Lecture 2: Domain and Application Engineering

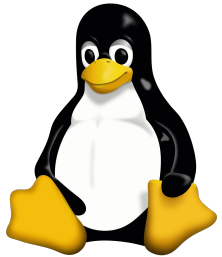
Gregory Gay
TDA/DIT 594 - November 5, 2020

Today's Goals

- Introduce Domain Engineering
  - (a process of developing Software Product Lines and other complex systems)
  - Domain and Application Engineering
    - Platform vs Specific Application
    - Design FOR and WITH reuse
  - Principles of SPLE
    - BAPO Model: Business, Architecture, Process, Organization

Software Product Lines

- Highly configurable families of systems.
- Built around common, modularized features.
  - Common set of core assets.
- Allows efficient development, customization.
- Examples:



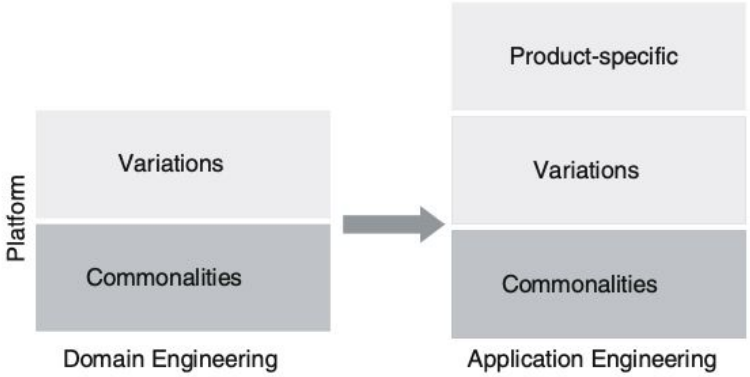

Domain and Application Engineering

SPLE Principles

- Variability Management
  - Variability must be planned for.
- Business-Centric Development
  - Product line must connect to long-term business strategy
- Architecture-Centric Development
  - Code takes advantage of similarities between systems
- Two-Life-Cycles
  - Domain Engineering, followed by Application Engineering

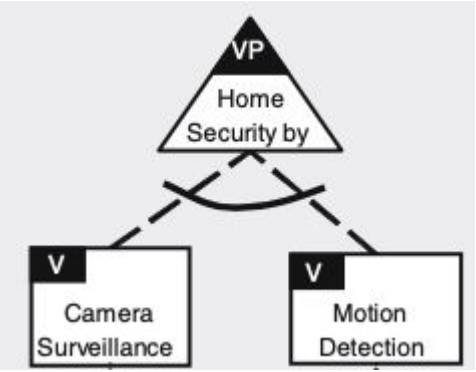
Variability Management

- Commonality
  - Shared between all products.
  - Implemented in the platform.
- Variability
  - Unique to subset of products.
  - Implemented so it is only in that subset.
- Product-specific
  - Something unique to a single product.
  - Platform must support unique adaptations.



Reasoning about Variability

- **Variation Point**
  - A point where a concrete system can differ from another.
  - Ex: which features are supported by this security alarm?



- **Feature**
  - The options that can be chosen at each variation point.
  - Ex: Motion detection, camera

Constraints on Variability

- Variability Dependencies
  - Dependencies for one variation point.
  - How many features can we choose from?
  - Which are mandatory? Optional?
- Feature Dependencies
  - Dependencies between features.
  - Choosing one feature requires also choosing another.
  - Choosing one feature excludes another.

Features and Products

- Any end-user-visible characteristic or behavior of a system is a **feature**.
  - (often, functionality a user can directly interact with)
- A concrete **product** is a valid **feature selection**.
  - Fulfills all **variability and feature dependencies.**

Application Engineering

- Should requirements for a concrete application become part of the product line platform?
  - If supported by the platform, add it to the platform.
    - (ex: can be added as an asset/tied to a variation point)
  - Else:
    - 1) Drop it.
    - 2) Add a new variation point/variant to the platform.
    - 3) Develop it as a unique part of this application.

Business-Centric Development

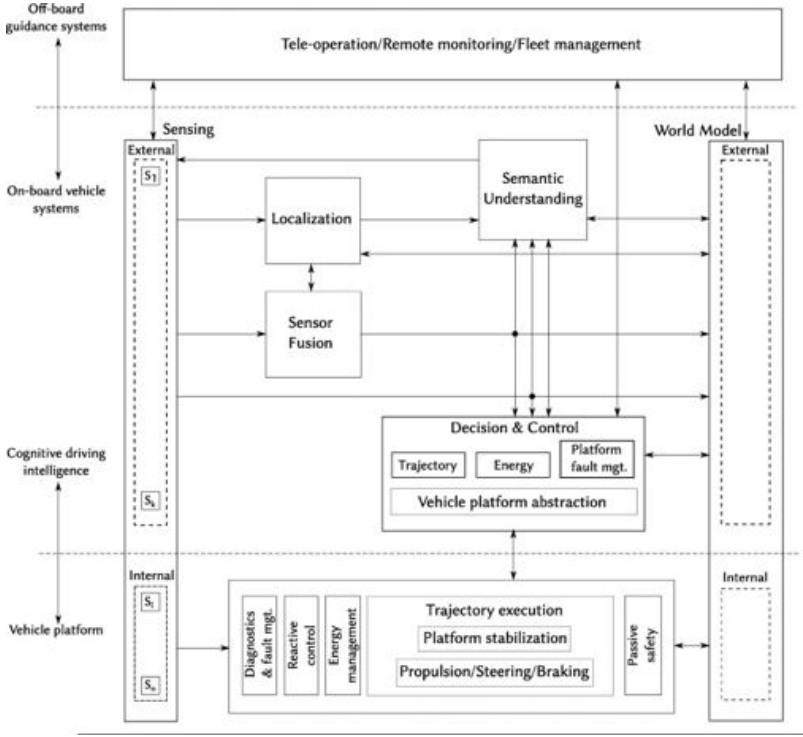
- Up-front planning and investment required.
- Long-term return on investment?
  - Does it make sense to implement a requirement as part of the platform or in one product?
  - 3+ concrete products: make it part of product line.

Scoping

- Product Portfolio Planning
  - Which products are we going to make?
  - How do they differ?
- Domain Potential Analysis
  - Will we get ROI on platform creation?
  - How complex should the platform be?
- Asset Scoping
  - Which specific components will be part of the platform?

