Embracing Product Line Engineering for Spacecraft FSW

December 13, 2016

Michael Phillips, Lockheed Martin Chief Software Engineer
Michael McKenzie, Lockheed Martin, Product Line Chief Architect
Adam Johnson, Lockheed Martin, Product Line Manager

Space Software Drivers: Current Challenges

Current Challenges

- *Current software for space systems has faced both development challenges and customer pressures for predictable and affordable software*
- Typical process improvement activities that save 10% - 30% will not meet our challenges
- Software Development is a mostly manual processes
  - Produces high quality and performance for point solutions
  - Lots of variation in details of production from program to program
  - Little cooperation among program production teams
- Clone and own does not scale to an enterprise-wide solution
- Internal investment across competing solutions for scarce funding is unsustainable
- Artifact re-use approaches used in several programs are still inadequate to meet goals

Space Software Drivers: Future Challenges & Opportunities

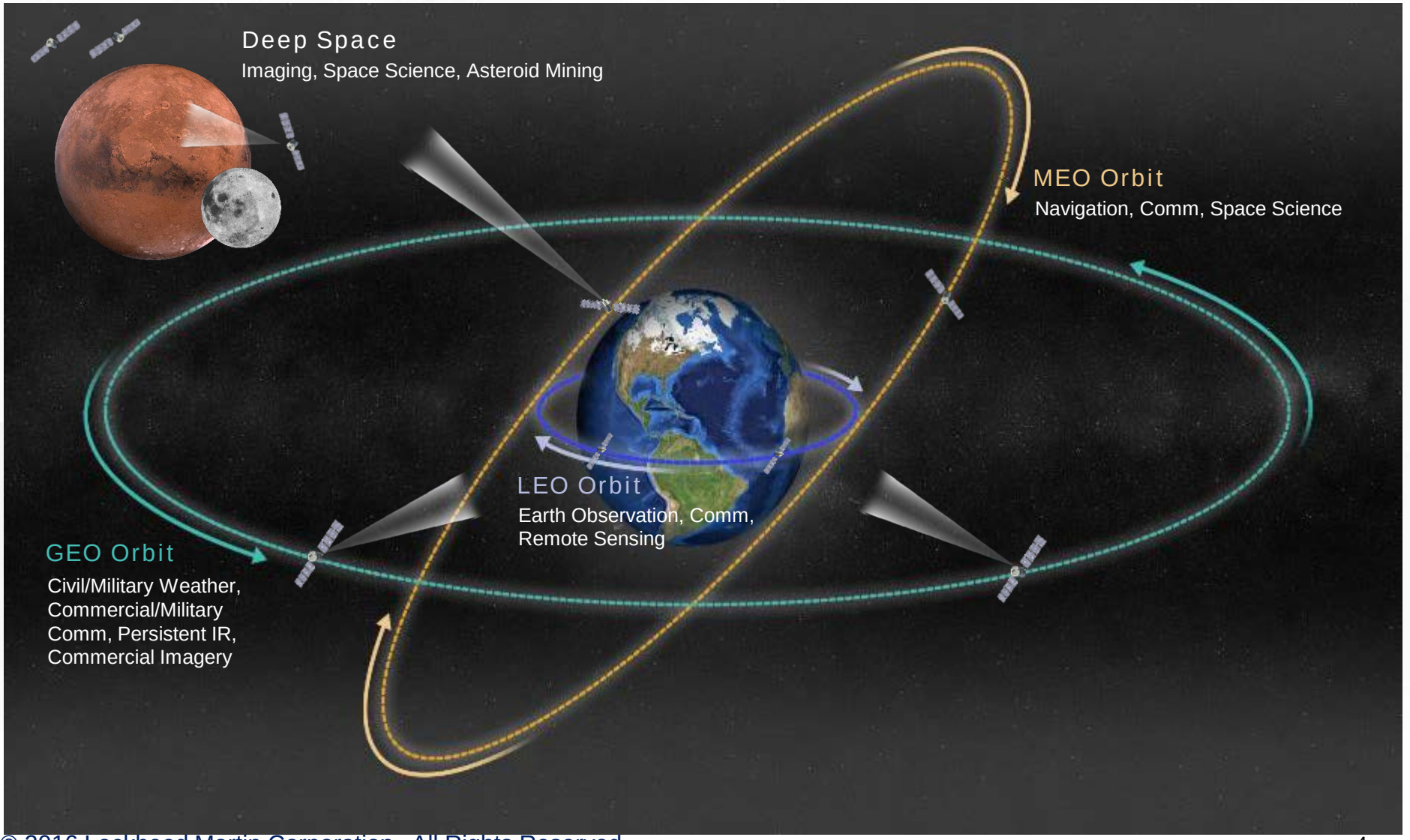
Future Challenges

