Consolidate.

Simplify.

Leverage.

Essentials of Product Line Engineering

ILTAM User's Association
Herzliya, Israel
November 27-28, 2013

Introduction

Outline

- 27 November – Product Line Engineering Essential Concepts

- **Objective and Description:** This day will introduce the essential concepts of product line engineering (PLE). PLE is a modern, proven engineering paradigm that is bringing 3x-10x improvements in time to market, engineer productivity, product quality, and product cost for product engineering organizations of all sizes and in all sectors. We will explain why PLE is an engineering approach worth studying, and how it achieves the improvements that have been attributed to it. We will give with an in-depth introduction to the key concepts, show how PLE has evolved over time in systems and software engineering, and explain what distinguishes PLE from other approaches for managing product portfolios. We will introduce First Generation PLE and Second Generation PLE, explain the difference, and show why 2GPLE is a quantum improvement over its conceptual predecessor. The first-day session will introduce two of the cornerstone activities of PLE, feature modeling and asset engineering, and show modern approaches for each and how they contribute to PLE at large.

- 28 November – Putting Product Line Engineering to Work in an Organization

- **Objective and Description:** The second day will take PLE from concepts and overviews into real-world organizational practice. We will explain how PLE for small projects (often the purview of academic papers) differs from PLE for large projects and large organizations. We will introduce the three dimensions of PLE, showing how the multi-phase dimension and the temporal dimension are needed to augment the oft-described multi-product dimension, for a true holistic solution in real-world settings. We will explore a number of "advanced topics," including product auditing and production line architecture. Finally, we will discuss what it takes to adopt PLE in an organization, and how to construct a comprehensive business for doing so.

"Notional" Schedule: Wednesday

08:45-09:00	Welcome
09:00-11:00	• Student and instructor introductions • Introduction an overview of product line engineering (PLE)
11:00-11:15	Coffee Break
11:15-12:45	First Generation vs. Second Generation PLE
12:45-13:45	Lunch Break
13:45-15:15	Introduction to system feature modeling for PLE
15:15-15:30	Refreshments
15:30-17:00	Introduction to asset engineering for PLE
17:00-17:15	Q&A

"Notional" Schedule: Thursday

08:45-09:00	Welcome
09:00-11:00	PLE in the large: Large organizations and/or large product lines
11:00-11:15	Coffee Break
11:15-12:45	PLE Advanced Topics: • Three dimensions of PLE • Production line architecture • Temporal management for PLE • Product auditing • PLE and Model Driven Development
12:45-13:45	Lunch Break
13:45-15:15	Building a business case for PLE
15:15-15:30	Refreshments
15:30-17:00	Organizational adoption of PLE
17:00-17:15	Q&A

Audience

- This seminar is intended for
  - Members of product engineering organizations who exercise a decision-making role concerning the approaches used to engineer the organization's products.
  - Executive and technical leadership in product engineering organizations
- Participants should have familiarity with
  - System engineering lifecycle concepts (such as requirements, design, testing, and the like)
  - Your organization's product portfolios and portfolio management approaches

Introductions

Introduction and Overview of Product Line Engineering

But First, A Story:
How I Became Interested in PLE

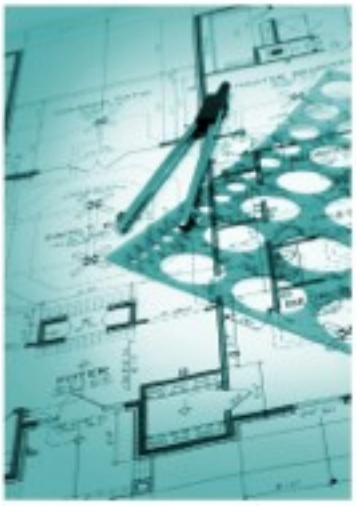
I used to work here.

I was interested in this.

(Except for systems, not buildings)

Question

- What is the relationship between a system's architecture and its ability to achieve a required set of quality attributes?
  - Performance
  - Reliability
  - Availability
  - Modifiability
  - Testability
  - ...and so forth



To try to answer this question, I was doing this.

A Revelation

- A colleague told me "I know a system designed to support a quality attribute you haven't thought of."
- Paul: "Really? What?"
- Colleague: "The ability to support an entire family of systems."
- Paul: "Where is this system?"
- Colleague: "Stockholm."
- Paul (grabbing his net): "When do we leave?"

CelsiusTech

- Swedish defense contractor with
  - approximately 2,000 employees
  - about $300 million in annual sales
- Longtime supplier of naval shipboard command and control systems to navies around the world

Reference: Bass, Len; Clements, Paul; & Kazman, Rick. *Software Architecture in Practice, 2nd Edition*, Addison-Wesley, 2003.

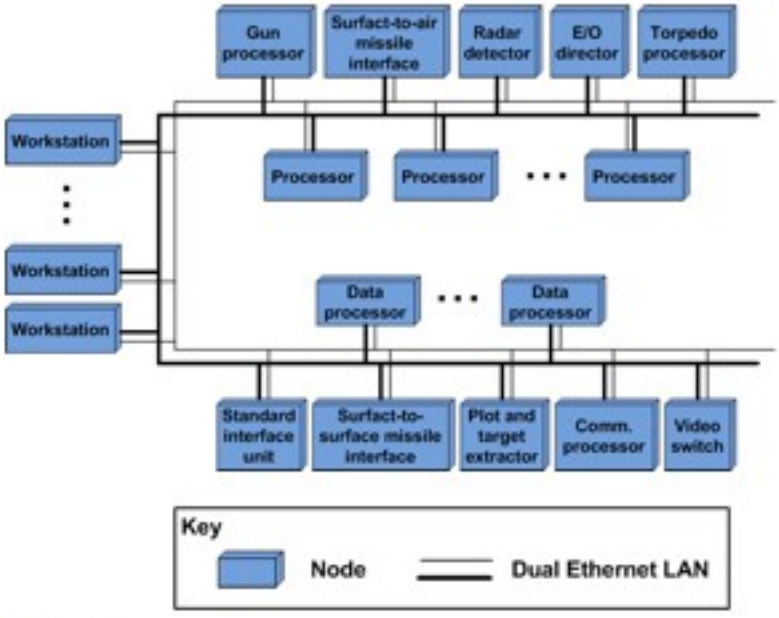
1985: Disaster Struck!

- CelsiusTech marketers landed two large contracts simultaneously.
  - 1,000,000 source lines of code (SLOC) each (estimated)
  - greater complexity of requirements than prior systems the company had developed
- CelsiusTech realized it could not fulfill both contracts unless it started doing business in a totally new way.
  - Earlier systems were troublesome to integrate and had cost schedule overruns.
  - Hiring was not an option: there was a shortage of engineers during this time.

CelsiusTech's Response

- CelsiusTech adopted what we would now call Product Line Engineering (PLE)
- Business strategy
  - Create a product family: ShipSystem 2000
  - New systems would be added to the family as business opportunities emerged
- Technical strategy
  - Create a new generation of systems:
    - hardware and software
    - supporting a PLE approach
  - Configure systems from their product family's shared assets
  - New projects would be added to the family.

SS2000 System Architecture



Typical System Configuration

- 15-30 nodes on a local area network (LAN)
- 30-70 CPUs
- 100-300 Ada programs
- 1-1.5 million Ada SLOC

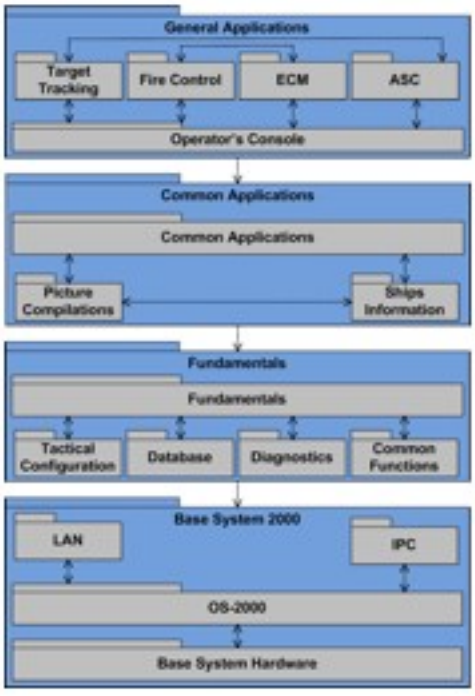
A Typical Layered Architecture

100% Ada

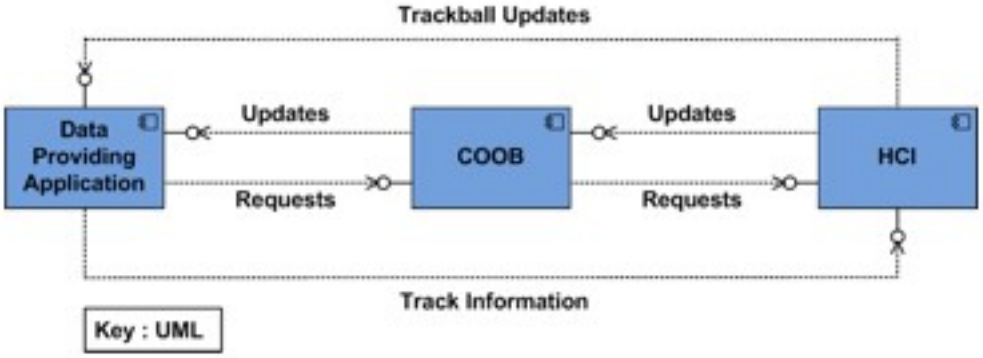
100% Ada

95% Ada,
5% assembly language

10% Ada, 90% C



Publish-Subscribe Style



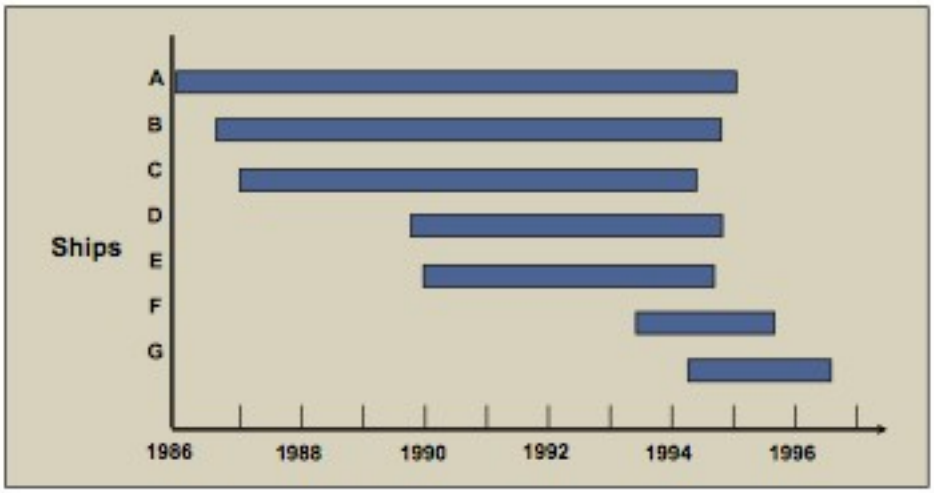
The "COOB" is a blackboard (a data repository that lets its clients subscribe to events of interest, such as an updated value).

- Data-producing elements send their updates to the COOB.
- Data-consuming elements retrieve data from the COOB.
- Trackball updates and track information bypassed the COOB for performance reasons.