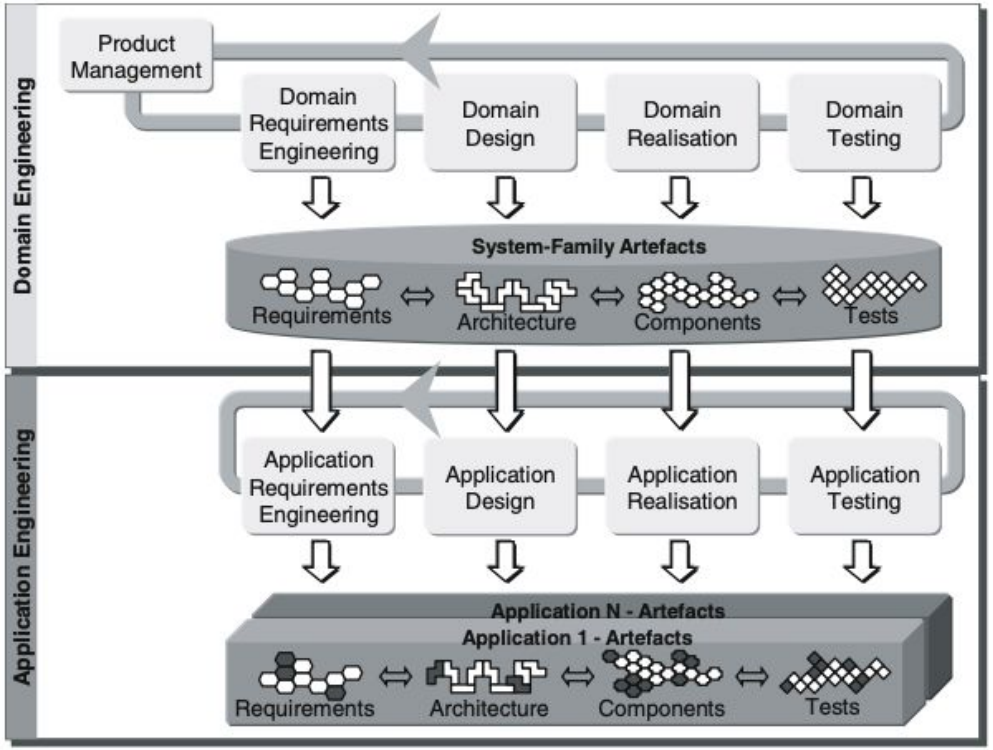
Architecture-Centric Development

- Product lines use **reference architectures**.
  - Common architecture for all products.
  - Variants follow the same interface standards to make them swappable at variation point.
  - Used to create a specific product architecture.



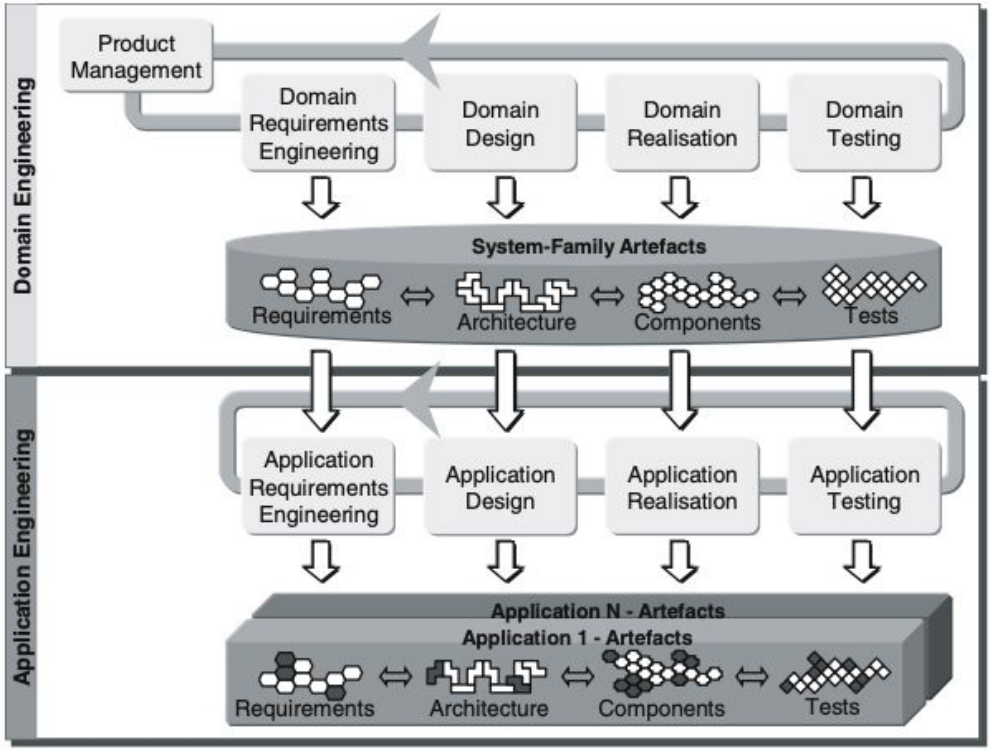
Domain and Application Engineering

- **Domain Engineering**
  - Development for reuse
  - Provides basis for creating individual products.
  - Requirements, design, code, etc. all developed planning for variability.



Domain and Application Engineering

- **Application Engineering**
  - Development WITH reuse.
  - Builds product on top of asset infrastructure.
  - Up to 90% of new product may be built from assets.



What is a Domain?

- An area of knowledge.
  - Scoped to maximize requirement satisfaction.
  - Encompases distinct concepts
  - Defines how to build systems in this area.
- High-Level Domains: databases, social networks, deep learning
  - Deep learning subdomains: classification, language processing, decision support, ...

