Solutions

- Problem solving is cultural and technical (NATO, 2000):

PAPERS	PROPOSED SOLUTION	PROPORTION
1,11	Error Control Implementation	8.3%
15,20,21	COTS product modification	12.5%
2,4,5,6,7,8,9,10,12,17,18,19,22,23,24	Use of standards, Procedural Evaluation and Selection	62.5%
13,16,	COTS Code Examination	8.3%
1,3	Use of Open Sources Software	8.3%

79%

**Table 2: Grouped distribution of the proposed solutions (Collignon&Cook,2002)**

# Candidate Solutions

- Problem solving solutions types:
  - *Adaptation of the system environment to accommodate the errors*
    - Examination of COTS product source code
    - Use of standards, Procedural Evaluation and Selection
    - Error Control Implementation
  - COTS products modification
  - Use of Open Sources Software pros & cons

# Candidate Solutions

- Open Sources Software pros:
  - Availability of the source code implies
    - Informal testing is made on a wide range of platforms
    - Easy modification of the products
    - A cadre of knowledgeable professionals exists that are familiar with OS products and their characteristics
    - Comprehensive quality awareness through code reviews undertaken by the substantial user community
    - Continuous access to Open Sources resources at zero cost
    - Control the software sources prevents withdrawal of support mandating unwanted upgrades.

# Candidate Solutions

- Open Sources Software cons:
  - Mainly on the developer's side:
    - No commercial imperatives driving the creation and maintenance of open source software
    - Community has no pre-allocated budget for Open Sources Products hence
    - Product release is on a ad-hoc basis and there are never any guarantees of delivery time of quality

*How can we capitalize on the use of both COTS and Open Sources Resources?*

# An Optimal Solution

- Bottom-up, design-to-inventory approach
- Evaluate:
  - COTS products against SoS requirements
  - Also evaluate open-source code
  - Developmental approach for modules
- Design SoS architecture
  - Several candidates based on a variety of approaches
  - Consider interfacing and the design glue code for each
  - Select the preferred candidate based on value analysis against key requirements
  - Verify that it will have the properties required, in particular robustness to failure, supportability, integrability, performance
- Integrate selected components into the final SoS.

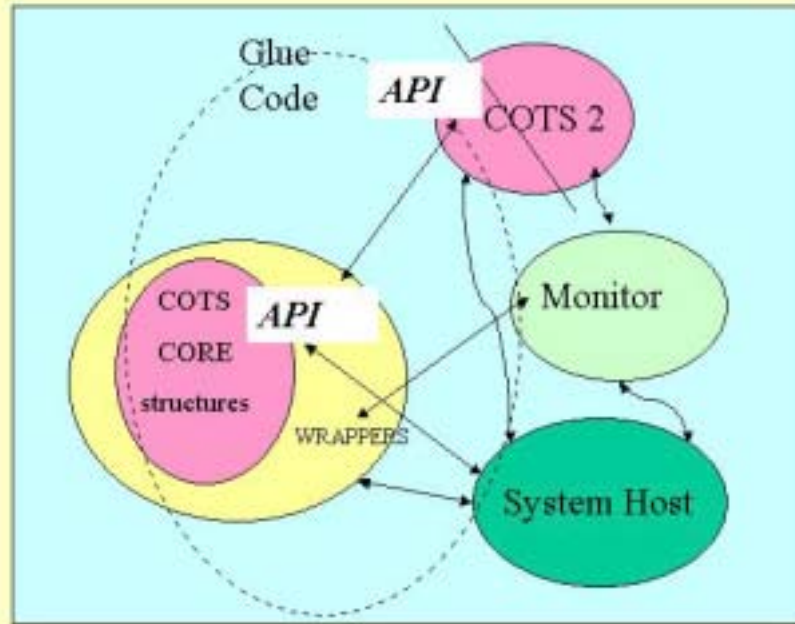
# An Optimal Solution

## The Importance of Interface Management

- Software products allow the implementation of high quality interfaces through the use of APIs
- These APIs are critical to the communication with COTS and non-COTS software components
- Well-managed interfaces are critical to the success of the system (Maier, 1998)
- Use of APIs is illustrated in the next slide:

# An Optimal Solution

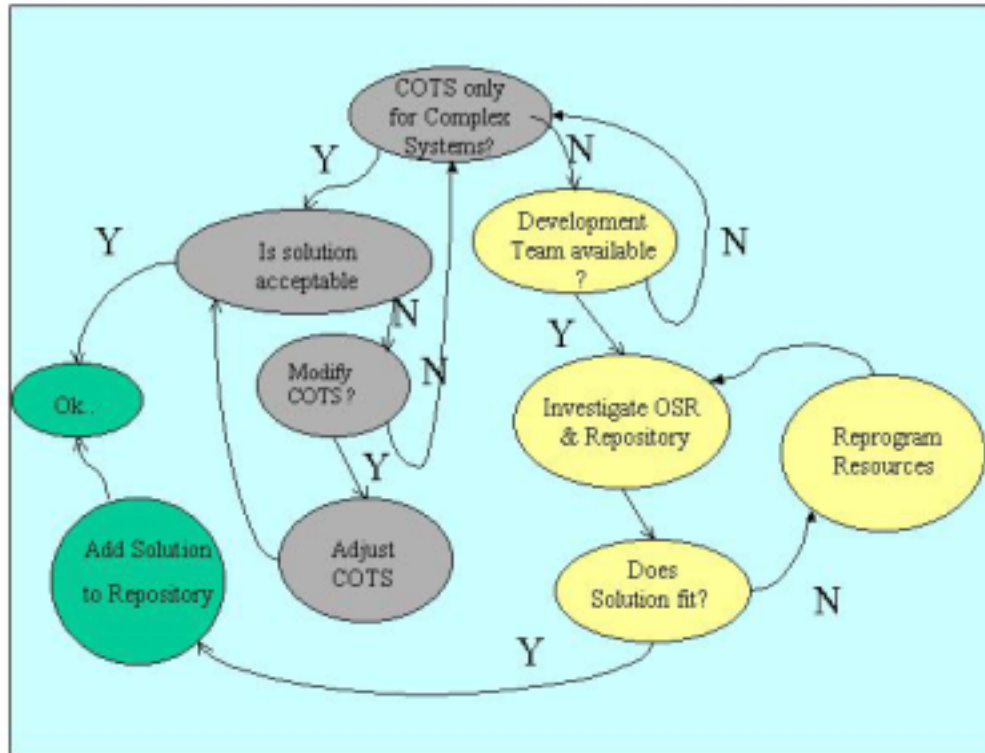
*It is necessary to have high quality system components and robust and well-documented interfaces*  
(Maier, 1998).



Implementation of the APIs in COTS

# An Optimal Solution

## Selection of Components to Form Candidate SoS



The decision shown will contain an important organisational resource component: a solution repository. The consequences of the decision on the creation of this component will influence the quality of the decisions taken after the first iteration of this cycle.

### Selection process between COTS and Open Sources

## *Some Examples of OSR Use*

- UK: GNU C Compilers modified to produce parallel architecture code
- Germany: substitution of Windows platforms by Linux
- USA: One of the FOSS report recommendations is selective approval of OSR (MITRE, 2003)
- Japan: Use of Open Sources resources to integrate their IT activities in their industry

# Some Examples of OSR use

Country	Area-of-Interest (AOI)	Ability to generate systems	Importance Of the AOI	COTS Usability	Principal Activity In the AOI	Next step of the Open Source Initiative
UK	University	Yes	Marginal	No	Promotion	Commercial
Germany	Small Business	Yes	Substantial	No	Substitution of COTS	Diffusion
USA	DoD	Yes	Primary	No	Alternate Solution	Evaluation
Japan	Industry	Yes	Integration	No	Control & Quality	Production

## Diversity of areas-of-interest for OSR users

# Conclusion

- Architectural design takes on even greater importance when integrating existing software products
- Consider use of Open Source software as an alternative to the sole use of COTS software products
- A higher level of skill is required to integrate COTS and Open Source software than is often realised
- Special techniques are needed to build reliable systems of systems.
- Education must be tailored to these needs
- Need to create a pool of developmental expertise and resources (business opportunity?)



*Thank You*

*Questions ?*