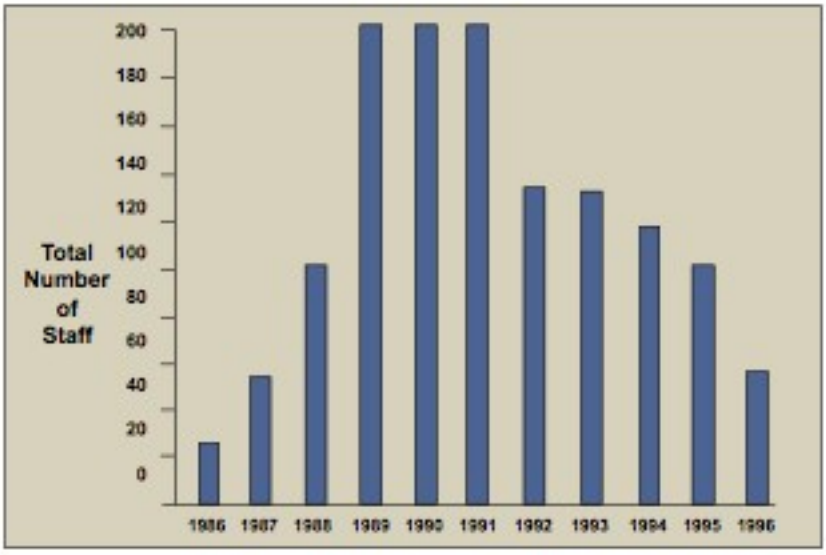
Members of Product Family

- Sold 55 variants in SS2000 product family
  - Swedish Goteborg class coastal corvettes (380 tons)
  - Danish SF300 class multi-role patrol vessels (300 tons)
  - Finnish Rauma class fast-attack craft (200 tons)
  - Australian/New Zealand ANZAC frigates (3,225 tons)
  - Danish Thetis class ocean patrol vessels (2,700 tons)
  - Swedish Gotland class A19 submarines (1,330 tons)
  - Pakistani Type 21 class frigates
  - Republic of Oman patrol vessels
  - Danish Niels Juel class corvettes

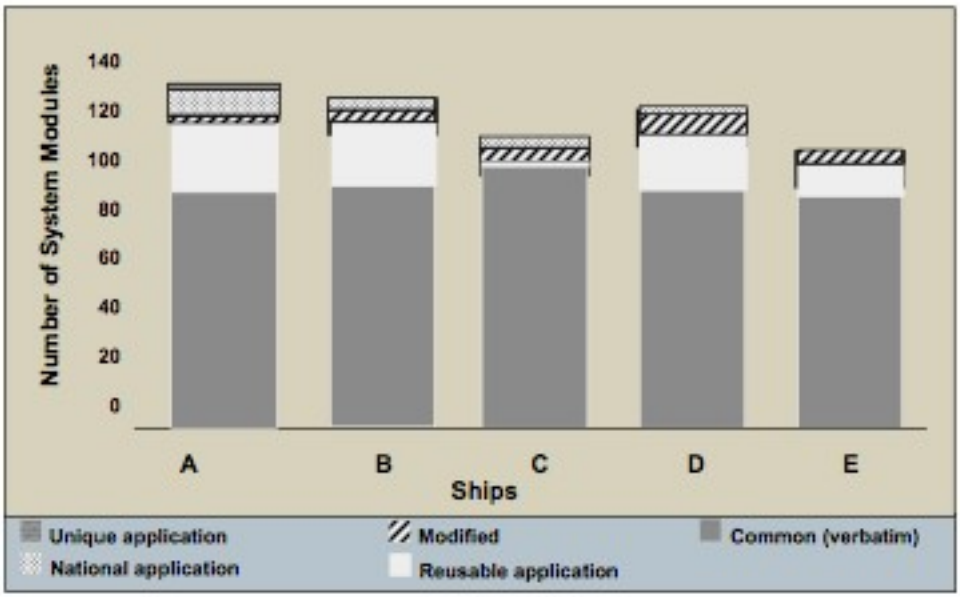
Result of Changes: Shrinking, Predictable Schedules



Result of Changes: Less Staffing



Result of Changes: Reuse



Organizational Lessons Learned

- A different organizational structure was required, so the organization was divided into two groups:
  - 1. "Customer" projects
  - 2. Family-wide shared assets
- A technical steering group was needed to
  - keep the product line current in the domain
  - be a catalyst for cross-product-line sharing
  - watch for promising technologies
- Focus on skill development was critical.
  - Expertise was required in the domain, not in coding.
  - The training provided was extensive compared to industry norms.

Business Lessons Learned

- Economic
  - The up-front investment was reduced by using off-the-shelf platforms, components, and graphical user interface (GUI) builders.
  - The software-to-hardware cost ratio was reversed.
- Customer related
  - Customers formed user groups to guide the evolution of the product line.
  - The negotiation of customer requirements became based on sharing over the entire product line.
  - A customer perception of lower risk/cost and high confidence existed.

More Results

- CelsiusTech was able to explore other domains in which its architecture and product line approach offered promise.
- Air defense sector
  - ground-based radar-driven anti-aircraft guns
  - "An air defense system is just like a ship on land that doesn't move very much."
  - 40% of the new system could be created immediately from SS2000.

What did I learn?

- Architecture (what I went to study) was necessary for success, but the actual architecture was pretty boring!
  - "You went all the way to Stockholm for *that*?!?"
- *Business* and *organizational* lessons learned were at least as important as technical lessons.
- We have not seen improvement results like this since the dawn of the computer age.

- I went to Stockholm to study architecture.
- I came home determined to study product line engineering.

...of my story

But...

...of my interest in product line engineering

Now I work here.

BigLever Software, Inc.

BigLever at a Glance

- Industry leader in Systems and Software Product Line Engineering (PLE) tools and services
  - 11 years of commercial practice with *Gears™* technology and methods
  - Strategic partner of IBM Rational
- Industry standard PLE framework, tools and methodology
  - *BigLever Gears PLE Tool and Lifecycle Framework™* and industry-wide integrations
    - IBM Rational, Serena, Microsoft, Open Source, Perforce, ...
  - *BigLever 3-Tiered PLE Methodology™*
- Proven success in large-scale PLE deployments

What is Product Line Engineering?

Product Line Engineering

- Full name: "Systems and software product line engineering"
- Definition: *The disciplined engineering of a **portfolio of related products** using a common set of **shared assets** and a common **means of production**.*

To be competitive, most product development organizations deliver a product line –
a portfolio of similar products or systems with variations in features and functions

Products

- Products can include any combination of
  - software

  - systems in which software runs

  - non-software systems that have software-representable artifacts (such as engineering models or development plans) associated with them.

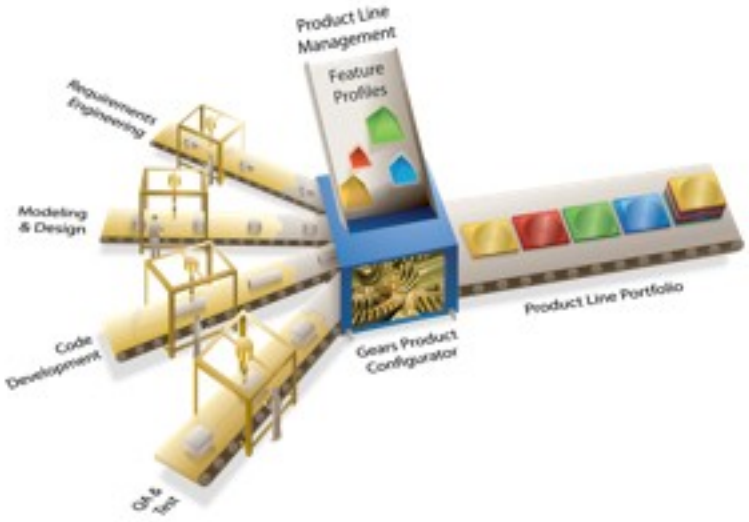
Why is PLE Important?

- PLE is of interest because of remarkable efficiencies it has shown in the development process:
  - Large-scale productivity gains
  - Decreased time to market
  - Increased product quality
  - Decreased product risk
  - Increased market agility and presence
  - Increased customer satisfaction
  - More efficient use of human resources
  - Ability to sustain unprecedented growth

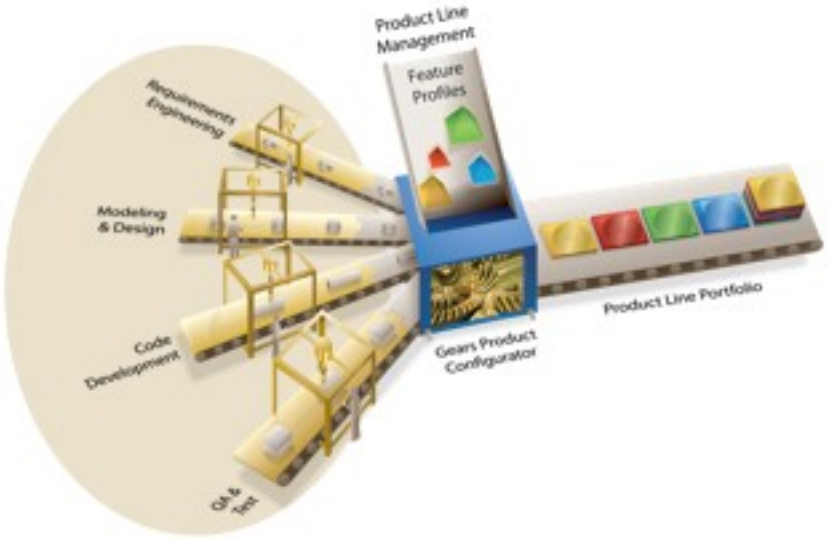
Look Around

- What product lines do you see in this room?
- What product lines did you encounter on your trip here this morning?

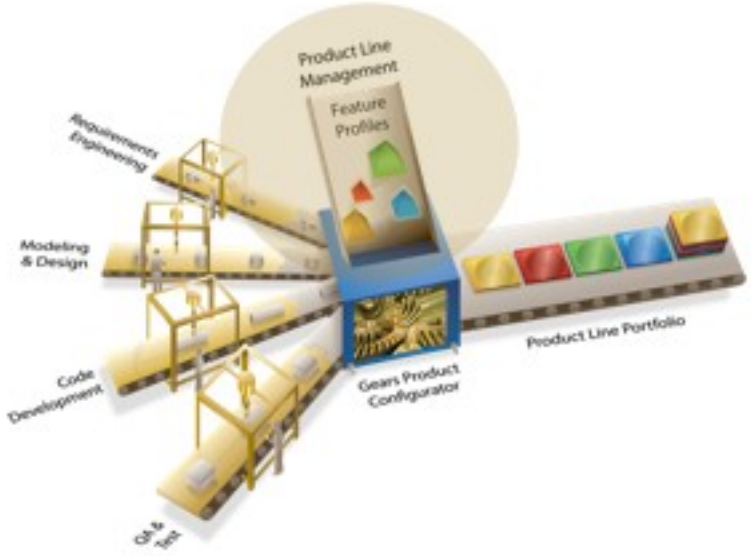
An Efficient Means of Production for Systems and Software Product Lines



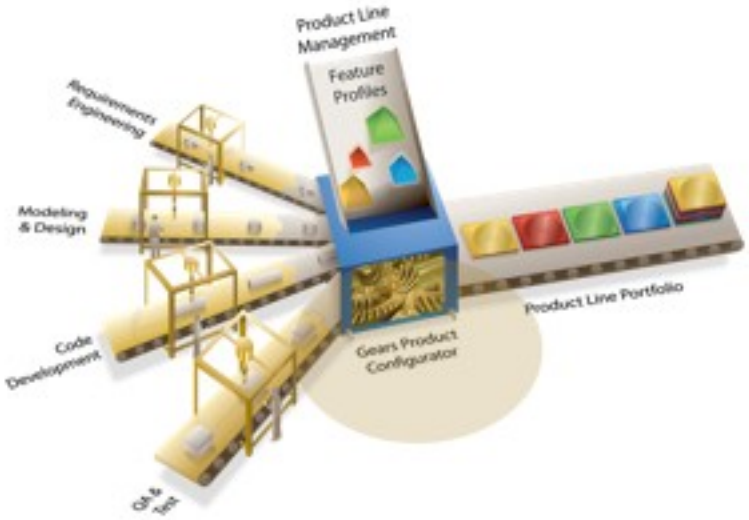
Shared assets are like the factory's supply chain.