- *Future space systems will require significantly more software to provide the required capabilities to support mission needs*
  - Software is a key enabler of future mission needs for our customers

Software Product Line Opportunity

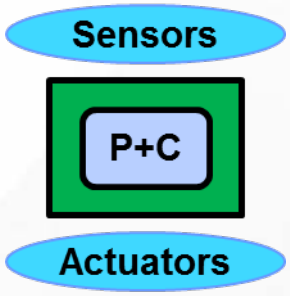
- *To be competitive, most product development organizations deliver a product line – a portfolio of similar products or systems with variations in features and functions*
  - Required Mission variation does not support “one size fits all” approaches
  - We need to embrace natural variation without exceeding our cost targets

Breadth and Depth of Current/Past Spacecraft Missions

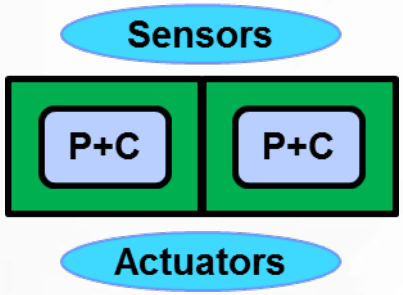


Avionics Architecture Challenges

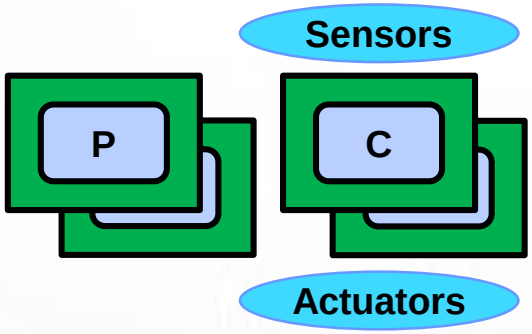
CURRENT



Small Single-String Centralized Electronics

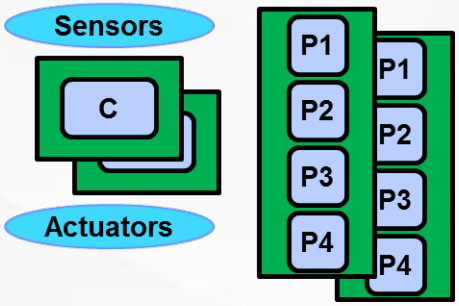


Dual-String Centralized Electronics

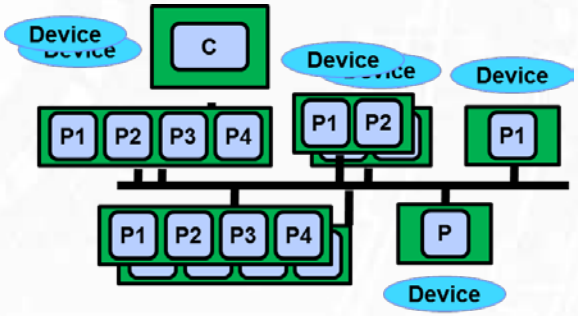


Dual-String Distributed Electronics

FUTURE



Dual-String Centralized Multicore Electronics



Networked Electronics with Distributed Processing Nodes

Legend
P – Processor Card
C – Circuit Cards

Selection Approaches for Establishing a Software Technical Baseline

Software Product Line Engineering (SPLE)