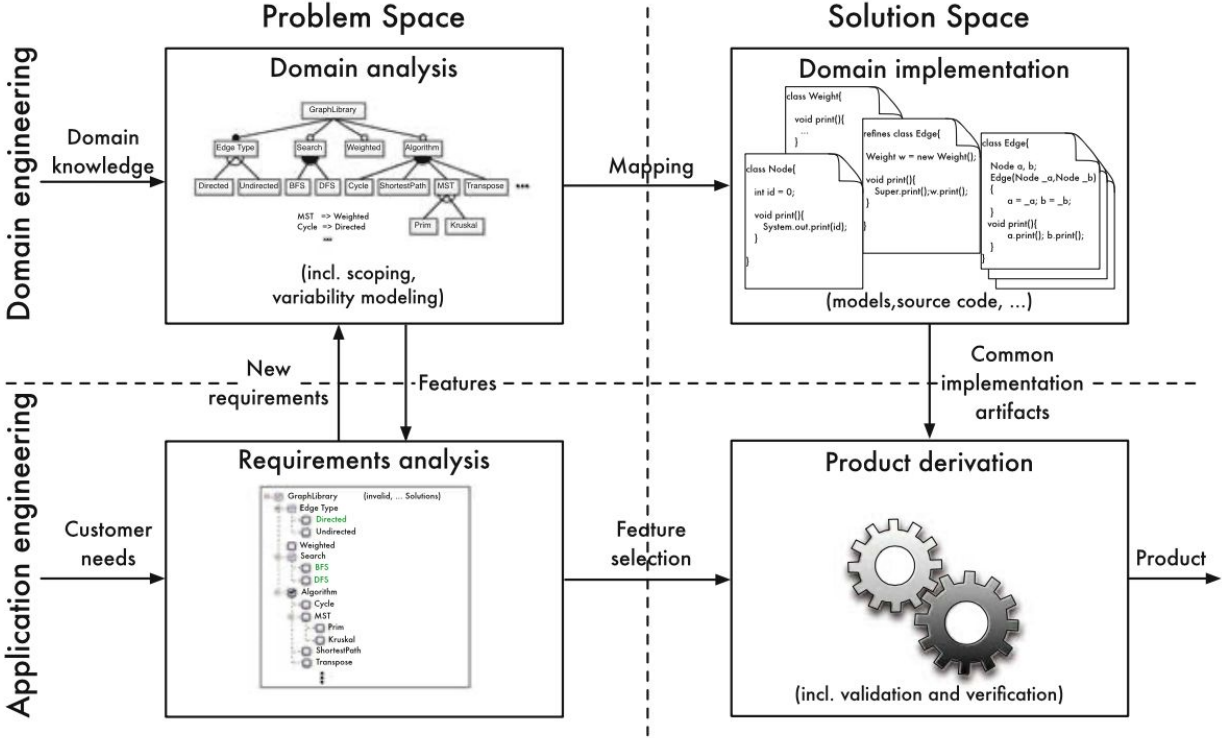
Problem and Solution Space

Problem Space

- Stakeholder's view
- Characterized by features

Solution Space

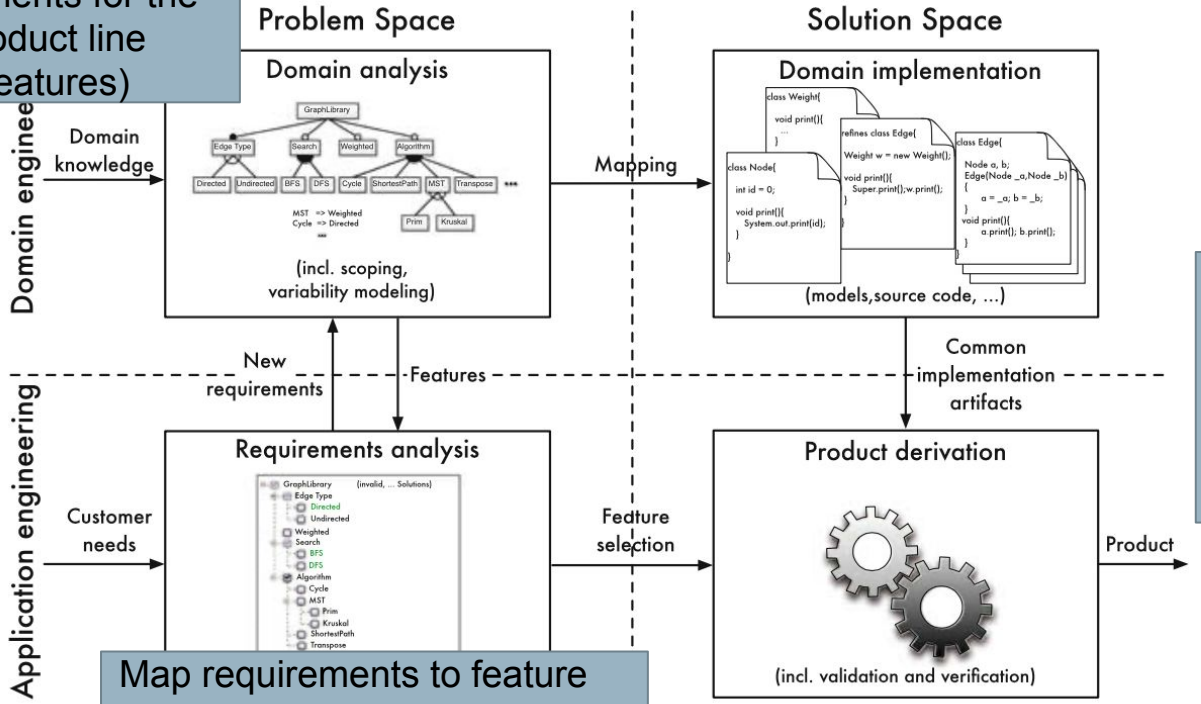
- Developer's view
- Characterized by code structure
- Implementation of features.



Key Task Clusters

Develop reusable assets.

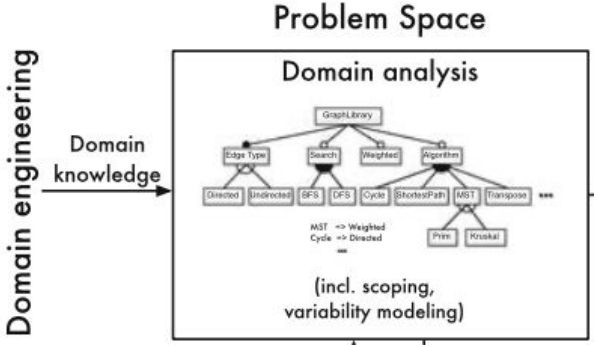
Requirements for the entire product line (scope, features)