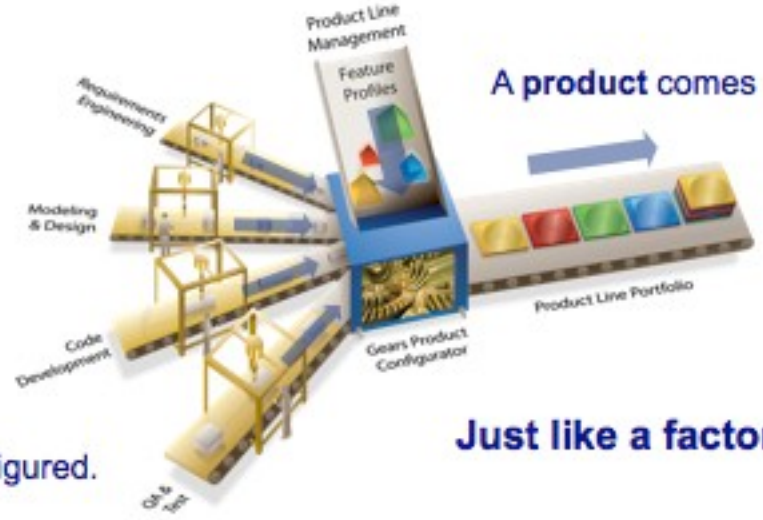
Features describe capabilities that vary among products.



Assets are configured according to the *feature profiles* of the products you want build.



Features come in.

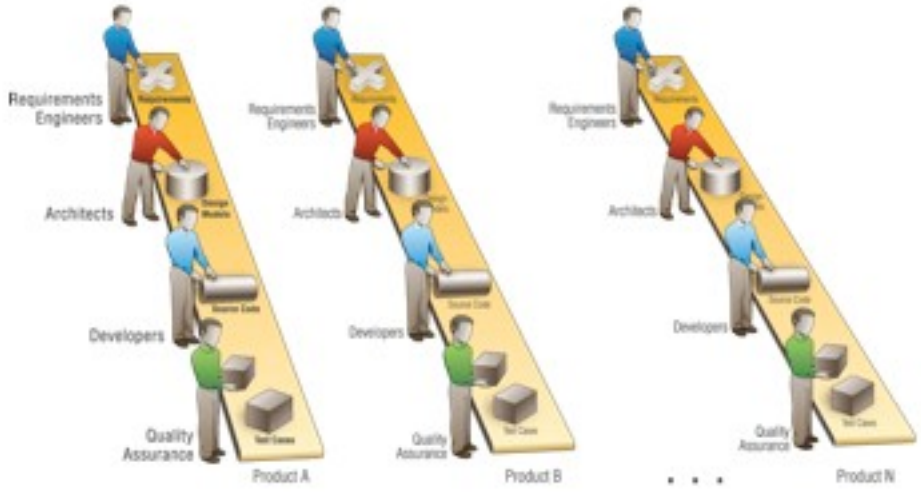


A **product** comes out.

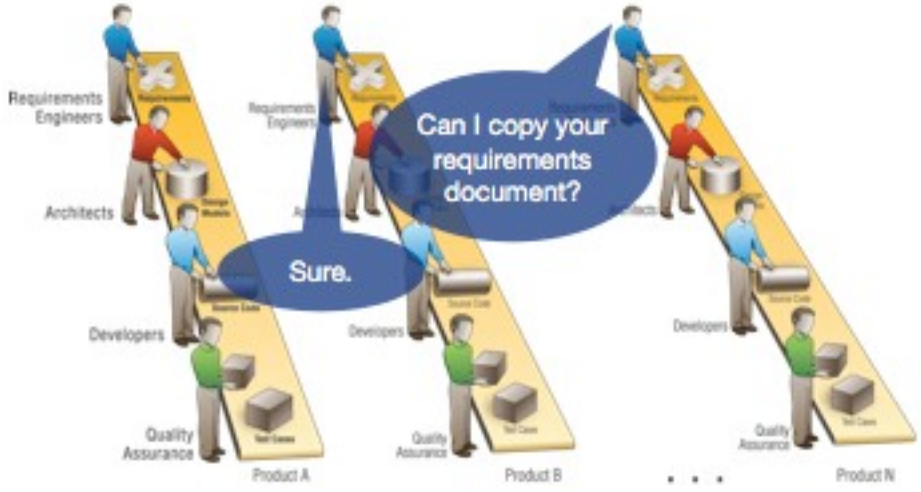
Assets are configured.

Just like a factory.

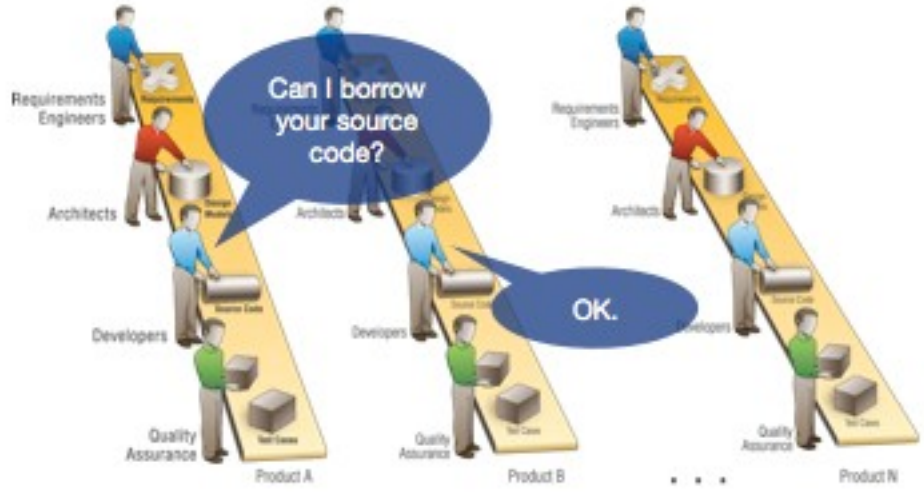
PLE is a move away from the traditional product-centric development silos.



With traditional approaches there may be some reuse among products, but it's usually not systematic.

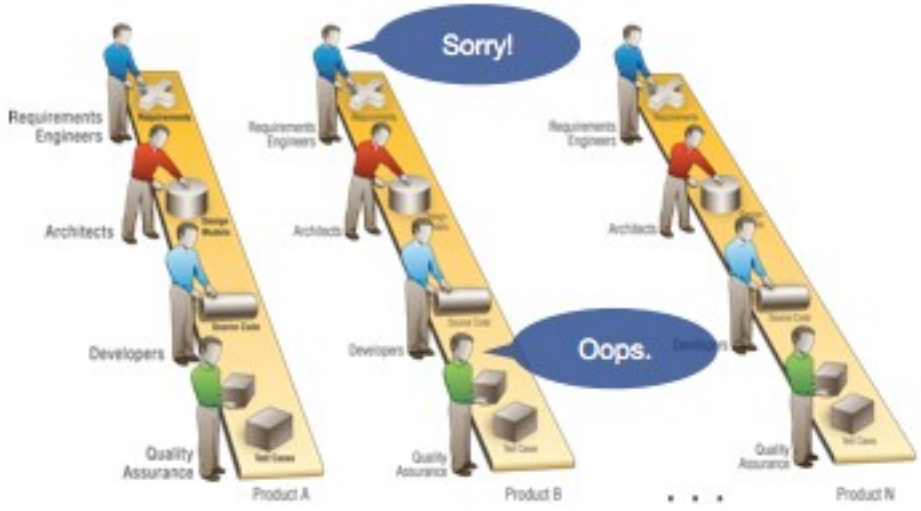


With traditional approaches there may be some reuse among products, but it's usually not systematic.



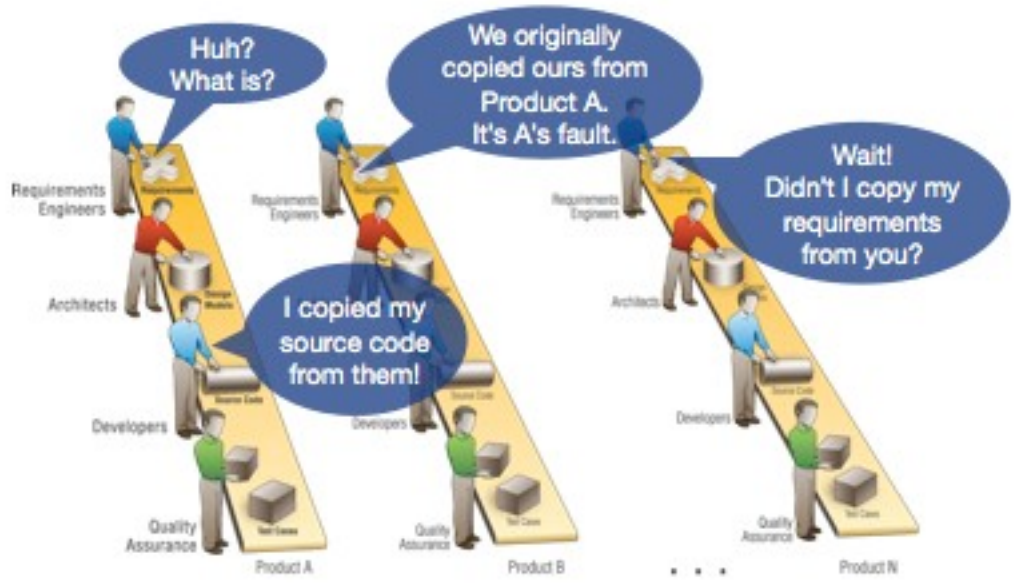
This doesn't scale very well.

Suppose Product B has a defect.
Suppose it came from Product B's requirements.

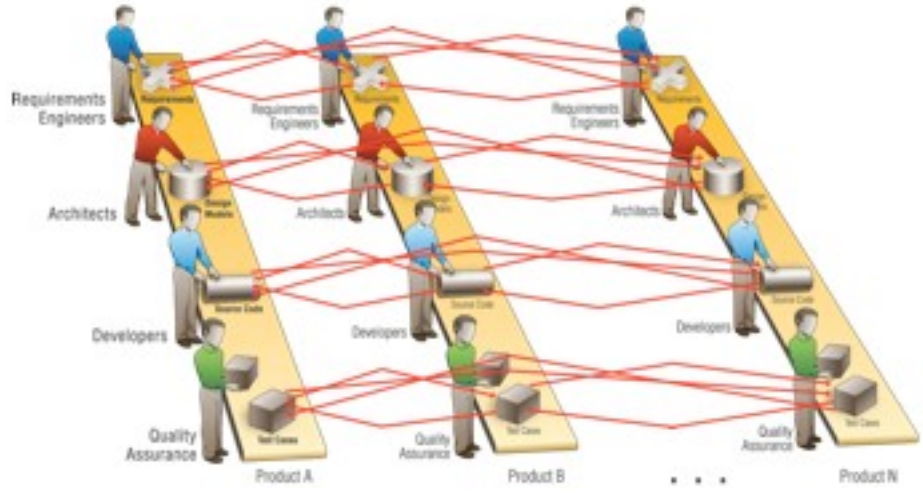


Team B can fix Product B.