- Reuse assets designed explicitly for reuse
- Product Line treated as whole, not multiple products
- Core product with component-based architecture
- Enterprise product, people, tools, processes

Product Development Organization (PDO)

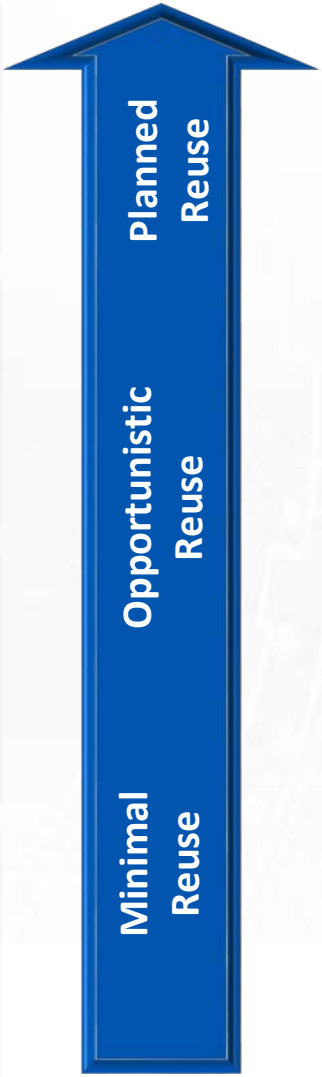
- Leverages heritage product & people for each mission
- Modify, add new, field, and maintain centrally
- Best executed NRE FSW Costs for LMSSC
- Reuse typically 60%-80% (other is mission-unique)

"Clone and Own"

- Select baseline with best applicability ("borrow")
- Modify, add new, field, and maintain separately
- Reuse typically 30%-75%
- Product team or Program team

Establish New Implementation Baseline

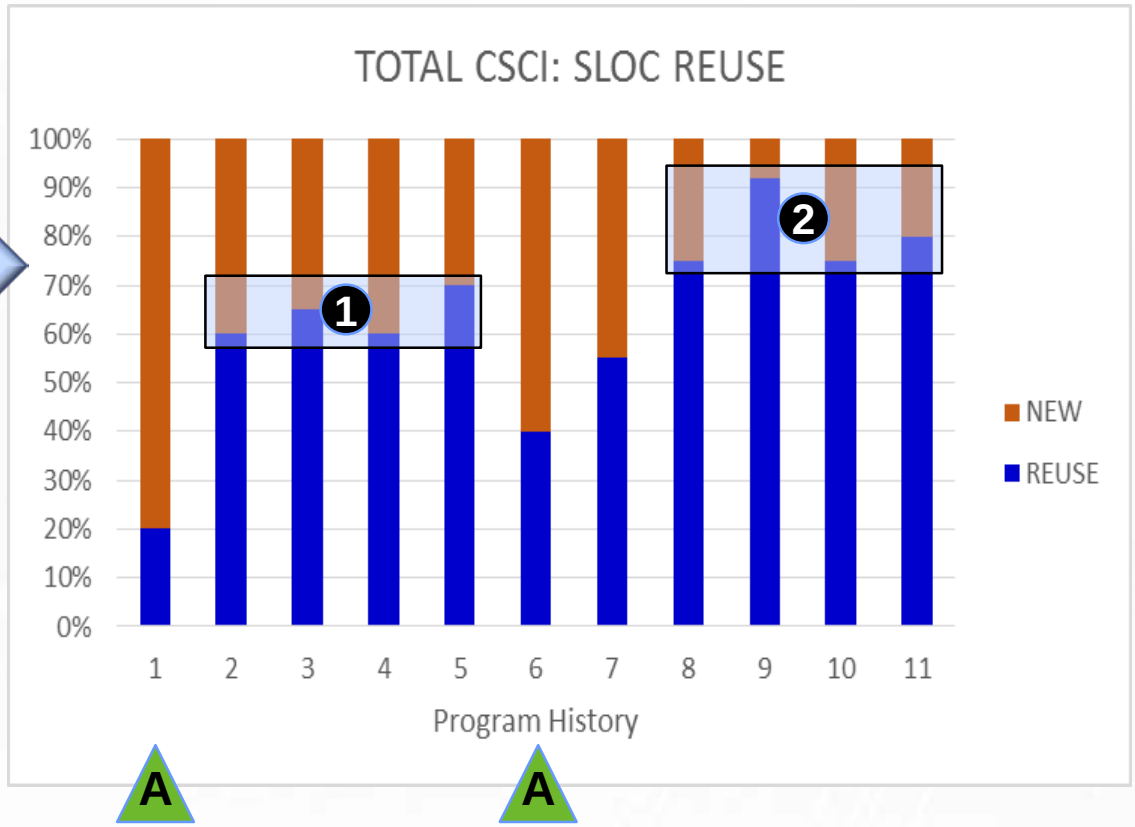
- Create new code baseline with some design heritage
- May be driven by SW Obsolescence (e.g. Ada to C++)
- Design reuse <50%, Code reuse <10%