Map requirements to feature selection, assess new requirements

Map requirements to feature selection, assess new requirements

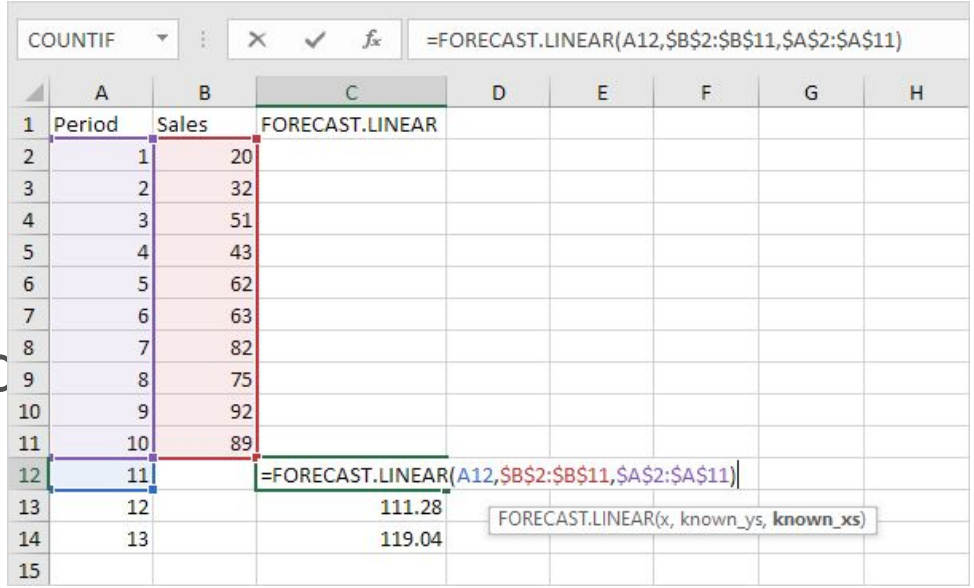
Domain Analysis



- Domain Scoping
  - Deciding on extent of product line
  - Features to support.
  - Trade-off between effort and customer range.
- Ex: Embedded Database Domain
  - Definite Features: Transactions, Recovery, Encryption, Queries, Aggregation, Multi-OS (eCos, TinyOS, Linux),
  - Out-of-Scope: Cloud Storage
  - Consider: Multi-User Support

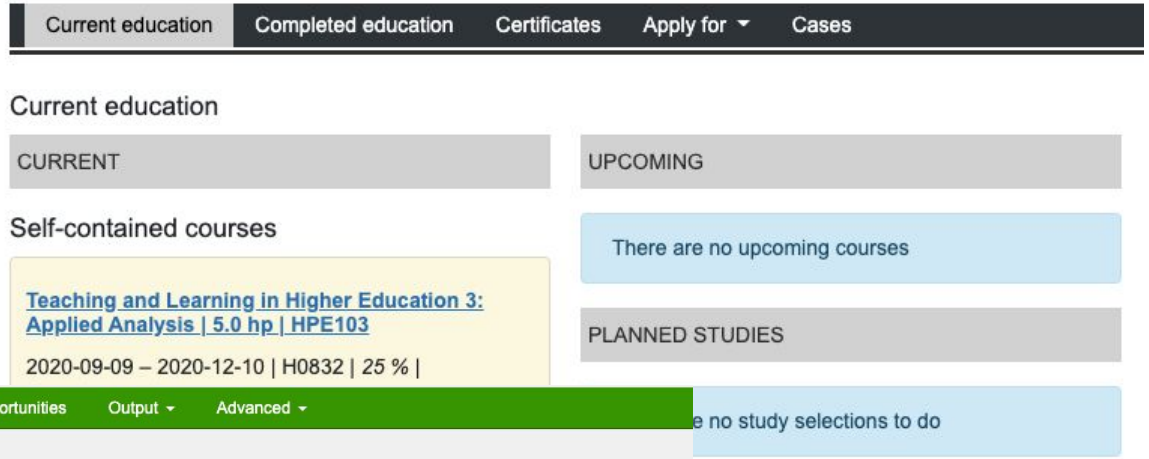
Example: Spreadsheets

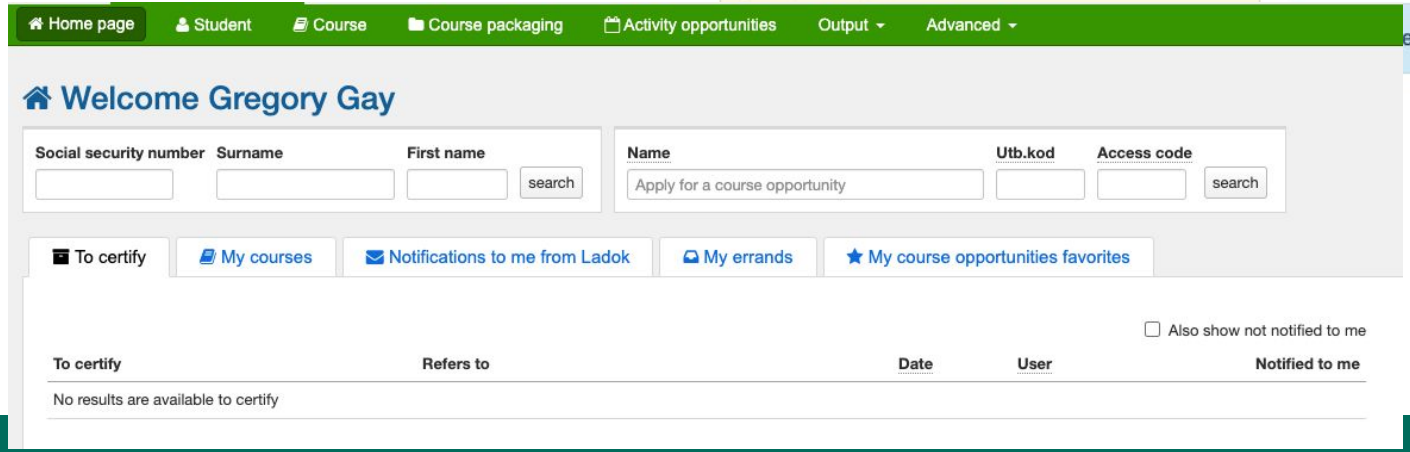
- Look at existing products: Excel, Google Sheets, …
- What are some features a user would expect?