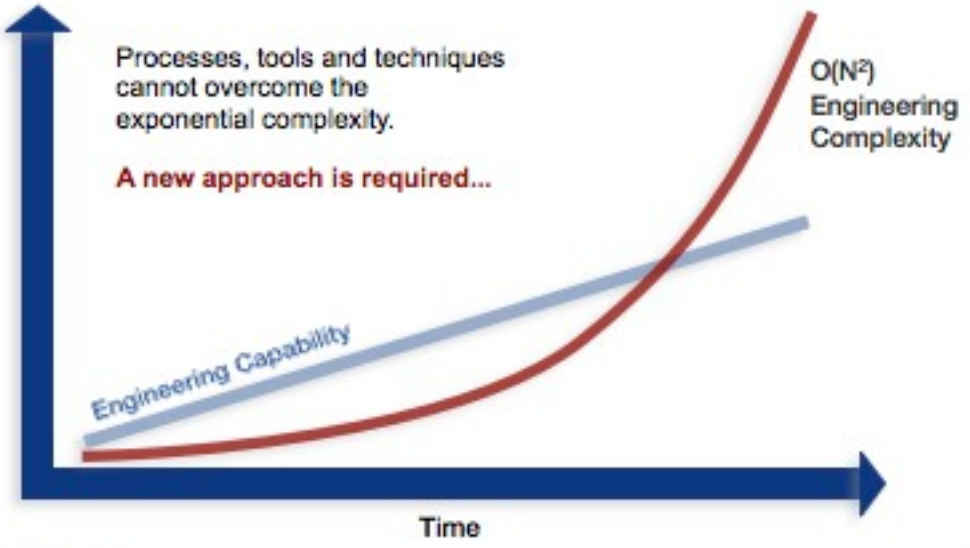
However...



The communication and coordination that has to occur for a portfolio of **N products** is proportional to N^2



The Challenge of Product Line Engineering: Harnessing Complexity



An Incredibly Brief History of Product Line Engineering

1103 AD: Ying tsao fa shih (營造法式)

- Written by Li Chieh, state architect of emperor Hui-tsung, published in 1103 AD
- Set of building codes for official buildings
  - Described layout, materials, and practices for designing and building
  - Listed standard parts and standard ways of connecting the parts
  - Parameterized variations of the parts
  - Allowed components based on the building's purpose
  - Gave options for various component choices
- Defined a "product line" of buildings

1960s: IBM 360 and OS/360

- From the *Principles of Operation*:
  - *Models of System/360 differ in storage speed, storage width (the amount of data obtained in each storage access), register width, and capabilities for processing data concurrently with the operation of multiple input/output devices.*
  - *Several CPU's permit a wide choice in internal performance. Yet none of these differences affect the logical appearance of these models to the programmer.*
  - *An individual System/360 is obtained by selecting the system components most suited to the applications from a wide variety of alternatives in internal performance, functional ability, and input/output (I/O).*

1970s: Paper by David Parnas

- "On the Design and Development of Program Families" by David L. Parnas, 1976
  - Says that it pays to consider similar software programs as a single family

On the Design and Development of Program Families

DAVID L. PARNAS

1970s-1990s

- Software reuse movement
  - Emphasized code repositories.
  - Emphasized opportunistic reuse, as opposed to planned reuse.
  - Primary contribution to PLE was to instill the notion that software systems might not (or should not) be built from scratch.
- Generative programming
  - Uses domain-specific languages to specify a product
  - Engineers work on shared assets (requirements, design, and so forth) that apply across the entire portfolio.

1980s: Boeing 757 and 767

These two very different aircraft were designed together and have about 60% of their parts in common. Parts were designed to work on *both* aircraft.

1990s-2000s:
Software Product Line Engineering

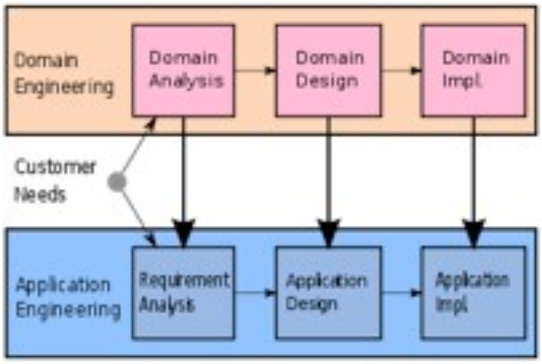
2000s - 2010s

- First generation approaches recognized, distilled
- Dawn of Second Generation Product Line Engineering

First Generation vs. Second Generation Product Line Engineering

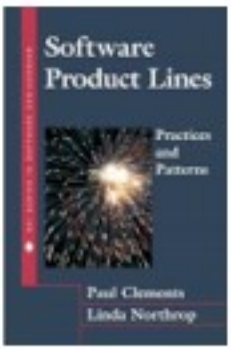
First Generation Approaches #1

- A strong dichotomy between *domain engineering* and *application engineering*
  - Or *core asset development* and *product development*.



First Generation Approaches #2

- Explicit allowance of non-software artifacts in the collection of "core assets"…
  - …but an unmistakable emphasis on *software*.



First Generation Approaches #3

- Focus on *features* as the language to describe a product line's domain and a way to discriminate products from each other.
- Feature: "A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems."

First Generation Approaches #4

- Acknowledgment of configuration management as an essential practice under PLE, but without a strong distinction between core asset CM and product CM.

Aspects Peculiar to Product Lines

CM is, of course, an integral part of any software development activity, but it takes on a special significance in the product line context. As illustrated in the following figure, CM for product lines is generally viewed as a multidimensional version of the CM problem for one-of-a-kind systems.

Configuration Management and Software Product Lines

The core assets constitute a configuration that needs to be managed, each product in the product line constitutes a configuration that must be managed, and the management of all these configurations must be coordinated under a single process.

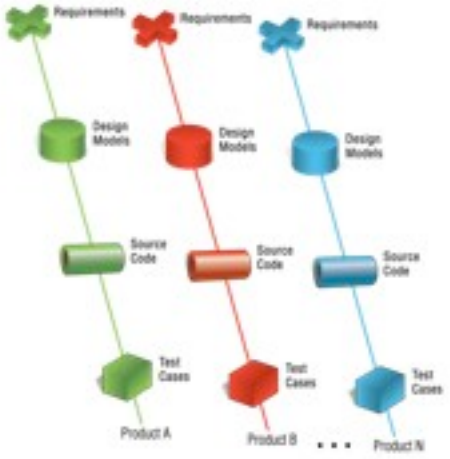
Problems with early approaches

- No clear, prescriptive, repeatable methodology
- Case studies tend to show ad hoc approaches
  - Every one is different
- Emphasis on software is a serious limitation
- Automation is welcomed, but never embraced

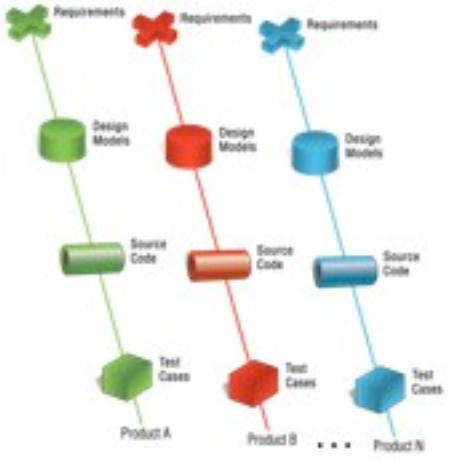
Second Generation PLE

Second generation PLE strongly embraces the factory paradigm.

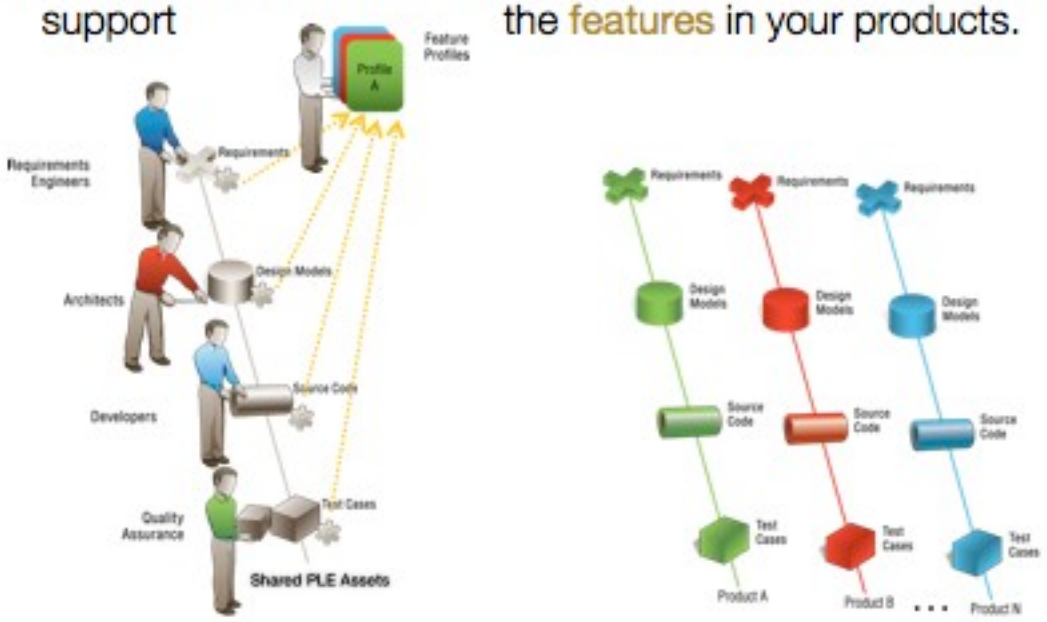
Product line engineers work on assets shared by the whole product family.



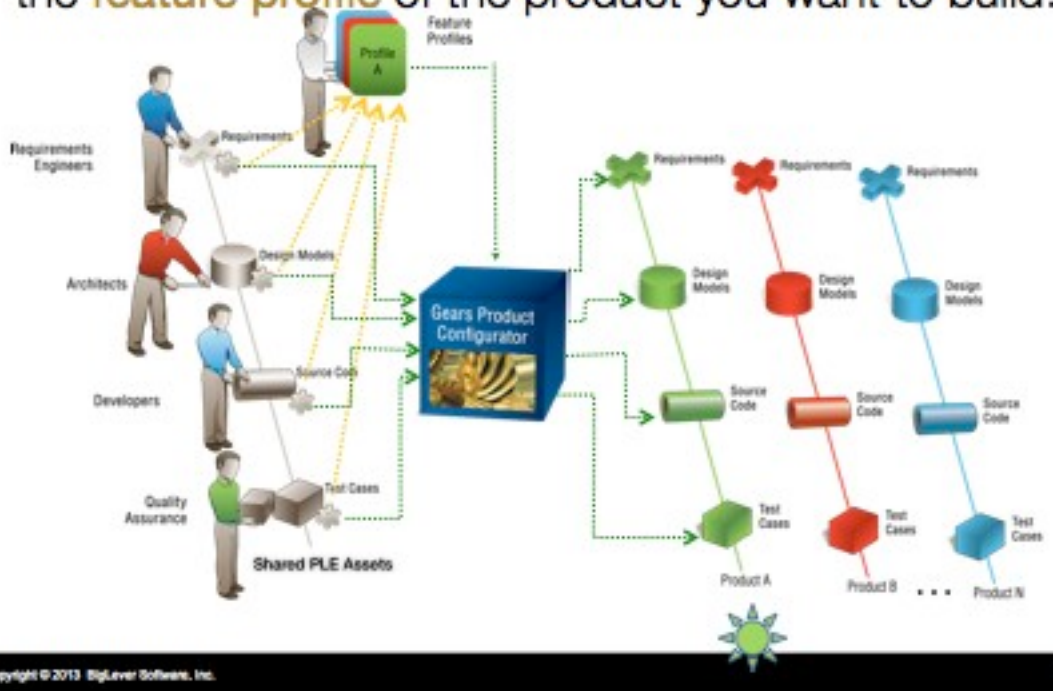
Shared assets are built with variation points.



