



Standards for Software Transparency^(*)

Case Study: Air Traffic Management

Judith Klein, Lockheed Martin Fellow

(*) Openness, understandability of design, traceability to requirements and tests

Air Traffic Management Case Study



- **Most systems we build**
 - Have long life cycles
 - Are mission-critical, safety-critical
 - Are high-assurance, real-time
 - Have large amounts of software content
 - Have high reliability
- **Handling “legacy” systems is a way of life; helpers:**
 - Applying sound system architecture, software architecture
 - Hierarchical, clean (simple, one-way) dependencies
 - Encapsulated components expose multi-language Application Programming Interfaces, hide details

Every day across the globe, 60 percent of the world's commercial air traffic is monitored and controlled by Lockheed Martin air traffic systems.

Motivations for Retaining the Legacy Languages

- **Large body of reuse (Non Developmental Items) in Ada, C,C++, Perl, Shell, dxi, PL/SQL, VB, XML, ...; Java added to the mix**
 - “Best” language used for each problem (online vs. offline)
- **Ada: positive experience**
 - Standardized, strongly-typed language promoting good software engineering principles (safety just one aspect)
 - Facilitates interface to other languages
- **C++:**
 - Facilitated use of Commercially Available Software (e.g., xlib, Netview) and reuse (e.g., Lockheed Martin-developed ViewMan display library, Simple Network Management Protocol – SNMP and Management Information Base – MIB)
 - Mixed language environment proved workable
- **Java added primarily for Service Oriented Architecture**
 - Customer-mandated products requiring Java technologies

Targeted re-write driven by technological, efficiency, synergy needs



Promoting SW Integrity, Security, Reliability, and Resilience

- **Language aspects** that promote integrity, security, reliability, and resilience in software are similar in most languages
 - E.g., a buffer overflow causes problems of integrity, security, reliability, resilience... Therefore must be avoided , prevented, detected – details are language-specific
- **Strongly typed languages** help by both preventing classes of errors (compile-time) and detecting classes of errors (run-time)
- **Using tools** to analyze code against prescribed standards of behavior helps automate the otherwise laborious process of inspection
 - E.g., Static analysis tools, Run-time memory leaks' detection
- Languages laden with features, more or less safe/ understood/ misused
 - **Language Independent Guidance**
 - Code Complexity, concurrency, memory control, packaging considerations, problem determination, ...)
 - **Language Style Guides**
 - Some language features are restricted for use
 - **Lessons Learned** (including programming for performance, both language-independent and language-specific)

Establish language-specific guidelines/ checklists, use tools to enforce

Assurance of Legacy Systems

- **Legacy Systems have field history, statistics**
 - Typically proven to work
- **Helpful to have key personnel who architected, designed, developed, debugged, documented the system now referred to as “legacy”**
 - Documented processes used, objective evidence retained
- **RTCA DO-278 Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management Systems (CNS/ATM) Software Integrity Assurance – newly mandated**
 - Heightened needs for data integrity can be addressed by a combination of COTS vendor/product compliance , checksum enveloping while the data is processed by legacy system, human-in-the-loop verification.

Assess process compliance against revised requirements, close gaps

Improving the State of Legacy Systems

- **Some legacy systems are better off being replaced**
 - Inadequate (users needs not satisfied), poorly documented (not maintainable), convoluted (brittle), COTS products can be used to replace large portions (e.g., developed message delivery system when so many inexpensive alternatives abound), etc
- **Systems with long life cycle, can become “legacy” as soon as they are deployed**
 - May meet requirements, user needs, be extensible, etc.
 - “Legacy” does not have to be a “bad” word
- **Assurance can play a big part in improving the state of legacy systems**
 - Promoting good engineering practices, maintaining current/correct, traceable life cycle products, applying rigor in methodology
- **Strive to achieve independence of platform, of tools vendors**
 - At least isolate and minimize dependencies

Transparency and traceability are required in legacy systems for both modernization and assurance/compliance



Judith Klein

Fellow, Transportation Solutions

9211 Corporate Blvd

Rockville, MD 20850

301-640-2790 judith.klein@lmco.com

Subject Matter Expertise

System Architecture (Certified System Architect), Software Architecture - large, distributed, near real time systems, focused on aspects of fault tolerance and system performance.

Customers, Programs, Domains Supported

Main customer is the Federal Aviation Administration (FAA). Primary program is En Route Automation. Domain: Air Traffic Management, CNS/ATM.

Other Activities

Adjunct faculty – The Johns Hopkins University – teaching Systems Engineering to ELDP. George Washington University External Advisory Board. Senior member IEEE, member INCOSE, ACM.