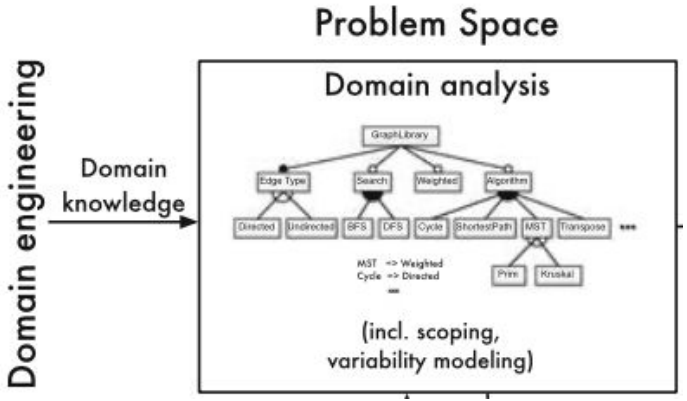
Example: Student Data Management (Ladok)

- Product Line: Student App, Teacher App





Domain Analysis

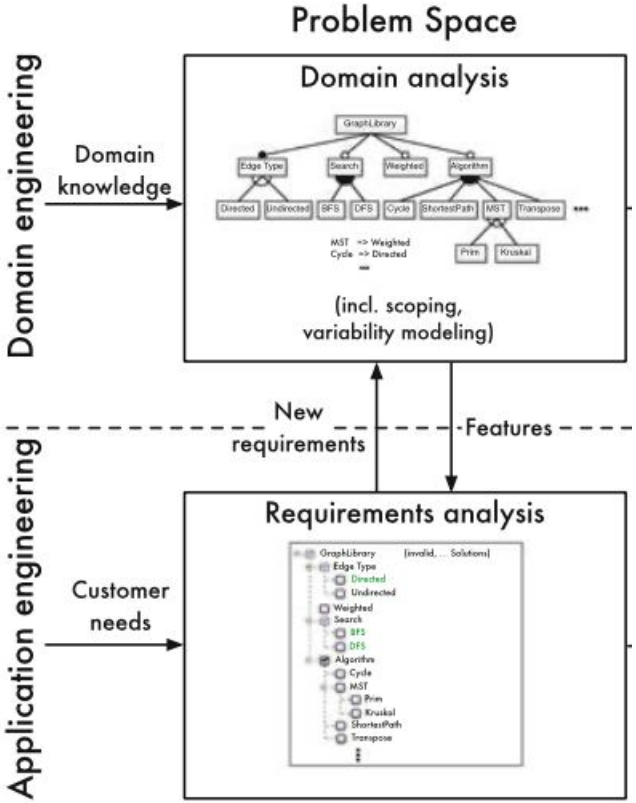


- Domain Modeling
  - Document the commonalities and differences between products in terms of features and dependencies.
  - Ex: Embedded Database
    - Features: Storage, Transactions, OS, Encryption
    - Storage, OS are mandatory.
    - Only one OS supported per product.

Let’s take a break!

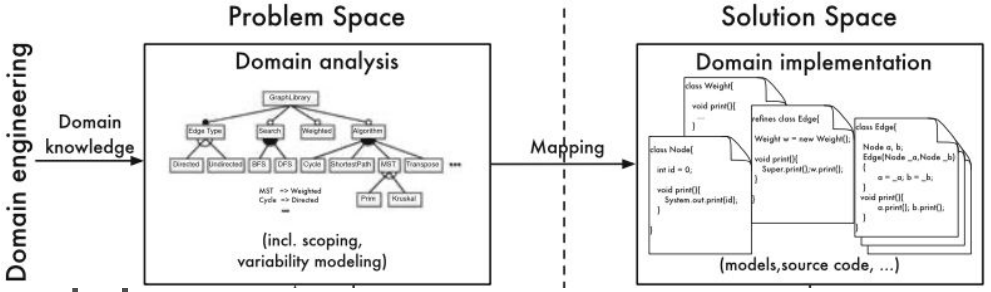
Requirements Analysis

- Map customer requirements to domain requirements.
- If requirements do not map to existing features:
  - Out of scope
  - Assemble as much as possible from reusable features, customize
  - Extend reusable assets with new/changed features.



Domain Implementation

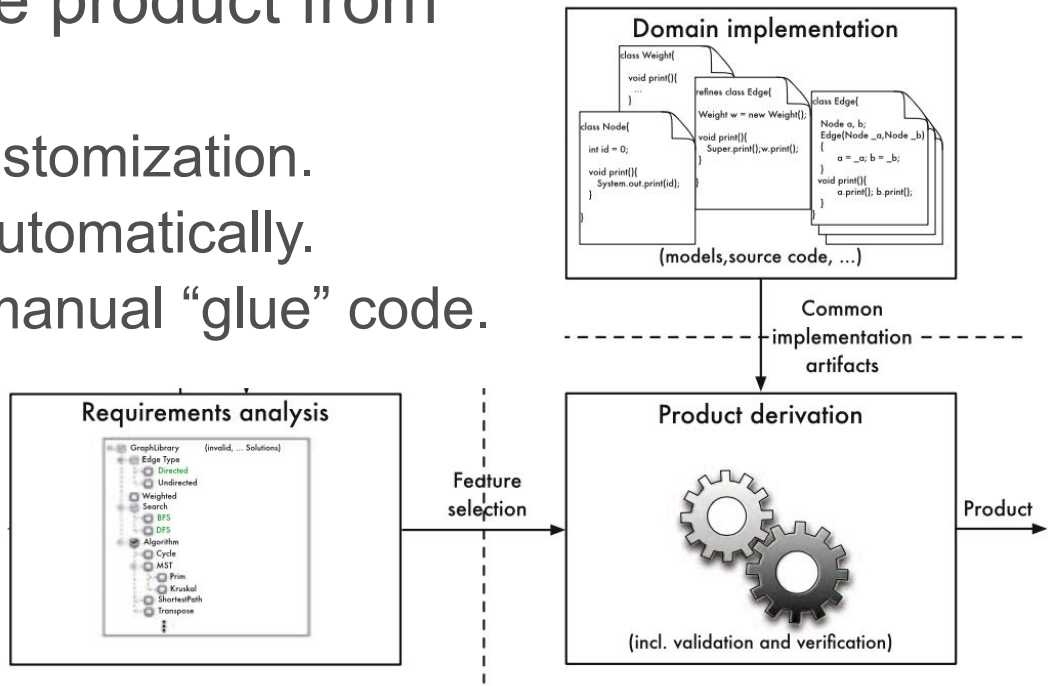
- Implement reusable assets from domain requirements.



- Strategy for combining modules.
  - Compile-time: only include requested code
  - Run-time: bind to class/service when executed
- Interfaces for “attaching” variable features.
  - How to implement variation points.