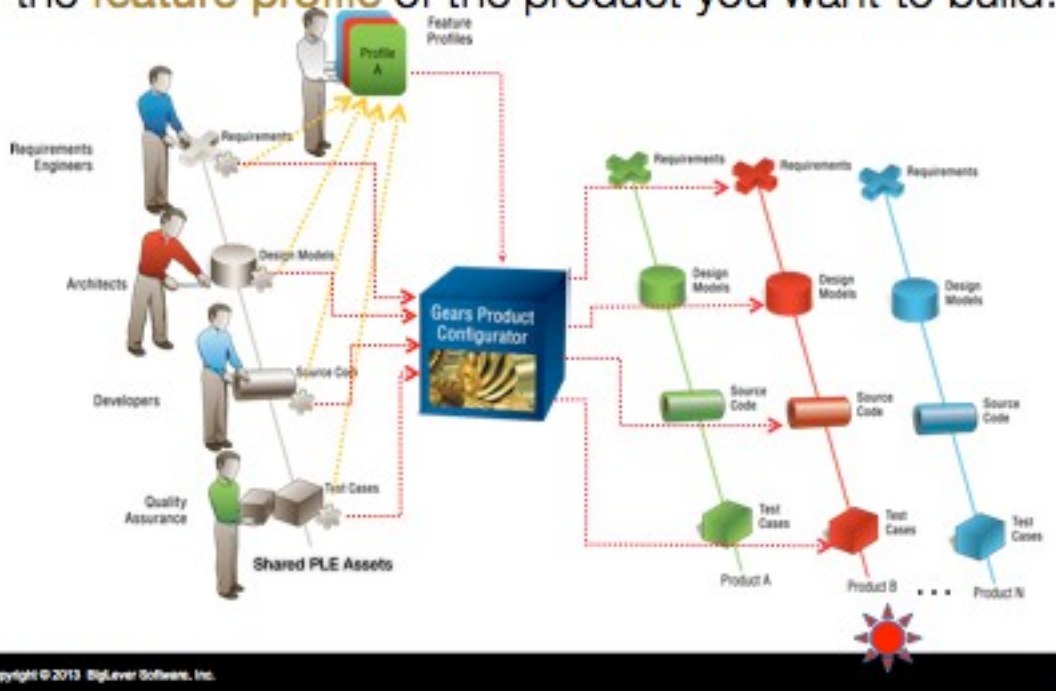
Variation points determine how the shared assets change to support the features in your products.



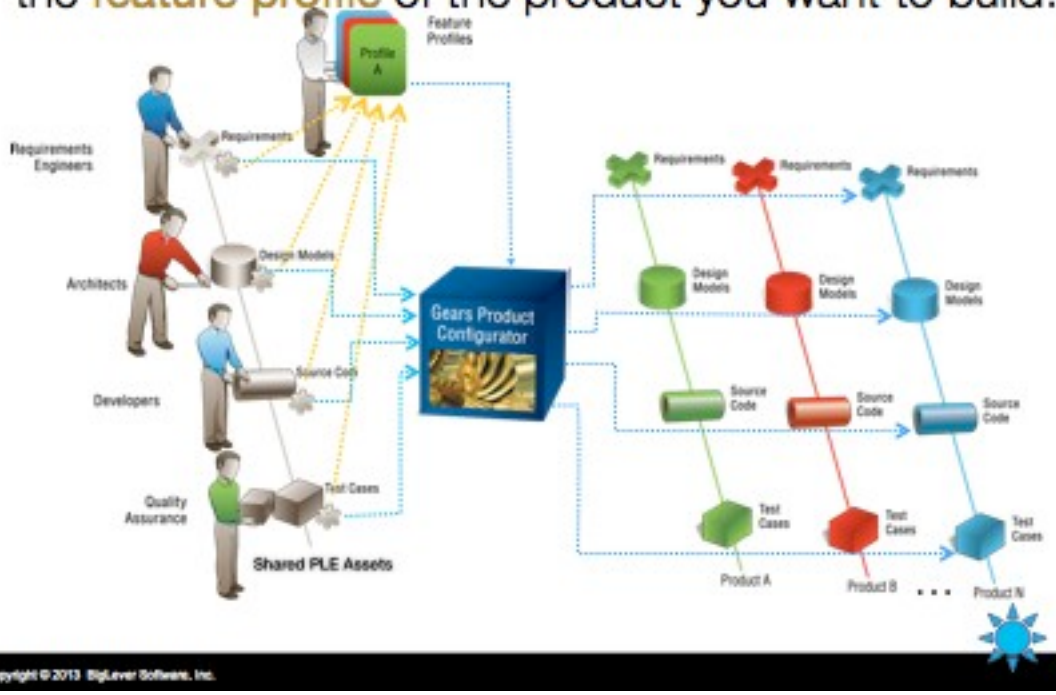
Gears exercises the variation points according to the feature profile of the product you want to build.



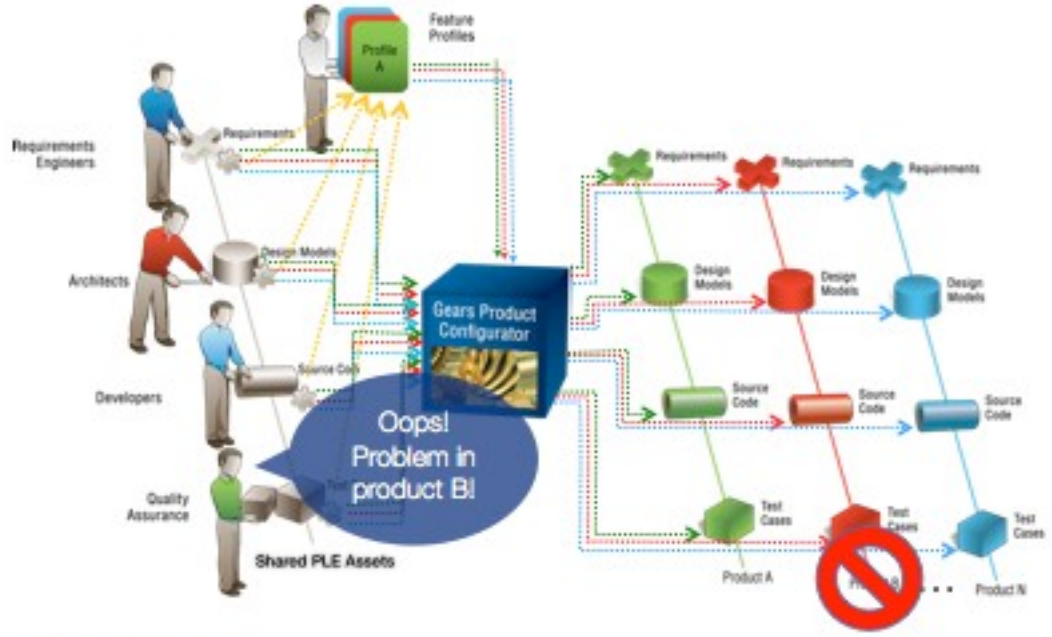
Gears exercises the variation points according to the feature profile of the product you want to build.



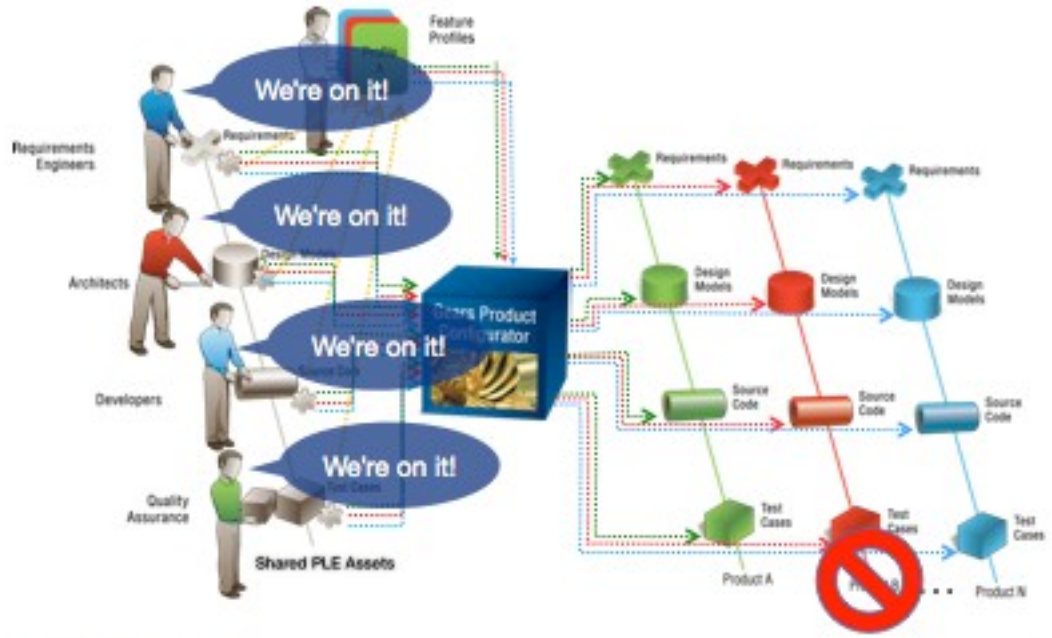
Gears exercises the variation points according to the feature profile of the product you want to build.



Now, to fix an error…

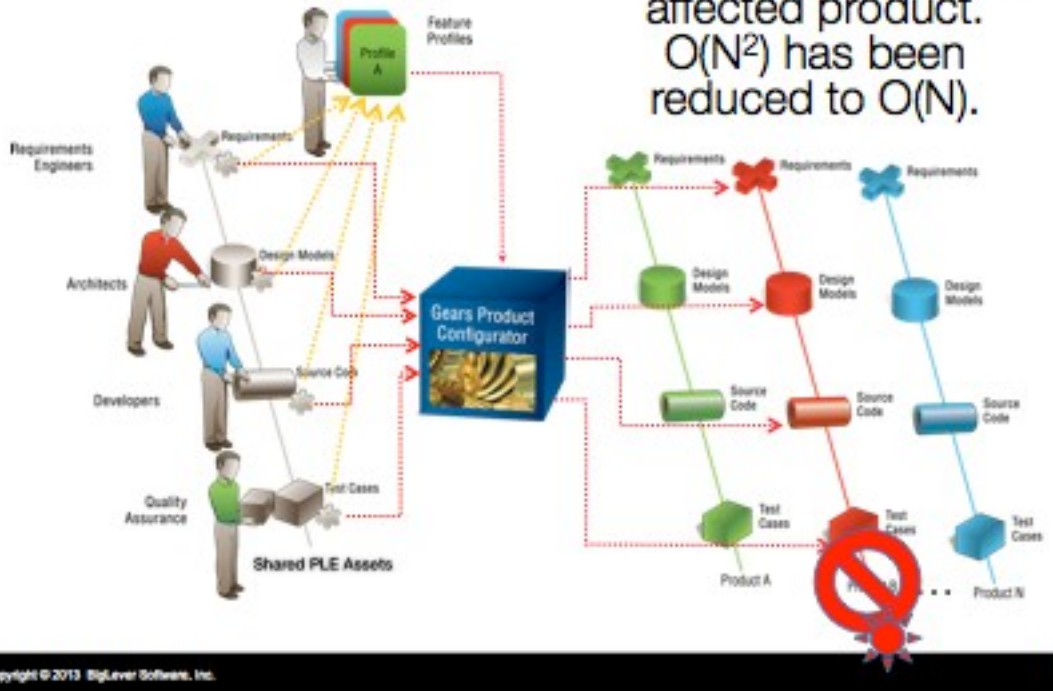


…we fix it once on the factory side…



BigLever Software, Inc.