Achievable Results: FSW Product Development Organization (PDO)

Example Product Line Reuse

- **A** Architecture update
  - Program 1 & 6
- **1** Good Reuse* (60%-70%)
  - Programs 2 - 5
- **2** Great Reuse* (75%-90%)
  - Programs 8 - 11



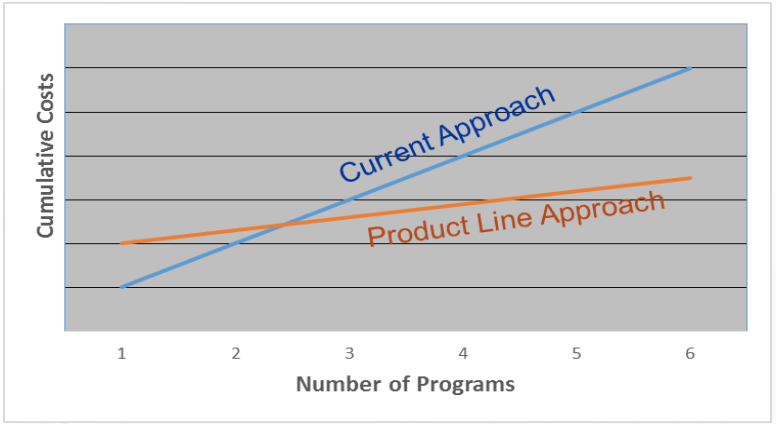
***REUSE is defined here as either: 100% untouched software file, OR a slightly modified file (<20%)**

Leadership in the PDO has driven high reuse of the platform, avionics, and software. New development is expected for new capabilities

Product Line Business Value for Flight Software

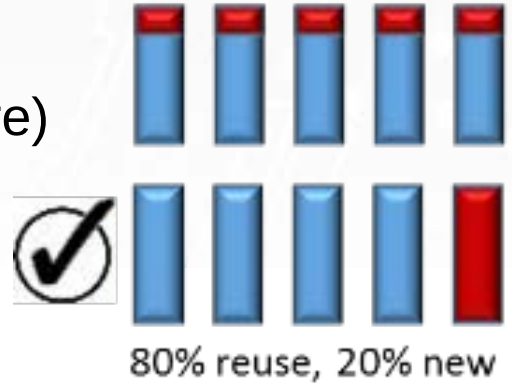
Organizing to bring business value to the development, delivery, and support of an evolving product-line leads to:

- Control diverse product configurations (embrace natural variation)
- Faster times to product delivery
- Faster turnaround on small modifications
- Higher software product quality & reliability
- Lower overall development costs



These characteristics are attained primarily due to:

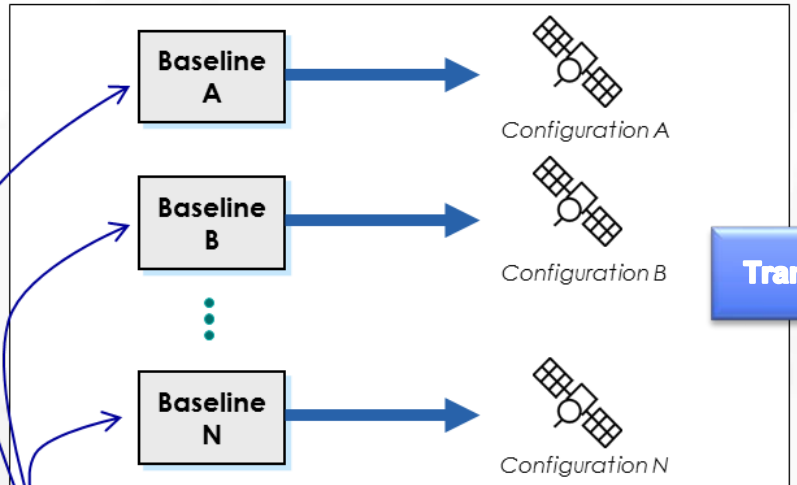
- Business model for product line engineering
- Pre-planned strategic reuse (architecture is at the core)
- Maintaining a Core Asset base (Requirements, Components, Tests, etc.)
- Product ownership that is independent of programs



Changing the Paradigm for Software

Independent Program Development

Multiple FSW Baselines to Support Multiple Configurations

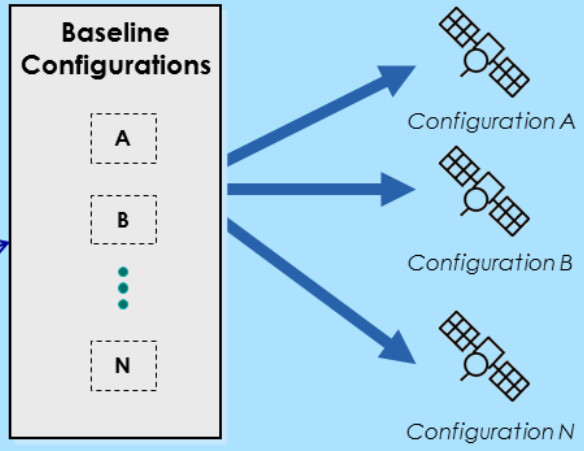


- Independent Program Management Structures
- Multiple Review Teams (Internal/External)
- Independent Development Teams
- Multiple Independent Source Libraries (Clone-and-Own)
- Variation of Architecture / Functional Allocation
- Variation of Processes and Tools
- Multiple Program Plans
- Variations of Specifications/Documentation

Transformation

Consolidated Product Line Development

Single Set of Common FSW Assets to Support Multiple Configurations



- Common Oversight/Governance
- Unified Review Teams (Internal/External)
- Unified Development (Product-Oriented) Teams
- Common Core Asset Library
- Common Architecture / Functional Allocation
- Common Processes and Tools
- Common Management Plans
- Common Specification/Document Base (Tailored by Configuration)

Software Product Line Terminology

A ***product line*** is a family of products built in a way that takes advantage of the commonality shared across the family while systematically managing the variation.

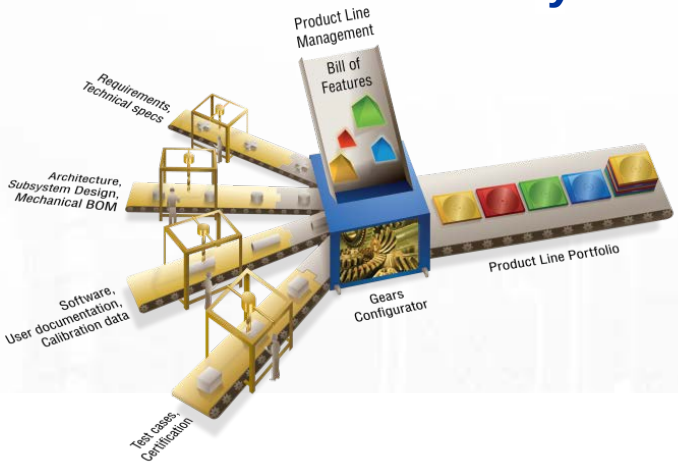
Product Line Engineering is defined as the engineering of a product line using a shared set of engineering assets, a managed set of features, and an efficient means of production that takes advantage of the commonality shared across the family while efficiently and systematically limiting and managing the variation among the products.

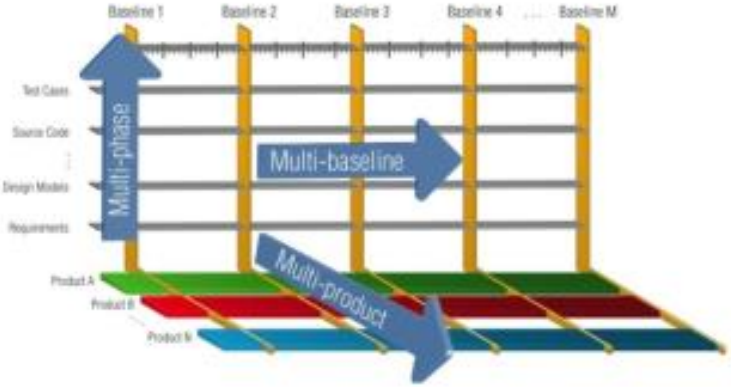
A ***feature*** is a distinguishing characteristic of a product, usually visible to the customer or user of that product.

Shared Asset: The "soft" artifacts associated with engineering life cycle of the products, the building blocks of the products in the product line. Shared assets can be whatever artifacts are representable with software and either compose a product or support the engineering process to create a product.

Shared assets are designed to be shared across the product line via built-in ***variation points***.

Treat SW Production like an automated factory





Support delivery of multiple products to multiple programs across all phases

Product Line Engineering (PLE) Methodology

1. Feature-Based
2. Automation-Centered
3. Shared Asset Focused (superset)
4. Centralized Factory Team

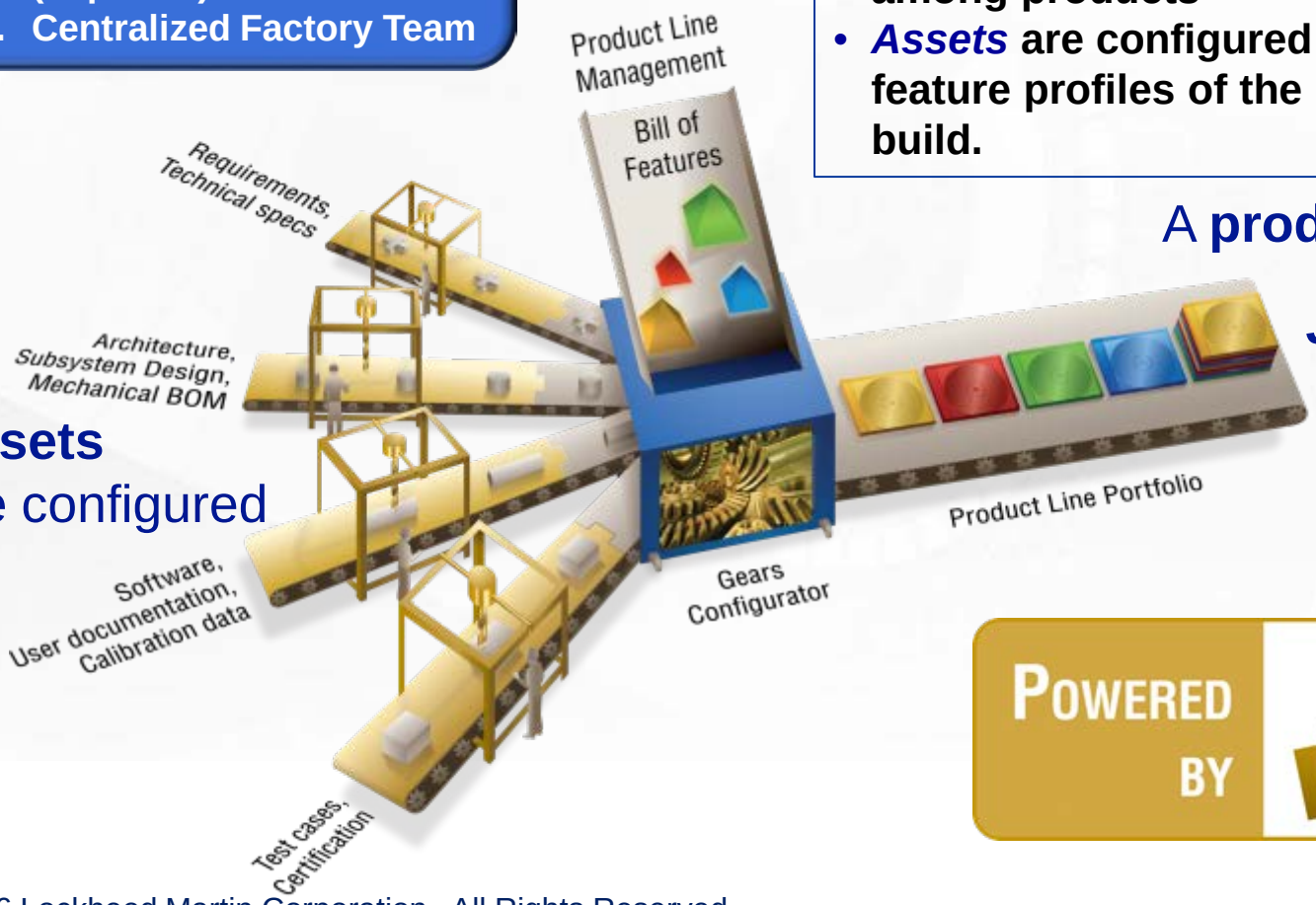
Features come in

- ***Shared assets*** **are like the factory's supply chain**
- ***Features*** **describe capabilities that vary among products**
- ***Assets*** **are configured according to the feature profiles of the products you want build.**

A **product** comes out

Just like a factory

Assets are configured



POWERED BY BigLever SOFTWARE, INC.

Software Architectures for Software Product Lines

- **The Product Line software architecture is central to success**
  - **Hardest aspect to change and the most critical to get right**
  - **1st artifact addressing Quality Attributes critical to Product Line engineering**
    - **Composability, Extensibility, Scalability, Modularity, Performance**
  - **Key to systematic reuse**

- **Mine heritage software artifacts for use in the Product Line**
  - **Requirements, designs, and implementations**
  - **Differences in program implementations help identify necessary variations as well as different instantiations of similar functionality**
  - **Adapt heritage artifacts into Product Line architecture**

- **Leverage Industry-Standard Architecture Patterns**
  - **Future Airborne Capability Environment (FACE) – DoD Aviation Platforms**
  - **Space Avionics Open Interface Architecture (SAVIOR)**
  - **CCSDS - Spacecraft On Board Interfaces Services (SOIS)**

Summary

- **Software is a key enabler of future mission needs for our customers**
  - Advanced capabilities on our platforms, payloads and instruments require new algorithms, designs, and software implementations
  - Workforce talent and software products must focus on next generation of mission challenges
- **Software affordability through commonality is critical**
  - Migration toward Software Product Line Engineering drives commonality and demonstrates affordable execution
- **Provides framework for managing commonality as well as variation**
  - Mission specific tailoring of off the shelf components