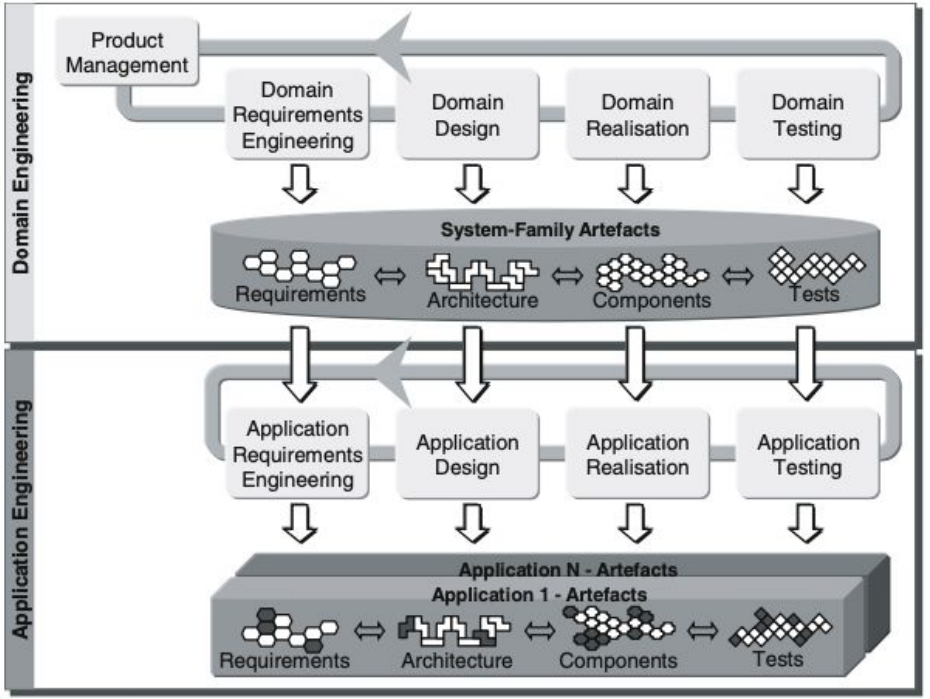
Product Derivation

- Build the final concrete product from reusable assets.
  - Add any necessary customization.
  - Ideally, can be done automatically.
  - Often requires some manual “glue” code.



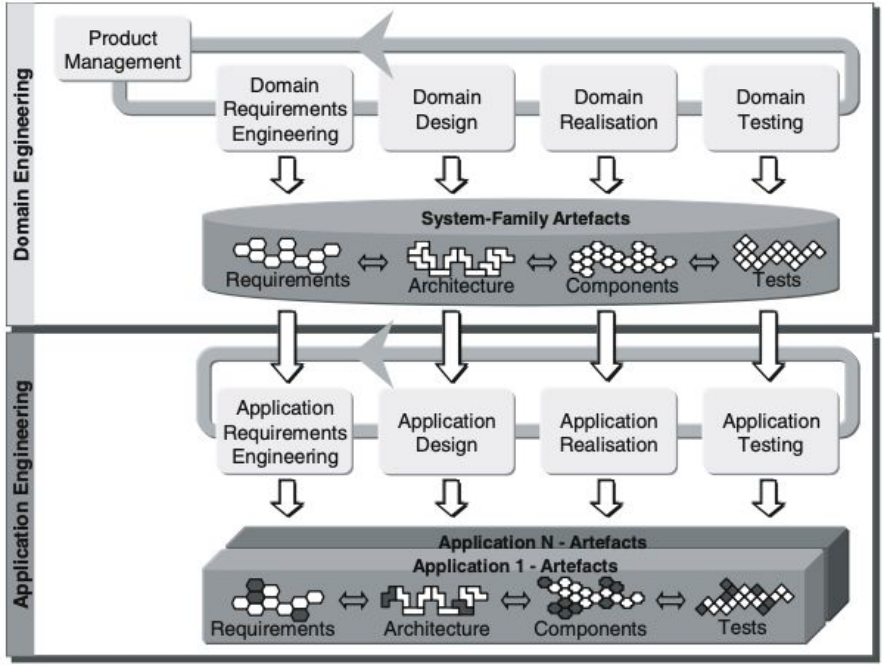
Two-Life-Cycle Approach

- Domain Engineering
  - Develop reusable assets
  - Designed for long-term, complex development.
- Application Engineering
  - Develop individual systems using platform.
  - Designed to deal with rapid changes.



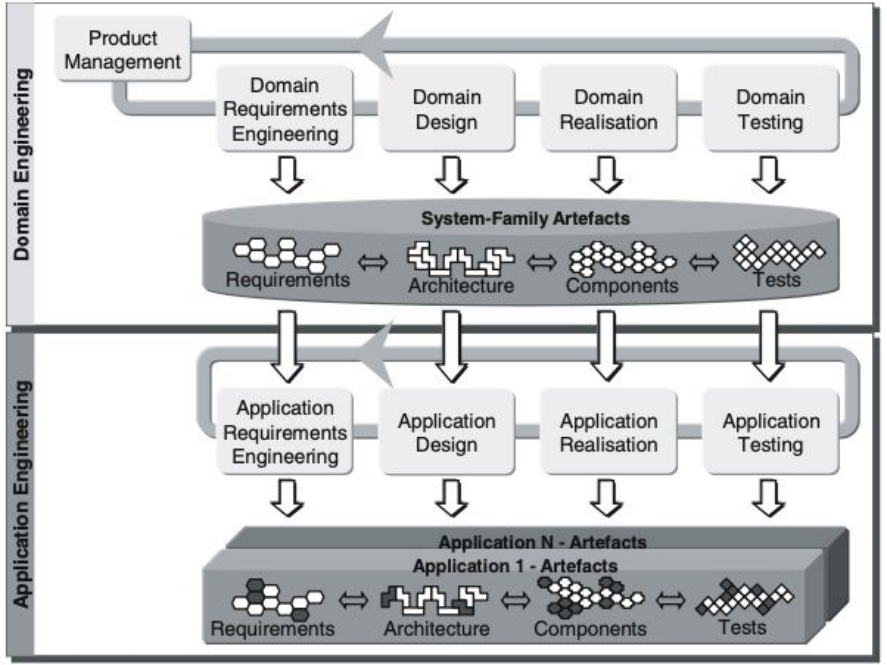
Domain Engineering Activities

- Product Management
  - Portfolio planning, economic analysis
  - Creates product roadmap
- Domain Requirements Engineering
  - Requirements for the platform, identification of variation points/variants.