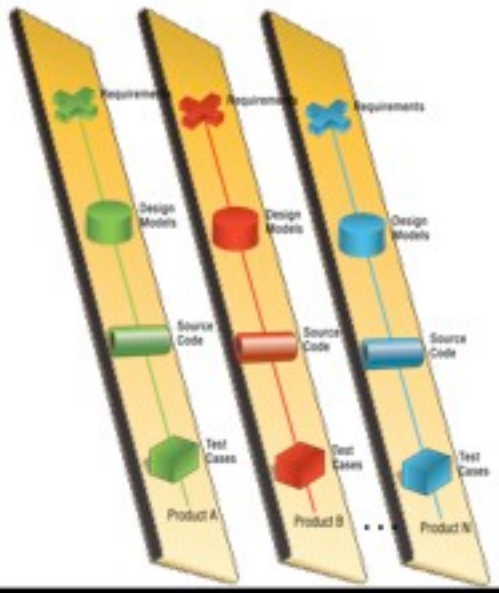
…and re-generate any affected product. $O(N^2)$ has been reduced to $O(N)$.



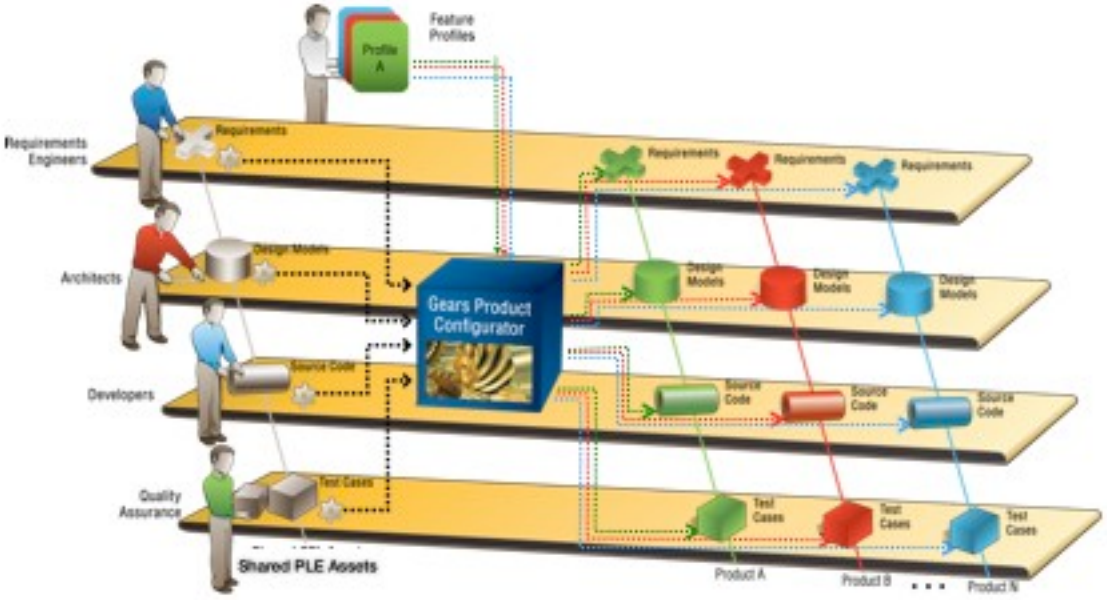
BigLever Software, Inc.

Product Line Engineering

is a shift from the vertical product perspective…



...to a horizontal shared asset perspective.



Second Generation Approaches #1 - #3

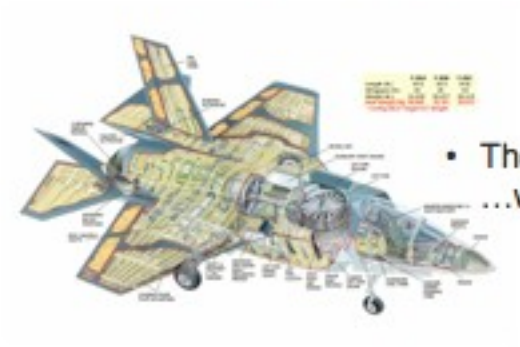
1. Features serve as the *lingua franca* to express product differences and to exercise the variation points in all assets.
   - This extends the notion of "feature" as a way to just capture domain analysis and product descriptions;
2. Industrial-strength easy-to-use automation is employed to maintain configurations and turn out products.
   - Before, automation was thought of as "nice to have" but not essential. Now it is.
   - Notice how "application engineering" has shrunk to almost nothing.
3. Artifacts from all lifecycle phases are treated as first class citizens, not just the software.

Multi-product Solution across the Lifecycle

Multi-phase Solution across the Lifecycle

Product Line Engineering (or) Systems and Software Product Line Engineering

- The products can be software.



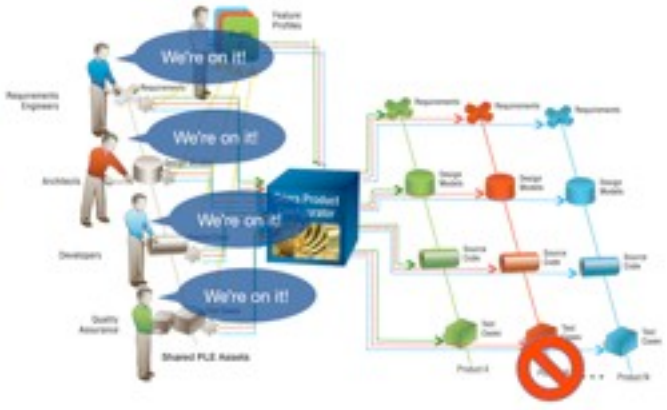
- The products can be systems …with software inside.

- The products may contain no software at all.

Second Generation Approach #4

4. A vastly simplified configuration management model

- CM applies to the shared assets, not the products
- Complexity of product line CM is reduced to that for *single products*.



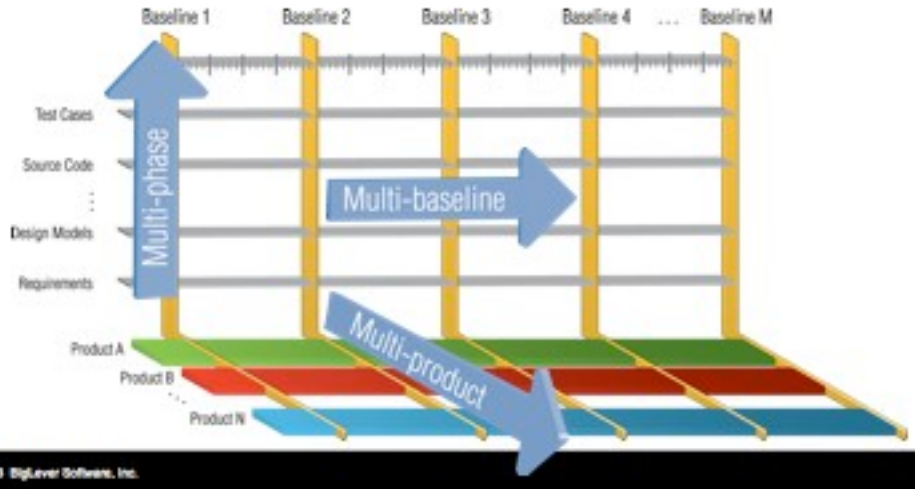
Second Generation Approach #5

5. Feature models with encapsulating constructs to facilitate modular and hierarchical product lines developed across organizational boundaries.

More on that later.

Synchronous concerns in a PLE Solution

- *Multi-product*. Feature-based variation management and automated production line
- *Multi-phase*. Product line lifecycle assets, architecture and traceability
- *Multi-baseline*. Product line change management and baseline management



Who Is Using Second Generation PLE?




General Motors: Mega-scale PLE

- GM has one of the most complex systems and software product line engineering challenges in the world
  - 3000 product line engineers
  - 300 hierarchical subsystems
  - Thousands of variant features
  - Millions of product instances
  - Tens-of-thousands of unique product variants
  - Dramatic increase in product line variation due to new propulsion systems and active safety
  - Global diversity in legislative regulations
  - Extreme economic and competitive pressures
  - Product line and feature set evolves annually
  - 15 concurrent temporal development streams




General Motors

- Largest automotive manufacturer in the world
- 9 million vehicles sold in 2011
  - More than *1,000 built per hour*
  - Built in 31 countries

Lines of code: ~6M

Lines of code: ~8M

Lines of code: ~10M

Unprecedented complexity is arising from interaction of new features

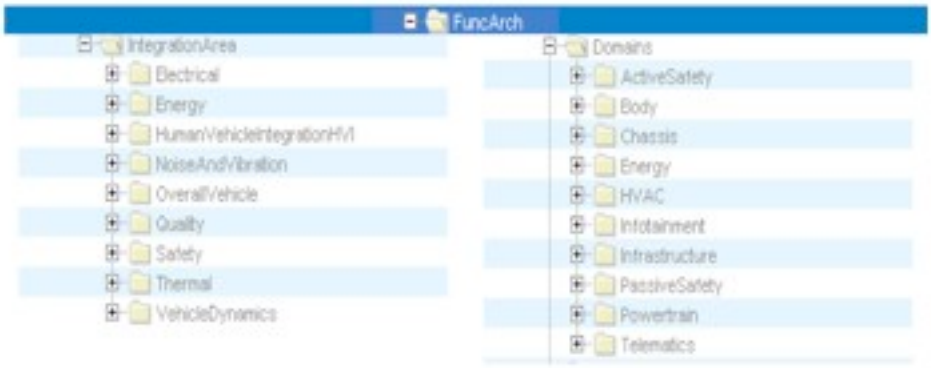
Active Safety

GM's Next Generation Tools (NGT) Initiative

- Current initiative in GM's long march towards "convergence"
  - 1991: Merging of branded divisions
  - Late 1990s: Adoption of common electrical/electronic architecture and requirements management
    - Goal: Common look and feel for features
  - Early 2000s: Commitment to adopt AUTOSAR open standards
- NGT was intended to answer the question "What common tools and processes shall we all adopt to power this convergence?"
- 2G PLE plays a large role in the answer.

Necessity of hierarchy

- GM divides their engineering into many separate (but inter-related) areas of expertise
- Powertrain, brakes, interior lighting, exterior lighting, infotainment, entry controls, …



Necessity of hierarchy

- It would be infeasible for 3,000 engineers to work on a single production line.
- Instead, they work on *production lines* for their own areas.
- Gears provides the mechanisms to have those production lines inter-operate to "describe" an entire vehicle.

LOCKHEED MARTIN

POWERED BY BigLever

Aegis Combat System

Integrated weapons system for 100 ships

US Navy Aegis Cruisers and Destroyers

95

LOCKHEED MARTIN

POWERED BY BigLever

Aegis Combat System

Integrated weapons system for 100 ships

US Navy Littoral Combat Ships

96

LOCKHEED MARTIN

POWERED BY BigLever

Aegis Combat System

Integrated weapons system for 100 ships

US Coast Guard National Security Cutter

97

LOCKHEED MARTIN

POWERED BY BigLever

Aegis Combat System

Integrated weapons system for 100 ships

Aegis ships for International Navies

98

LOCKHEED MARTIN

POWERED BY BigLever

Aegis Combat System

With Gears, Lockheed Martin manages the entire family with a single set of requirements and a single set of source code.

Requirements for a new ship take **days** to produce instead of **months**.

99

NetApp

(previously LSI Logic Engenio)

POWERED BY BigLever

300 product line engineers

Over 250K installed systems worth over $12B

Engenio Storage Division:

High-end server storage OEM for IBM, Sun, Cray, Teradata, SGI, and more.

100

NetApp

(previously LSI Logic Engenio)

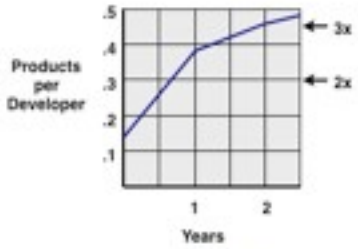
Engenio Storage Division:

High-end server storage OEM for IBM, Sun, Cray, Teradata, SGI, and more.

With Gears...

POWERED BY BigLever

product capacity quintupled



productivity tripled

ikerlan ALSTOM

With Gears...

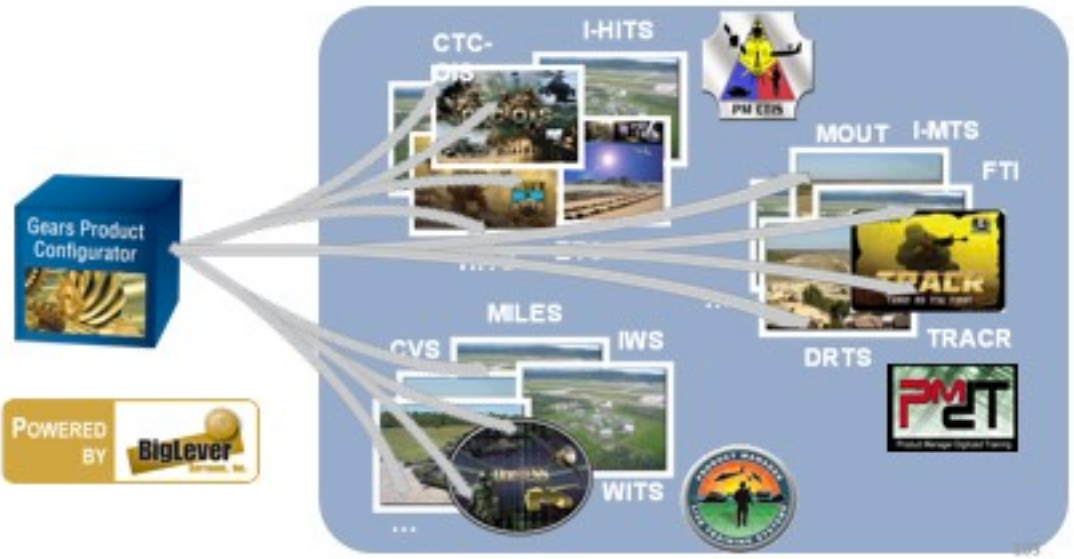
90% reduction in development time

25% reduction in development costs

Wind turbine control systems customized to optimize performance based on wind direction and speed, temperature and other factors

GENERAL DYNAMICS

Live Training Transformation (LT2) family of training systems for the U.S. Army



POWERED BY BigLever

GENERAL DYNAMICS

Live Training Transformation (LT2) family of training systems for the U.S. Army

General Dynamics teamed with BigLever to create the winning proposal for the US Army Live Training Transformation "Consolidated Product-line Management".

With Gears…

...US Army projects $580 million cost avoidance over 12 years and 150 deployments.

POWERED BY BigLever

HomeAway

Worldwide leader for online vacation home rentals

50 million users
130,000 properties
100 countries
$20B in travel bookings

50 different product instances engineered and hosted using BigLever's Gears.

HomeAway

Worldwide leader for online vacation home rentals

Their product line was up and running 60 days after they started using Gears.

Demonstration of 2GPLE using Gears

Introduction to System Feature Modeling for PLE

Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Shared assets' variation points are exercised by the configurator to produce instantiations specific to a product, which is also described by its features.

Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Shared assets' variation points are exercised by the configurator to produce instantiations specific to a product, which is also described by its features.

Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Feature: A distinguishing characteristic of a product, usually visible to the customer or user of that product.

Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Feature: A distinguishing characteristic of a product, usually visible to the customer or user of that product.

An example is a function that one product can perform that others cannot.

Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Feature: A distinguishing characteristic of a product, usually visible to the customer or user of that product.

Features express the customer-visible diversity among the products in a product line.

Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Feature: A distinguishing characteristic of a product, usually visible to the customer or user of that product.

Features are captured in feature declarations.

Second Generation Product Line Engineering (2GPLE)

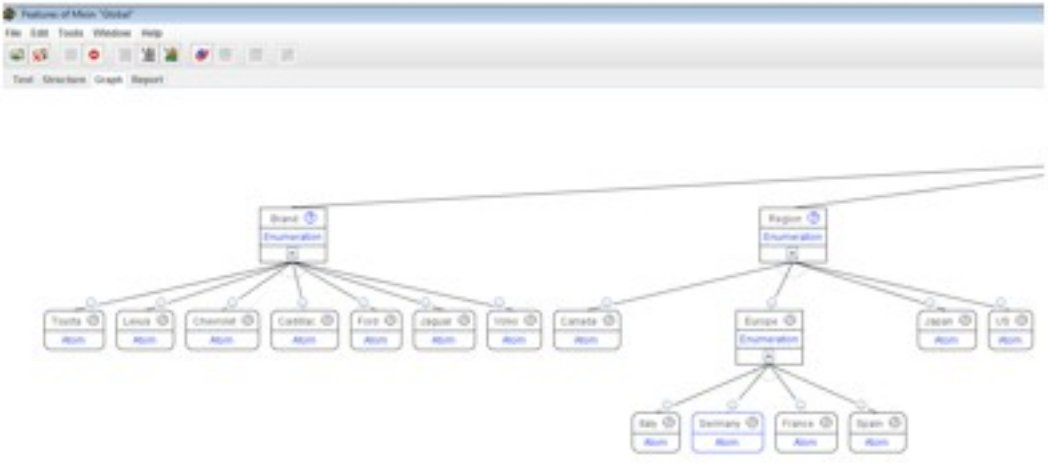
Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Features are captured in feature declarations...

...A language construct that names a feature and its possible values.

Feature declarations establish the set of *choices available*.

Feature declarations: Choices available



Features Capture What's Different

- If two products differ by a distinguishing characteristic, that's likely to be a feature.
- If every product shares a particular characteristic or capability, that's typically *not* modeled as a feature.

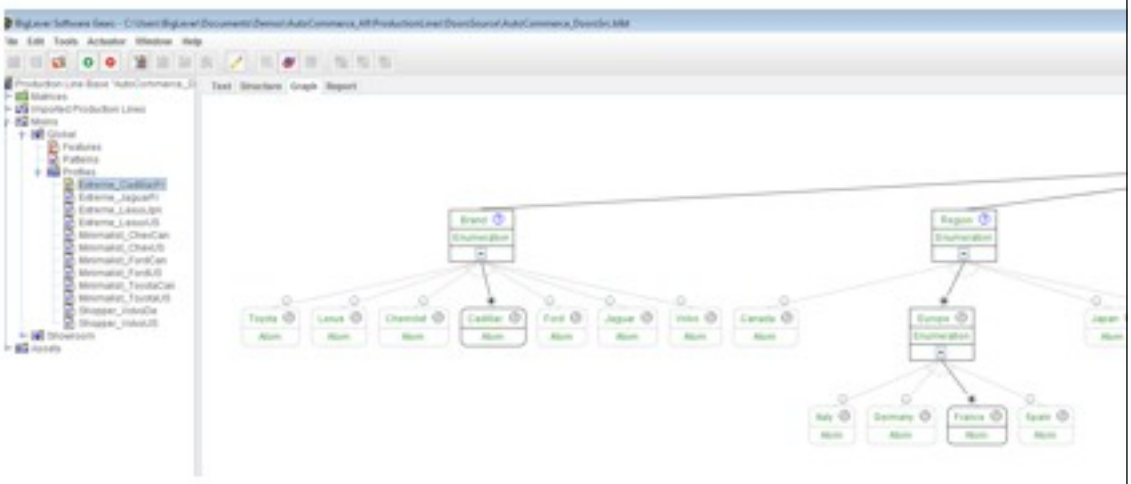
Second Generation Product Line Engineering (2GPLE)

Shared assets across the entire lifecycle are endowed with variation points expressed in terms of product features.

Feature declaration: A language construct that names a feature and its possible values. Feature declarations establish the set of *choices available.*

A feature profile is a set of *choices made* that define a product.

Feature profile: Choices made



Feature Types in Gears

Enumeration	Select exactly one value from subordinate features.
Set	Select zero or more values from subordinate features.
Record	Select all values from subordinate features.
Atom	Named member/value of a set or enumeration.
Boolean	true, false
Integer, Float	Signed or unsigned numeric value
String	Character string delimited by double quotes. Use backslash to quote special characters.
Character	Single character delimited by single quotes. Use backslash to quote special characters.

Feature Modeling Exercise

Team 1 Feature Modeling Exercise

- Take a few moments to think about kitchen ovens....




...and their control system interfaces.

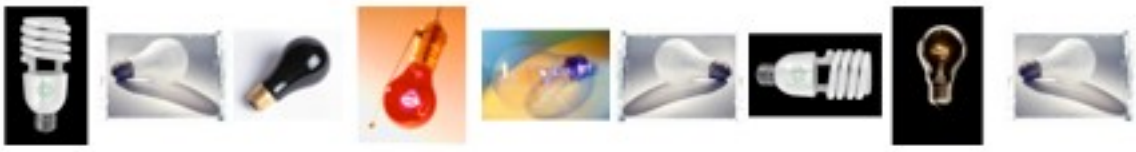


Feature modeling exercise

- Suppose we work for the engineering department for a company that designs, manufactures, and sells kitchen ovens.
- What are the salient characteristics that differentiate our ovens from each other?
  - Functions and capabilities
  - Type
  - Control interface
  - ...

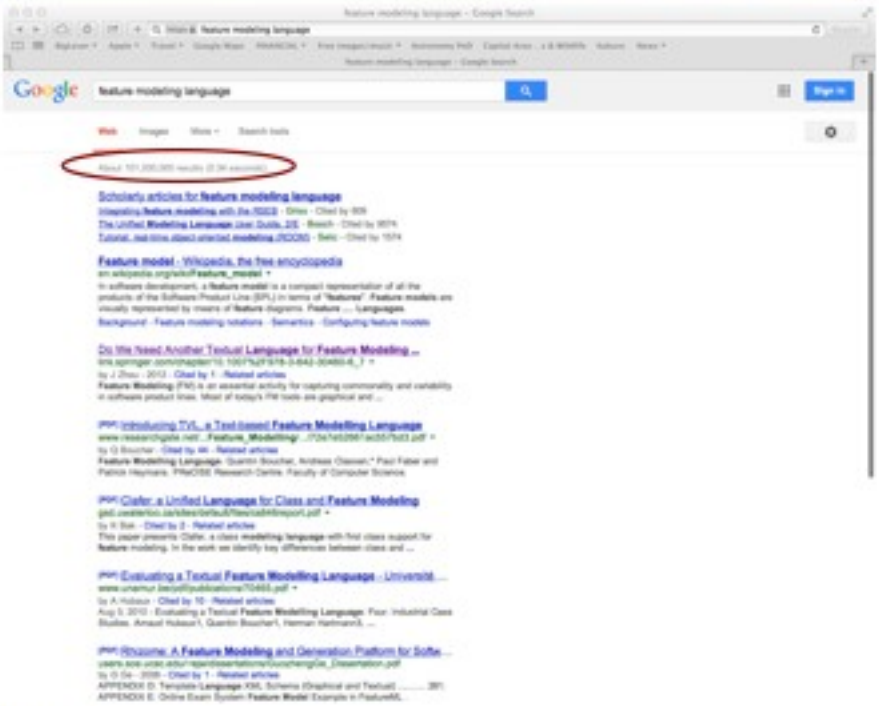
Team 2 Feature Modeling Exercise

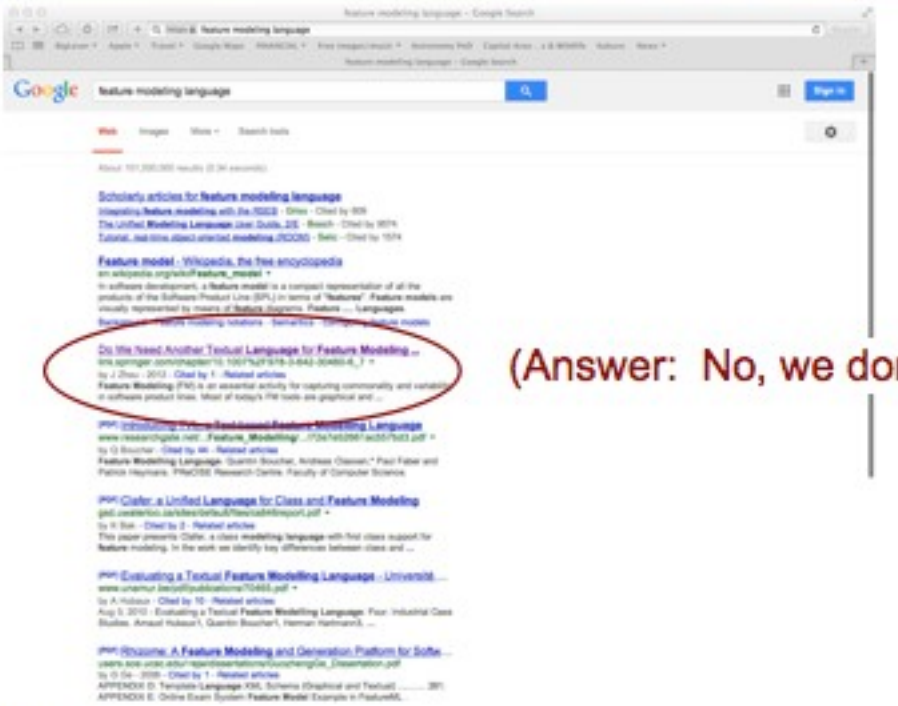
- Do the same thing for a product line of light bulbs.



Team 3 Feature Modeling Exercise

- Do the same thing for *one subsystem* of a modern automobile.

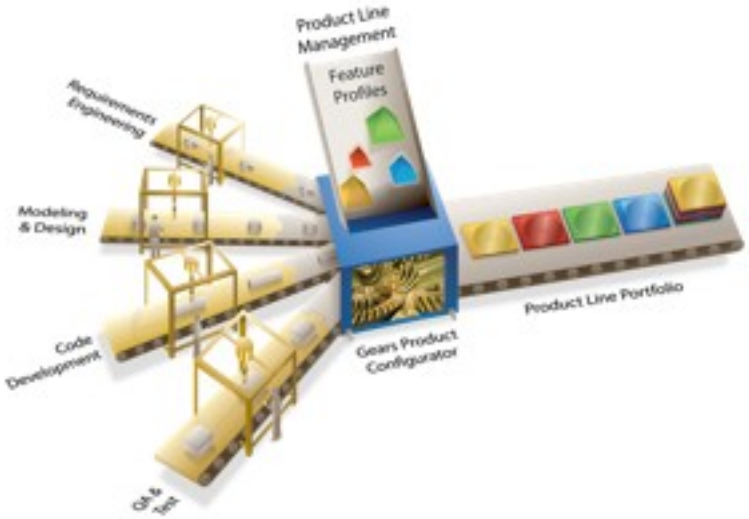




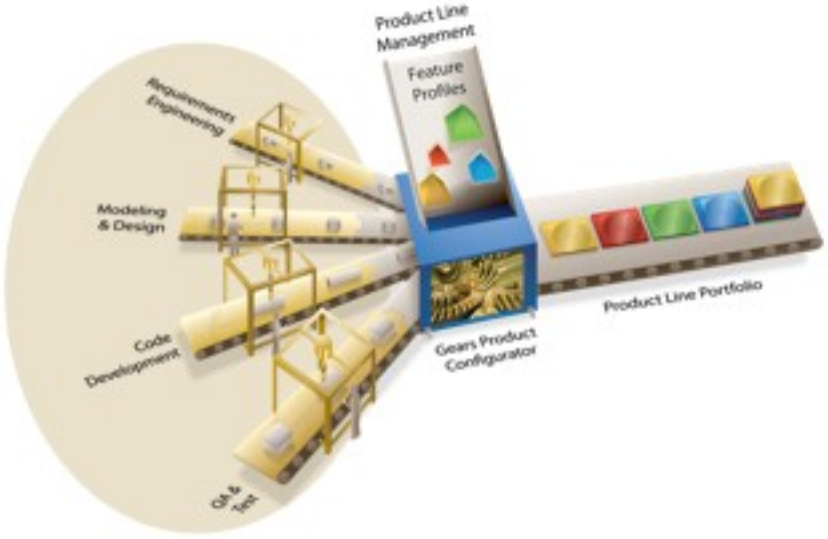
(Answer: No, we don't.)

Introduction to Shared Asset Engineering for PLE

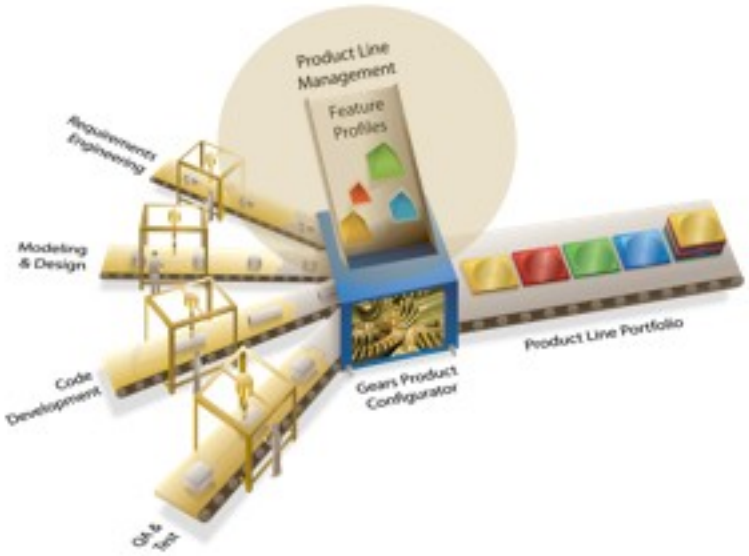
The PLE Factory



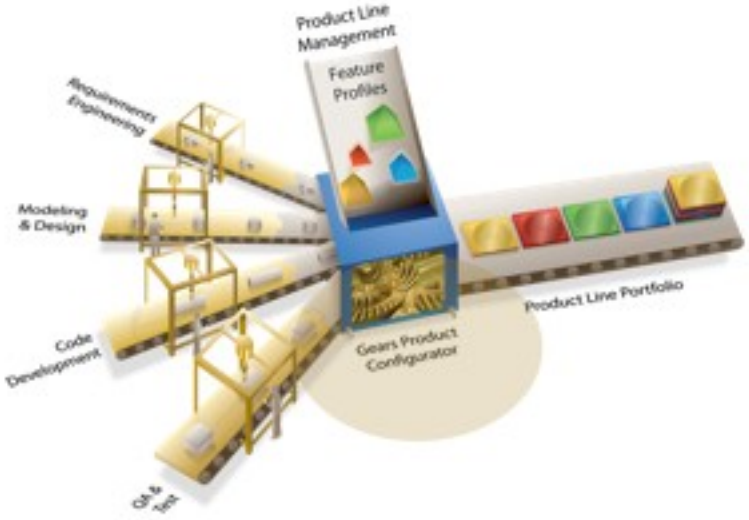
Shared assets are like the factory's supply chain.



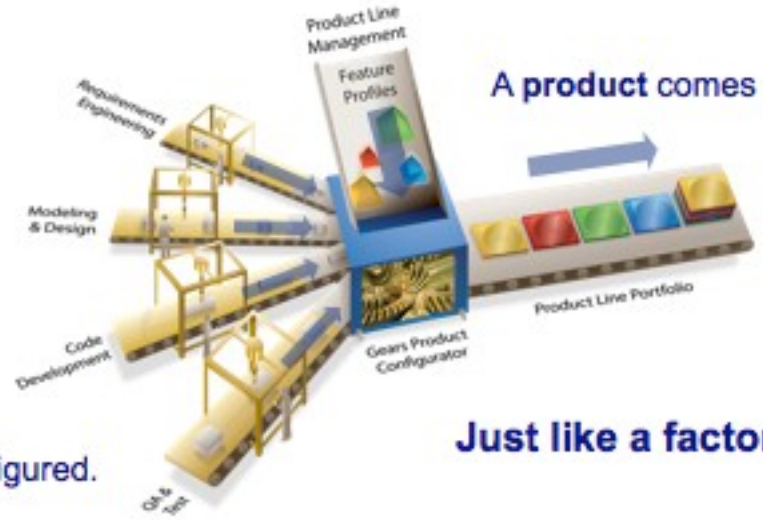
Features describe capabilities that vary among products.



Assets are configured according to the *feature profiles* of the products you want build.



Features come in.



A **product** comes out.

Assets are configured.

Just like a factory.

Demo of Gears and Variation Points in Shared Assets

Types of Variation

- Asset level variation
  - An entire asset is included or omitted
  - One variant of an asset is chosen to be included (out of *n* available)
  - One variant of an asset is chosen to be included, but altered by regular expression pattern substitution

Types of Variation

- Asset level variation
- Element level variation
  - One or more elements *inside* the asset is included or omitted
  - A variant of an element is included (out of *n* available)
  - An element inside the asset is included, but altered by regular expression pattern substitution

- Examples of *elements*
  - A requirement or requirement section
  - A block of source code
  - A block of text (plain or, for example, in a Word document)
  - A row or column of a spreadsheet
  - A model element in a UML diagram (object, class, behavior...)
  - A test case