

# Challenges of Establishing a Software Product Line for an Aerospace Engine Monitoring System

Tim Kelly, Ibrahim Habli  
*Department of Computer Science  
University of York*

Ian Hopkins  
*Control Systems Engineering  
Rolls-Royce plc*

## Outline

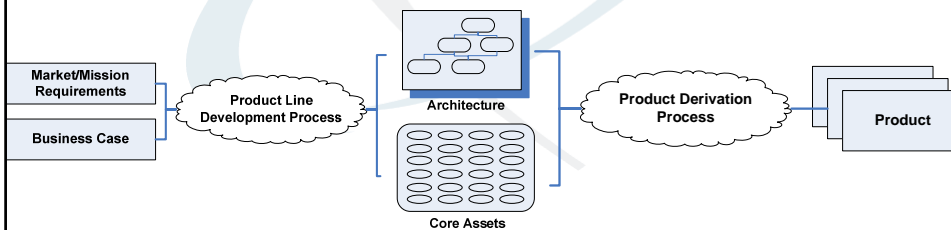
- Why reuse?
- Dangers of reuse in critical systems
- Product lines and critical systems
  
- Case Study
  
- Further challenges and work

# Context

- Developing Safety Critical Software is expensive
  - 1 Verified LOC costs \$150-\$250
  - Complete development of 100 kLOC ≈ \$25M
- Reuse is a sensible approach to reducing the development cost of critical software
  - Develop it once and reuse in multiple systems
- But ... Software reuse has a bad reputation in the critical systems domain
  - ARIANE 4 (Realignment function) -> ARIANE 5 (No need, yet retained)
  - Therac 20 (Hardware Interlocks) -> Therac 25 (No Hardware Interlocks)
- Naïve assumption that reuse 'builds upon' previous op. record
- Reuse may even decrease reliability
  - Seemingly trivial or unrecognised variation may result in new hazards

# Product Lines

- Reuse can be more systematic and trustworthy if **planned** from early development stages
  - Not opportunistic, ad-hoc reuse '*mining*'
  - Traceable link(s) between reusable requirements, design, and code
- Product line development as a holistic approach
  - Predefined and planned processes
  - Large-scale reuse
  - Predefined architecture and associated constraints



## Product Line Concept

- The term **product lines** refers to engineering techniques for creating a collection of similar systems from a shared set of assets, using a common means of production
- Similar does not mean identical
  - **Commonality** – what makes it a family
  - **Variability** – what makes a family member distinctive
- Concept is general
  - Considerable interest in *software* product lines
  - Software is the primary determinant of function for many systems
    - Increasingly determines variations between individual products
  - Software an enabler, but need commonality in the **system**

## Software Product Line

- Clements and Northrop definition:
  - *“A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market sector or mission and that are developed from a common set of core assets in a prescribed way”*
- **Key concepts:**
  - Many products very similar – planned rather than by accident
  - Core assets - £ Benefit (but have to ask where £ is currently)
  - Select components from assets
  - Tailor assets
    - Mechanisms pre-planned (e.g. parameterisation)
  - Assembly rather than generation
  - ‘Prescribed way’ implies **architecture**

## Example Product Lines

- **General Motors Power Train (GMPT)**
  - 5 engine control modules
  - 4 transmission control modules
  - 3 power train control modules
  - 17 software builds reduced to 3
- **Cummins diesels**
  - Large range of diesels, of several types
  - Sizes vary from 4-164 litres
  - Total of 1000 variants of 20 basic builds
- **Engineered products**
  - Opportunity for control system PL as embedded in a broader PL

## Examples of Benefits

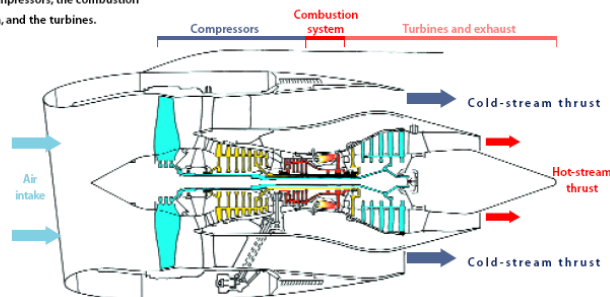
- Rare to get hard data on benefits, but some examples
- **Cummins diesels**
  - Productivity improvement of 3.6
  - RoI of 10:1
  - Rapid move from vehicle to industrial diesels (secure new market position)
- **Bosch (automotive electronics)**
  - Configure software for new ABS in a product line within days
    - e.g. new variant of Renault Clio