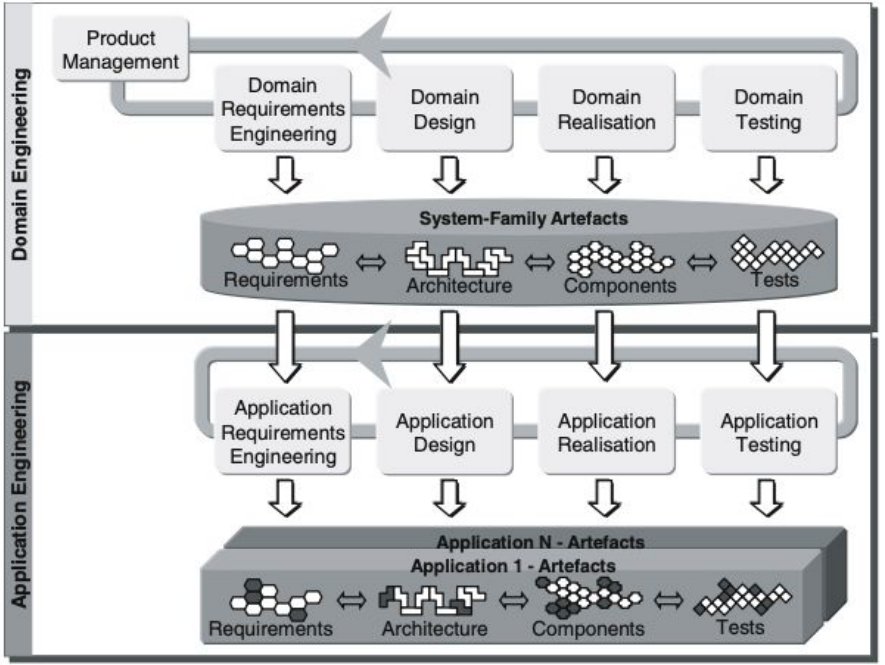
Domain Engineering Activities

- Domain Design
  - Create reference architecture.
- Domain Realization
  - Design and implement reusable assets.
- Domain Testing
  - Test assets in isolation, generate test data for integration in concrete applications.



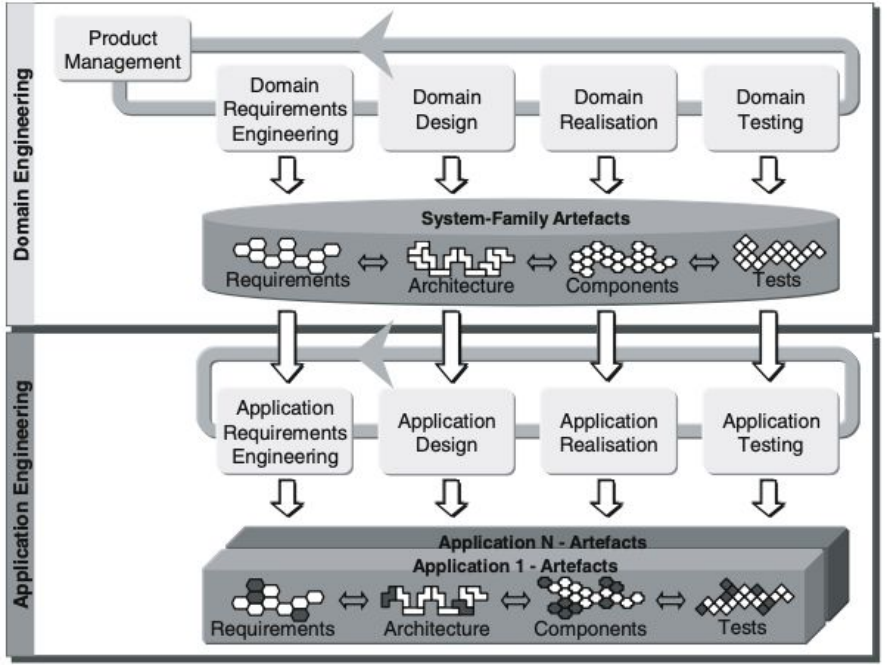
Application Engineering Activities

- Application Requirements Engineering
  - Requirements for the specific product, starting from existing variabilities.
- Application Design
  - Instantiates reference architecture, adds specific adaptations.



Application Engineering Activities

- Application Realization
  - Reuse and configure existing assets, build new components.
- Application Testing
  - Test new components and integration of reused assets.

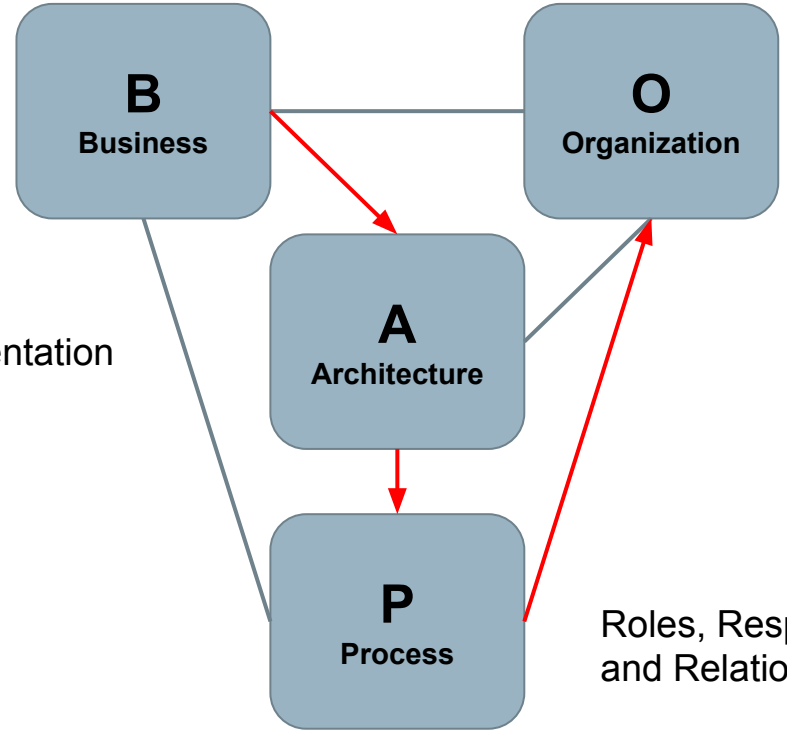


Additional SPLE Concerns

BAPO Model

Strategy and Planning

Technical Implementation



Hosting and Assignment of Responsibility

Roles, Responsibilities, and Relationships

Business Concerns

- Requires significant up-front planning. However…
  - Reduction to < 50% time to market.
  - > 70% smaller code size
  - > 20% reduction in maintenance costs
  - > 20% cheaper to operate
  - Common look and feel = happier customers
  - Features propagate to new products quickly
  - Many more fixed bugs

Architecture Concerns

- Domain architects design the reference architecture
  - Enables reuse of code, tests, other artifacts.
  - Important to control variability.
  - Ensure requirements do not conflict.
  - Ensure architecture can be changed over time.
- Application architects specialize the architecture to match application requirements.
  - Decide what to promote to the platform.