# Aero-Engine Control Product Lines?

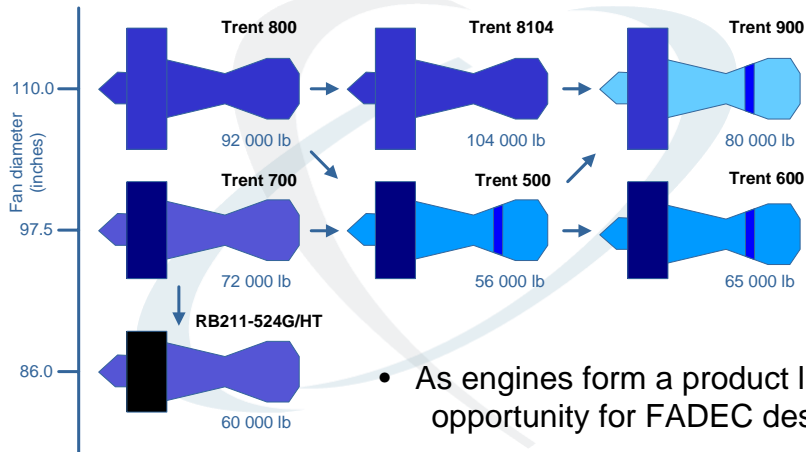
## Full Authority Digital Electronics Control (FADEC) System

- Modern civil aircraft gas turbine engines are controlled by a Full Authority Digital Electronics Control FADEC system (FADEC)
  - A high-integrity embedded computer system controlling engine operation
- The FADEC monitors and controls performance parameters of the engine
  - Provides warning to the Airframe including the pilot and the maintenance team

The gas turbine has three main sections: the compressors, the combustion system, and the turbines.



## Engine Product Line



- As engines form a product line, opportunity for FADEC design

## Commonality and Variability

- Overall function the same
  - Reverse thrust to slow aircraft



- Mechanisms different
  - Design to exploit commonality, and ease variation

## Observations

- For FADECs, “context” conducive to product lines
  - Embedded in a wider system (product line or set of product lines)
  - High degree of commonality
  - Long-term stability in core functions
- There are some particular challenges
  - Need to achieve **certification**
    - Safety process might undermine family design/reuse
  - Much of the cost in is **verification**, not construction
  - Need to manage “multiple customers” (see next slide)
    - Airframer, airlines, certification authorities
- Realising benefits requires challenges to be overcome

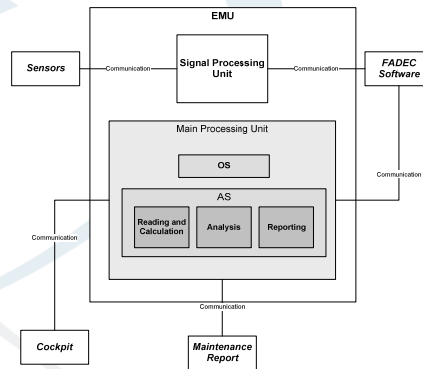
## Example: Customer Interface Management

- Marketer: “You know, if you were to relax this requirement over here and drop that little one over there and change this one over here just a bit then we could build you a system that’s in our software product line.”
- Customer: “I see. And this matters to me because ...?”
- Marketer: “Because if we build your system from scratch it will cost you \$4M, take two years, and your software will be unique. If you take a member of our product line, it will cost you \$2M, be ready in six months, and your software will be the same as is running reliably for 16 other customers. But, of course it’s entirely up to you.”

(Clements and Northrop example)

# Aerospace Engine Monitoring Systems

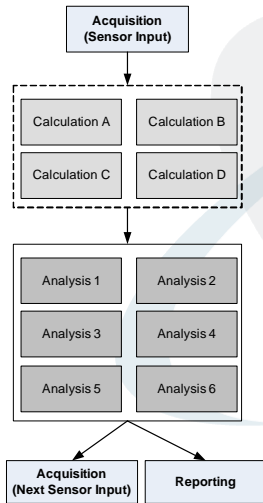
- Engine health parameters are monitored by an **Engine Monitoring Unit (EMU)**
- An EMU is an engine mounted unit comprising two processing modules
  - Signal Processing Module (SPM)
  - **Main Processing Module (MPM)**



# EMU Product Line

- Incentive – Two similar projects:
  - **Commonality**: New project requiring common engine monitoring functions to previous project
  - **Variation**: As a result of differences in:
    - Fine-tuning (configuration)
    - Project specific communications
- Advantage
  - Cost effective development and maintenance of initial two and future EMU software

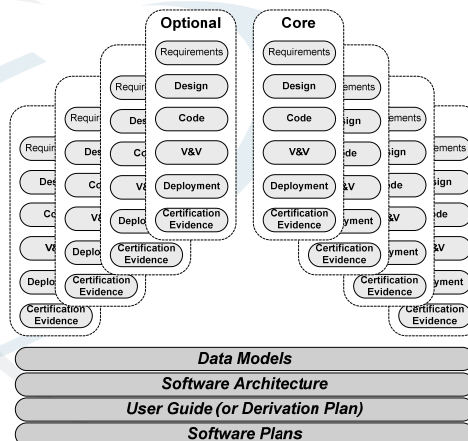
## EMU Software Application Elements



- Two EMU software applications can only differ in the number of analysis functions that they provide
- Calculation, Analysis and Reporting
  - Division carried through to the requirements
  - E.g. 4 requirements docs for calculation

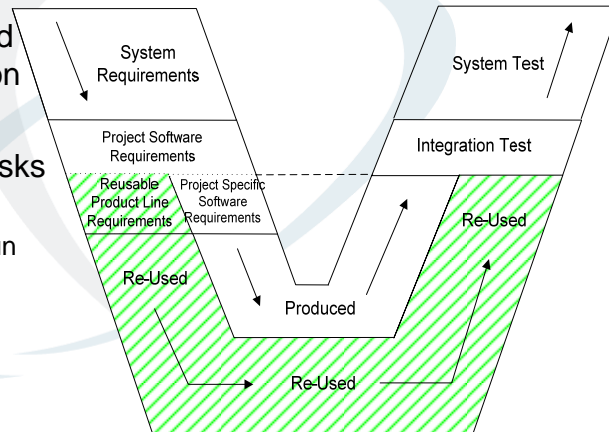
## Product Line Repository

- **Management assets**
  - Software plans
- **Development assets**
  - Requirements
  - Software architecture
  - Low-level design
  - Source code
  - Review records
- **Verification assets**
  - Test cases
  - Requirement-based test scripts
  - Requirement-based test results
  - Review records
- **Configuration assets**
  - Software configuration index



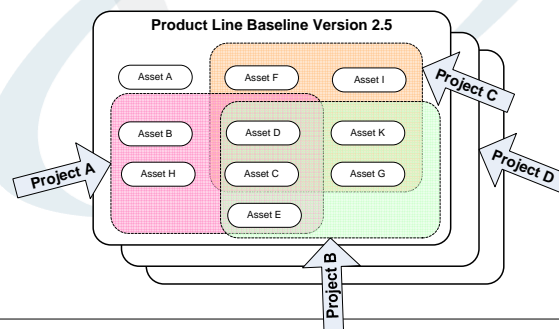
## Derivation Plan: User Guide

- Includes timing and memory information
- References plans
- Includes a log of risks and assumptions
  - E.g. how long to run before reset
- Derivation and Integration are still overheads



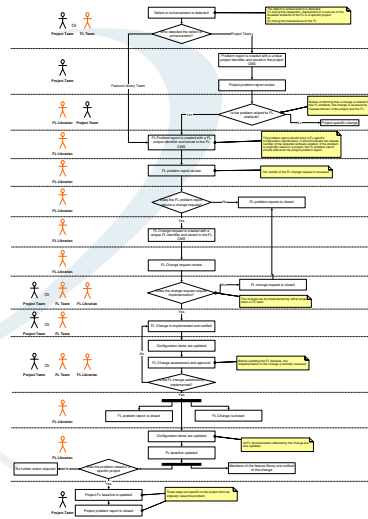
## Configuration Management

- Configuration Management emphasised in DO-178B
- Ownership of reusable assets
- Two configuration management dimensions
  - Across time: product line asset versions
  - Across projects



## Configuration Change Control

- Managed centrally by a librarian
- Governance
- Process for implementing a product line change
  - Change identified by product line team
  - Change identified by project team
- Process for propagating a change into individual projects



## Approval Challenges

- DO-178B guidelines are highly prescriptive
  - (see next slide)
  - Mandate or recommend activities, techniques and methods
  - Implicit assumption that the software is developed from scratch!
- EMU product line repository provides all applicable DO-178B lifecycle data
  - No project requirement to create lifecycle data for product line artefacts

# DO-178B Development & Verification

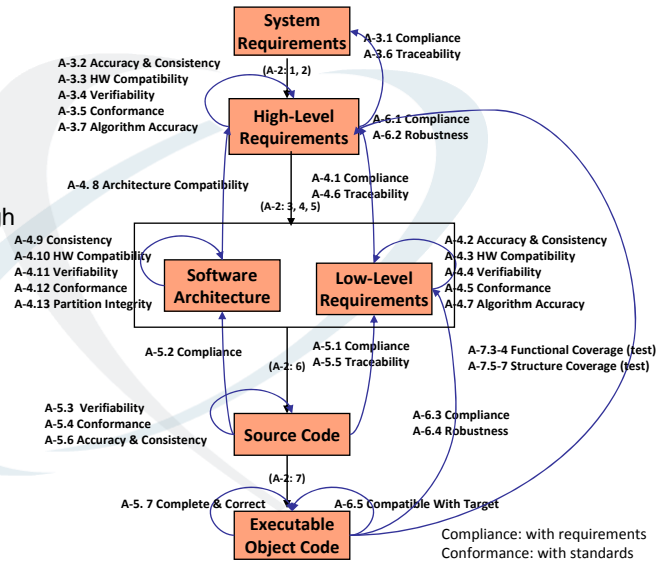
- DO-178B defines the software development process

- **Main stages**

- Software requirements (High Level, Low Level)
- Software design
- Software coding
- Integration

- **Emphasis:**

- Traceability
- Human review of requirements, etc
- **Testing** for verification



## Product Line Assets = Previously Developed S/W?

- Do (reused) product line assets have to be considered to be *previously developed s/w*?
- DO-248B
  - “PDS is defined as software already developed for use. This encompasses a wide range of software, including commercial off-the-shelf (COTS) software through software developed to previous or current software guidance.”
- Intention to use Prod Line must be stated in PSAC
- Certification responsibility of individual project
  - Arguments and Evidence concerning integration
  - Safety is not a “sum of parts” issue

## Incremental Acceptance in DO297

- DO-297 - “**Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations**” (2006)
- Adapted from FAA advisory circular for “Reusable Software Components”
  - Permits reuse of certain evidence about an IMA Platform or Application if certain criteria met
    - E.g. if context is the same MC/DC testing results could be re-used
    - Note - assumptions about process based evidence but product based context, evidence divided up for components
- Caveats:
  - Does NOT constitute certification of an item
  - First attempt to gain credit must be part of an actual system certification

## Challenges

- Assurance of the **process** underlying product line’s core assets
- Relationship between **hardware** and **software** product lines
- Relationship between **reuse** and **unreachable** code
- Encapsulation of **safety analyses**
  - Treated as reusable product line assets
- Acceptability of **incremental certification**
  - Reuse of pre-certified elements reduced re-certification effort

## Related Ongoing Work in HISE

- MOSAIC project
  - Project concerned with V&V cost reduction
  - Looking at composition of (Pre-)Verified Software Components
  - New RA from RODIN project
- Industrial Avionics Working Group
  - Modular Safety Cases Certification work
  - Modularity in Evidence (GD)
  - Identifying and Managing Dependencies (RDA)

## Summary

- Software Products are attractive given potential benefits
- But, introduction of a product line in high-integrity software development poses great challenges
  - Challenges assumed lifecycles
  - Challenges business 'project' models
  - Requires tight control – configuration management, management of assumptions
  - Can we (safely) carry verification evidence with assets?
- **Way forward:** embedding reliability and safety analysis and assurance into the product line plans and lifecycles
  - Making analysis results and arguments reusable according to predefined variation points