Process and Organization Concerns

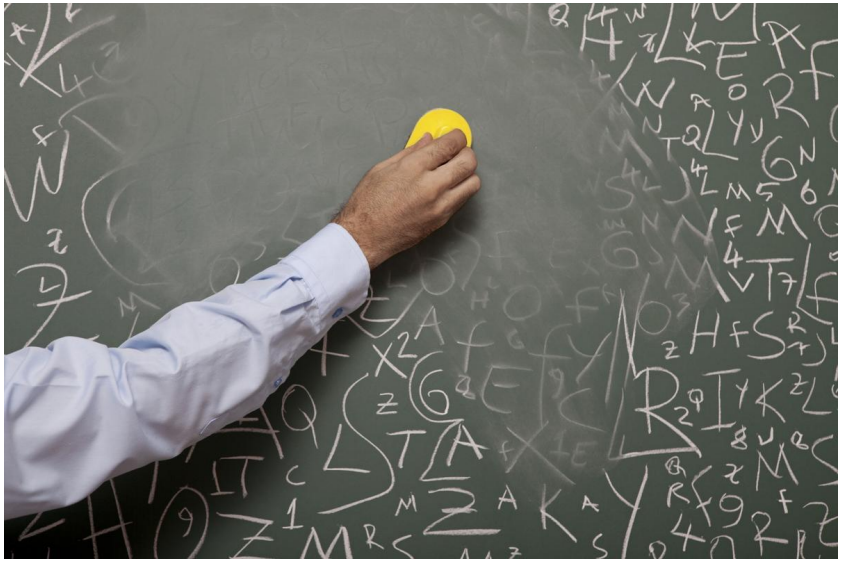
- Additional coordination needed between domain and application engineering efforts.
- Often separate domain and application engineers.
  - Domain engineers develop and maintain assets.
  - Application engineers quickly combine assets.
  - Specialists coordinate between domain and application.

Transitioning to a Product Line

- Proactive
  - Develop full SPL from scratch.
- Extractive
  - Start from existing products and refactor into a SPL.
- Reactive
  - Build a small SPL and extend it over time.

Proactive Approach

- Build from scratch.
  - Existing products halt development, are re-implemented.
- High quality products, reduced long-term costs.
- Requires ***SIGNIFICANT*** up-front investment.

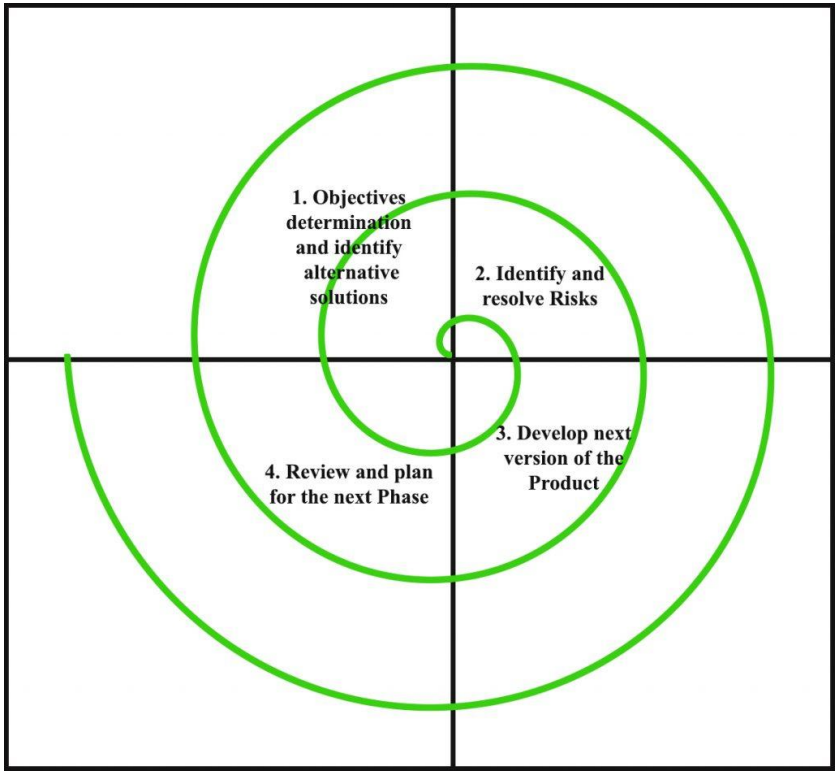


Extractive Approach

- Transition from existing products to product line.
  - Extract functionality as reusable assets.
  - Implement variation points to attach assets.
  - Done over time, while products remain in-service.
- Requires much less up-front cost.
- Code quality may suffer.

Reactive Approach

- Implement initial SPL.
  - In increments, identify and implement new features.
- Less upfront planning than proactive.
  - Adding unplanned features more difficult.
- More structured than extractive.



We Have Learned

- Domain Engineering
  - Development for reuse. Creates asset portfolio.
    - Provides basis for creating individual products.
    - Requirements, design, code, etc. planned for variability.
- Application Engineering
  - Development WITH reuse.
    - Builds product on top of asset infrastructure.
    - Up to 90% of new product may be built from assets.

Next Time

- Feature Models
  - Models that define and constrain variability.
  - Basis for planning a SPL.
- Team Selection Due Tonight!
  - 6-7 people, e-mail names to ggay@chalmers.se
  - E-mail me if you want to be assigned to a team.
- Assignment 1 out now!

Assignment 1 - Case Study

- **Due November 15, 11:59 PM**
- Case study examining development of a SPL or other reuse-driven system.
  - **Choose a system:**
    - Van der Linden, F. J.,Schmid, K., & Rommes, E. (2007). Software product lines in action: the best industrial practice in product line engineering. Springer Science & Business Media.
    - You may also choose any system with sufficient public information available.

Assignment 1 - Case Study

- **Must get approval from your supervisor!**
- Document:
  - **Context:** What kind of organization/market?
  - **Motivation:** Why a SPL or reuse-driven approach?
  - **Type of System**
  - **Approach:** What engineering practices?
  - **Challenges:** Key technical and process challenges.
  - **Results:** What happened?
  - **Conclusions:** What did they learn?

Robocode Introduction

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY