CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 2: Domain and Application Engineering

Gregory Gay

TDA 594/DIT 593 - November 4, 2021

Today’s Goals

- Introduce Domain Engineering
  - Domain and Application Engineering
    - Platform vs Specific Application
    - Design FOR and WITH reuse
  - Principles of SPLE
    - BAPO: 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:

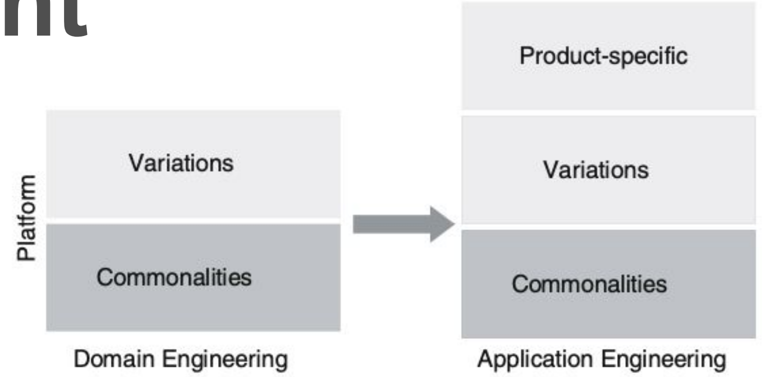
Domain and Application Engineering

SPLE Principles

- Variability Management
  - Variability must be planned for.
- Business-Centric Development
  - Connect to long-term business strategy.
- Architecture-Centric Development
  - Take advantage of system similarities.
- Two-Life-Cycles
  - Domain Engineering, then Application Engineering.

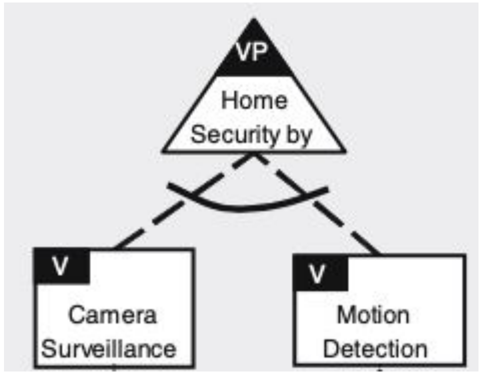
Variability Management

- Commonality
  - Shared between all products.
  - Implemented in core platform.
- Variability
  - Shared by subset of products.
  - Implemented in core platform, enabled in subset.
- Product-specific
  - Unique to a single product.
  - Platform must support unique adaptations.



Reasoning about Variability

- **Variation Point**
  - Where one product can differ from another.
  - Ex: Which features are supported by this security alarm?
- **Feature**
  - Options that can be chosen at each variation point.
  - Ex: Motion detection, camera



Constraints on Variability

- Variability Dependencies
  - Dependencies between features at one variation point.
  - How many features can we choose for this point?
  - Which are mandatory? Optional?
- Feature Dependencies
  - Dependencies between features at same or different variation points.
  - Choosing one feature requires choosing or excluding another feature.

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.
    - (can be added as an asset or tied to a variation point)
  - Else:
    - 1) Drop it.
    - 2) Add a new variation point to the platform.
    - 3) Develop it as a unique part of one application.

Business-Centric Development

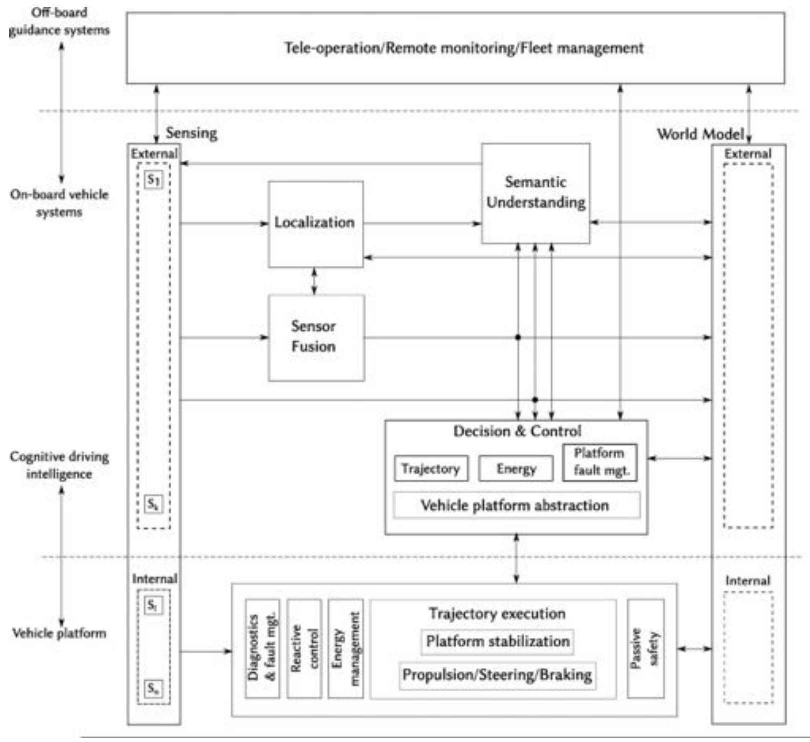
- Up-front planning and investment required.
- Long-term return on investment?
  - Implement requirement as part of platform or in a product?
  - 3+ concrete products: make it part of platform.

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?

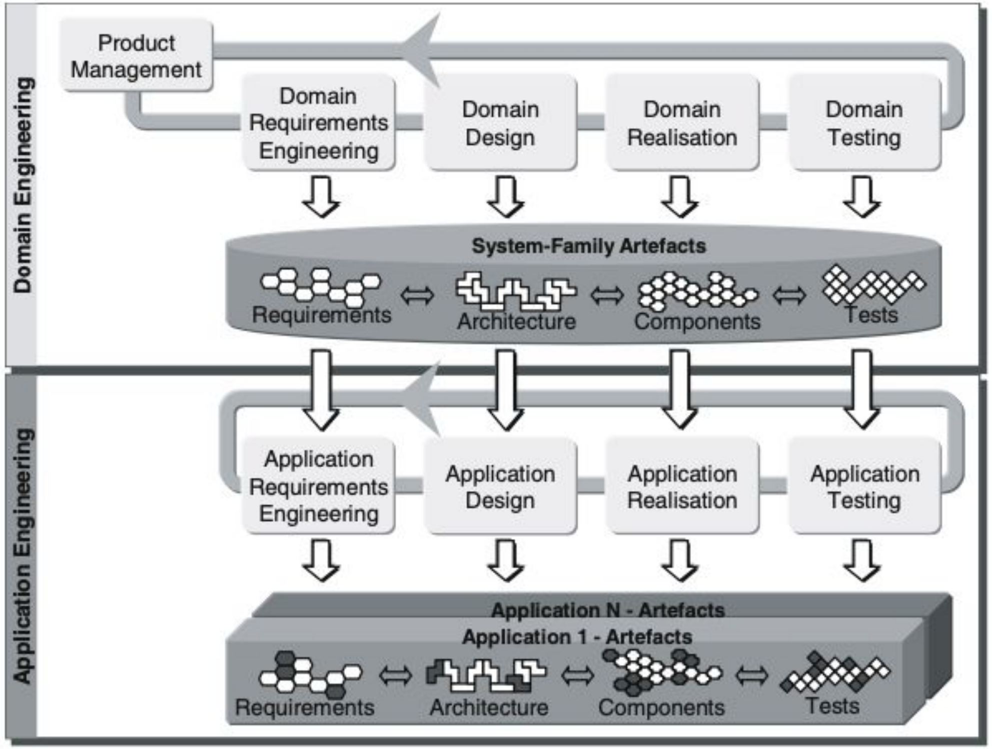
Architecture-Centric Development

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



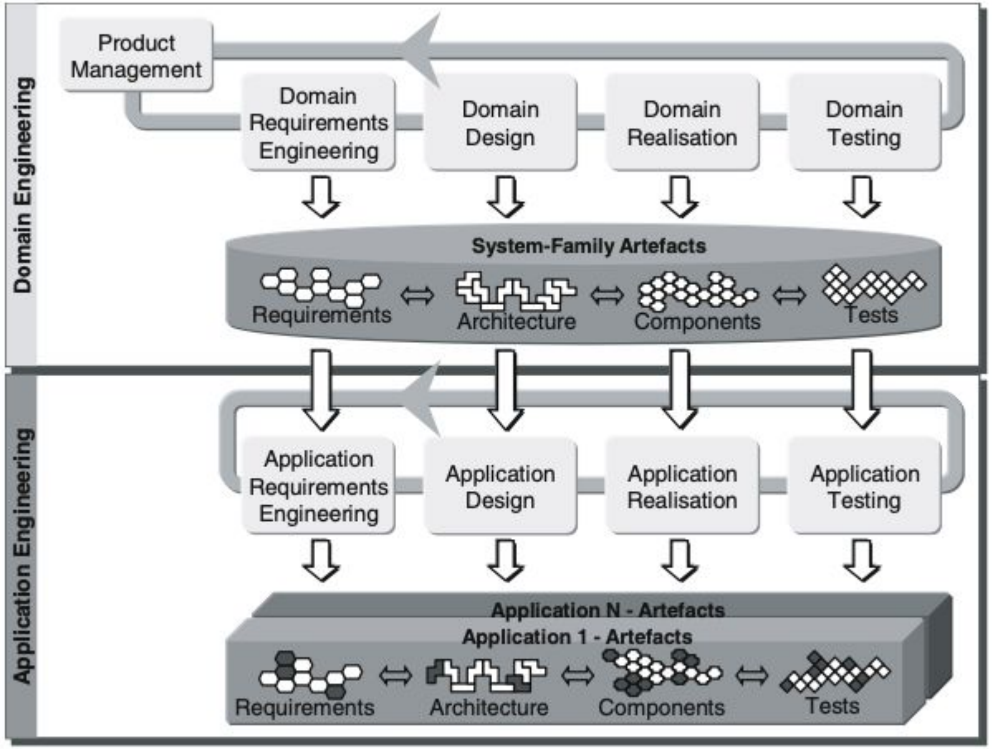
Domain and Application Engineering

- **Domain Engineering**
  - Enables reuse.
  - Basis for creating individual products.
  - Requirements, design, code, etc. all planned for variability.



Domain and Application Engineering

- **Application Engineering**
  - Development based on reuse.
  - Builds product on top of platform.
    - <= 90% of product 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, ...

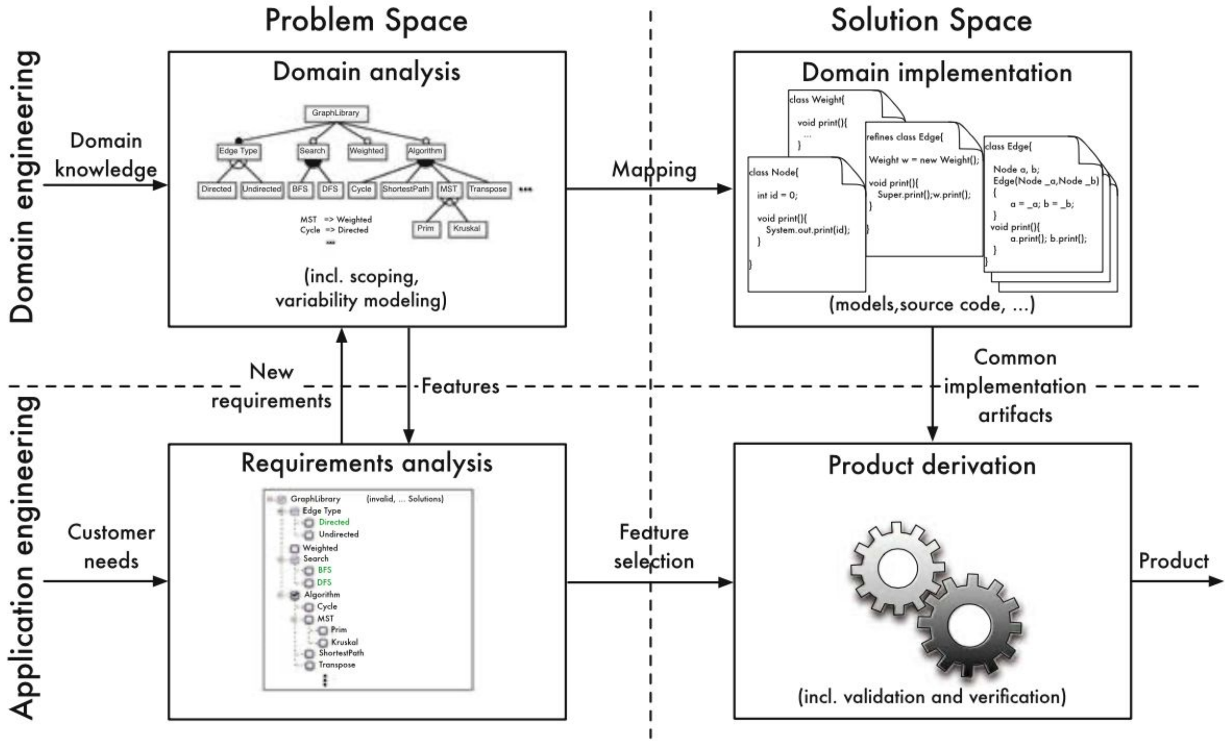
Problem and Solution Space

Problem Space

- Stakeholder view
- Characterized by features

Solution Space

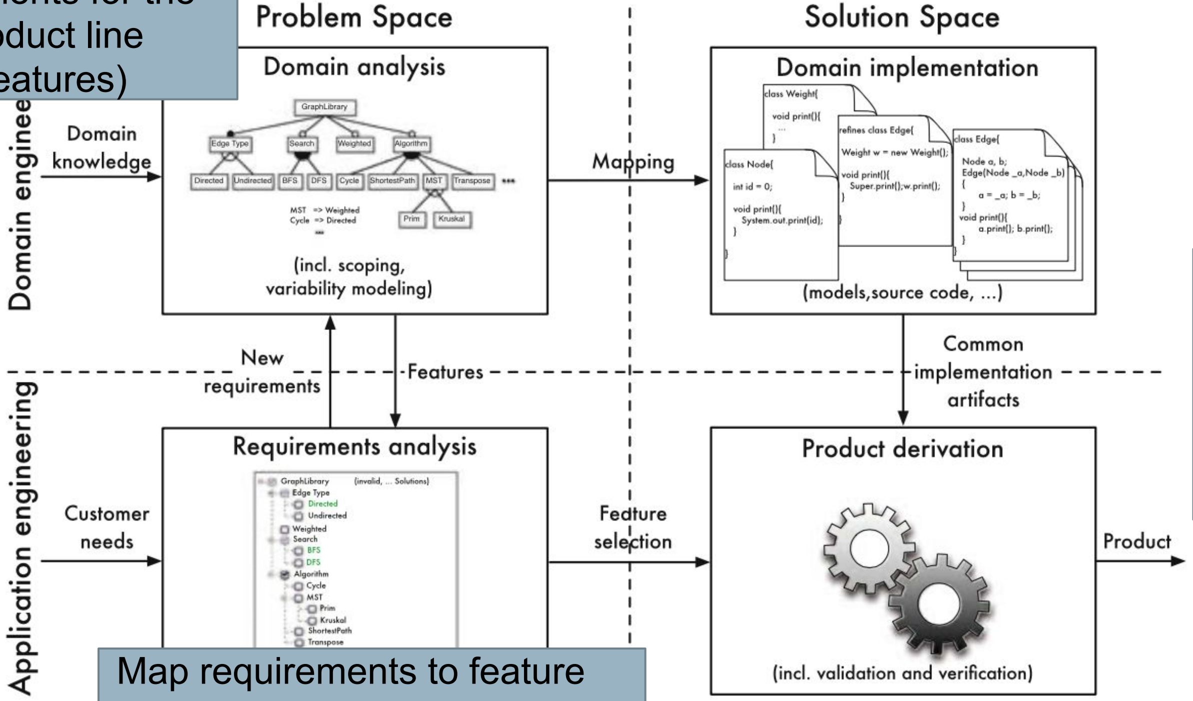
- Developer view
- Characterized by code structure
- Implementation of features.



Key Task Clusters

Develop reusable assets.

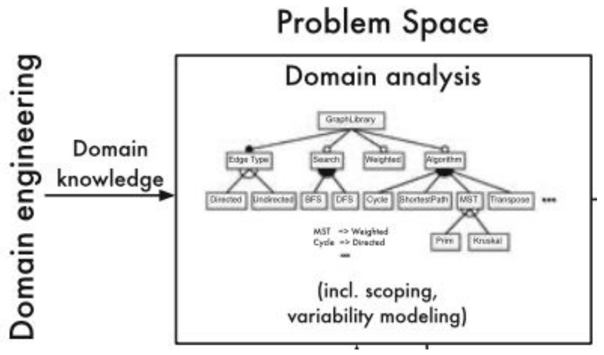
Requirements for the entire product line (scope, features)



Assets combined to form new concrete product.

Map requirements to feature selection, assess new requirements

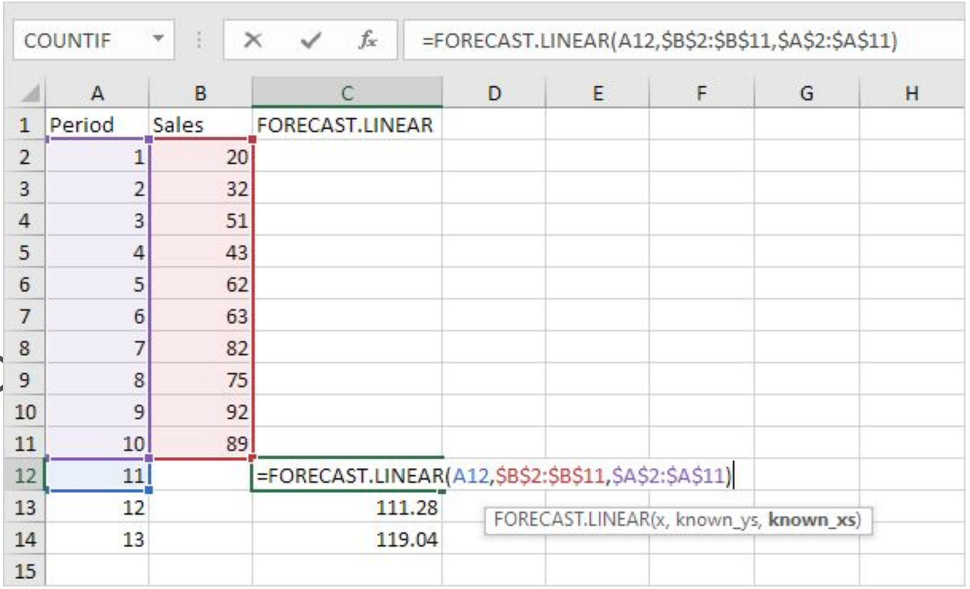
Domain Analysis



- 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?



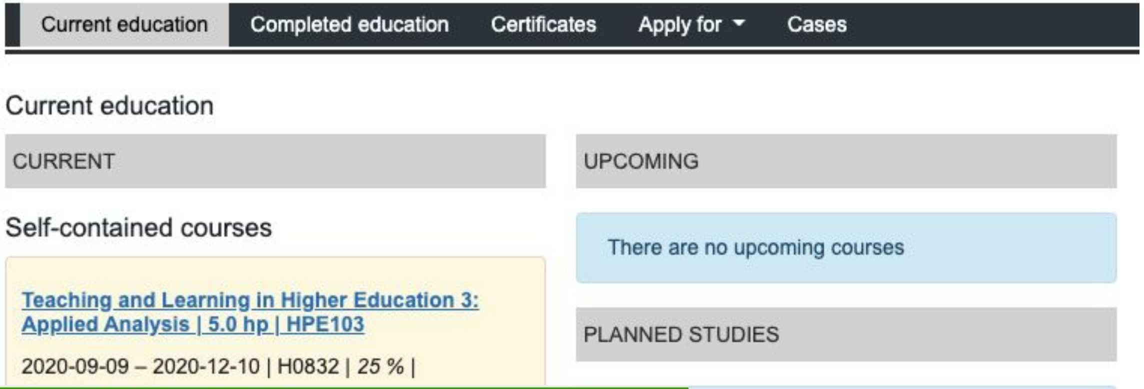
COUNTIF

=FORECAST.LINEAR(A12,$B$2:$B$11,$A$2:$A$11)

	A	B	C	D	E	F	G	H
1	Period	Sales	FORECAST.LINEAR					
2	1	20						
3	2	32						
4	3	51						
5	4	43						
6	5	62						
7	6	63						
8	7	82						
9	8	75						
10	9	92						
11	10	89						
12	11		=FORECAST.LINEAR(A12,$B$2:$B$11,$A$2:$A$11)					
13	12		111.28	FORECAST.LINEAR(x, known_ys, **known_xs**)				
14	13		119.04					
15								

Example: Student Data Management (Ladok)

- Product Line: Student App, Teacher App


Current education | Completed education | Certificates | Apply for | Cases

Current education

CURRENT

Self-contained courses

Teaching and Learning in Higher Education 3: Applied Analysis | 5.0 hp | HPE103

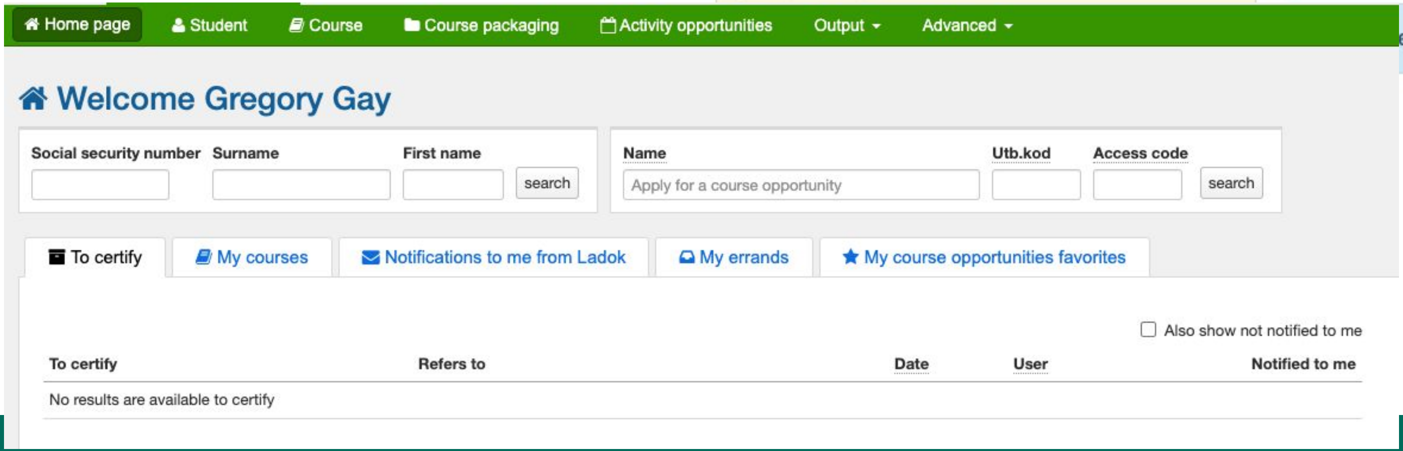
2020-09-09 – 2020-12-10 | H0832 | 25 % |

UPCOMING

There are no upcoming courses

PLANNED STUDIES

e no study selections to do


Home page | Student | Course | Course packaging | Activity opportunities | Output | Advanced

Welcome Gregory Gay

Social security number	Surname	First name	
			search

Name	Utb.kod	Access code	
Apply for a course opportunity			search

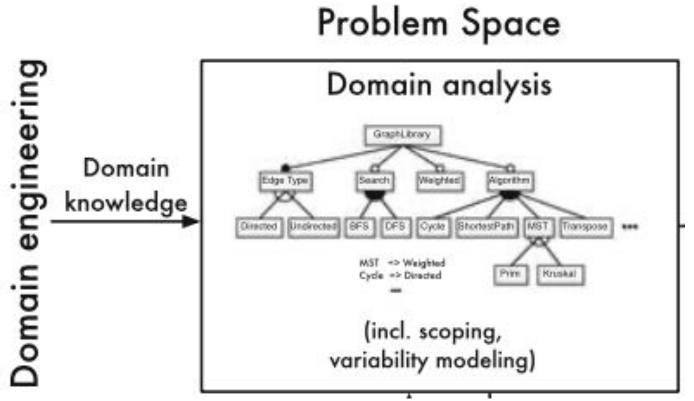
To certify | My courses | Notifications to me from Ladok | My errands | My course opportunities favorites

☐ Also show not notified to me

To certify	Refers to	Date	User	Notified to me
No results are available to certify				

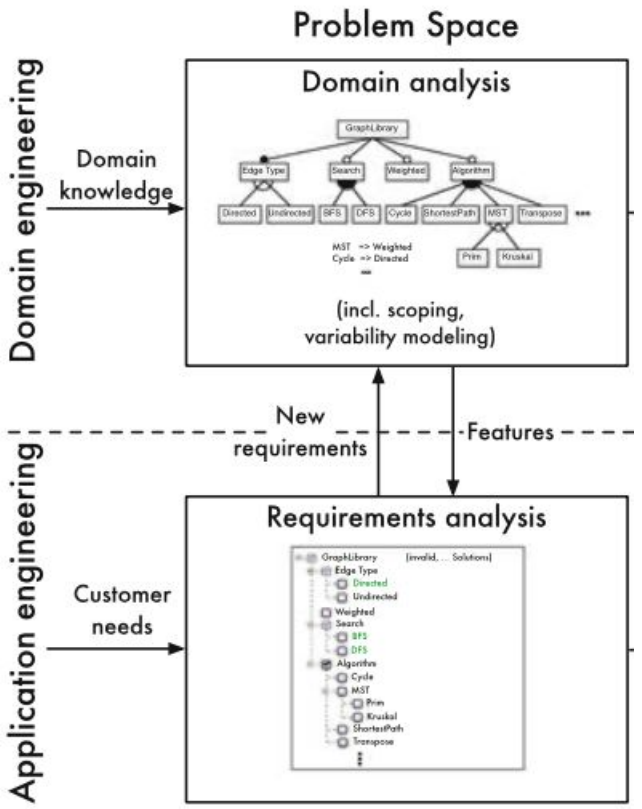
Domain Analysis

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

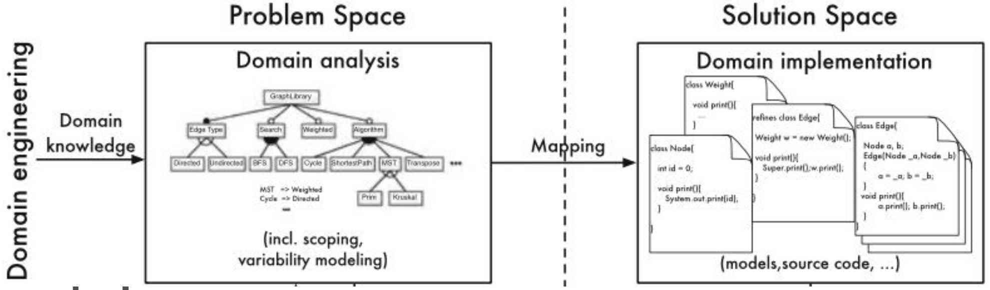


Requirements Analysis

- Map customer requirements to domain requirements.
- If requirements do not map to existing features:
  - 1) Out of scope
  - 2) Do much as possible with features, customize rest
  - 3) Extend platform with new features, variation points.



Domain Implementation



```
class Weight{

  void print(){
   ...
  }
```

```
refines class Edge{

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

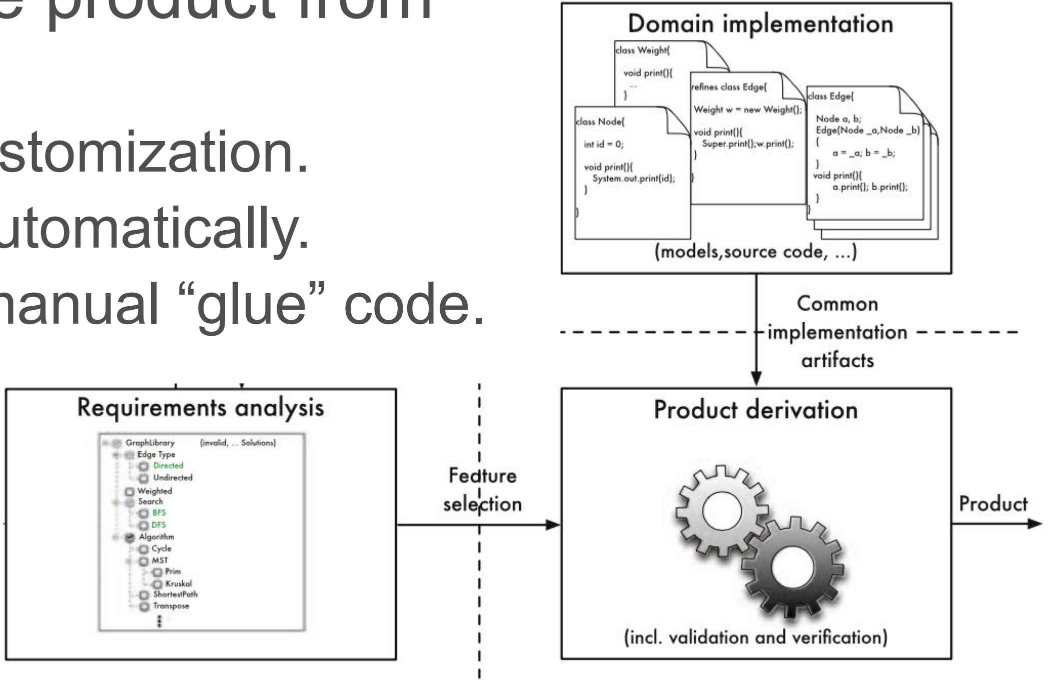
```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
class Edge{

  Node a, b;
  Edge(Node _a,Node _b)
  {
        a = _a; b = _b;
  }
 void print(){
        a.print(); b.print();
  }
}
```

- Implement reusable assets from domain requirements.
- Strategy for combining modules.
  - Compile-time: only include requested code
  - Run-time: include all code, bind when executed
- Interfaces for “attaching” variable features.

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.



```
class Weight{

  void print(){
    ...
  }
```

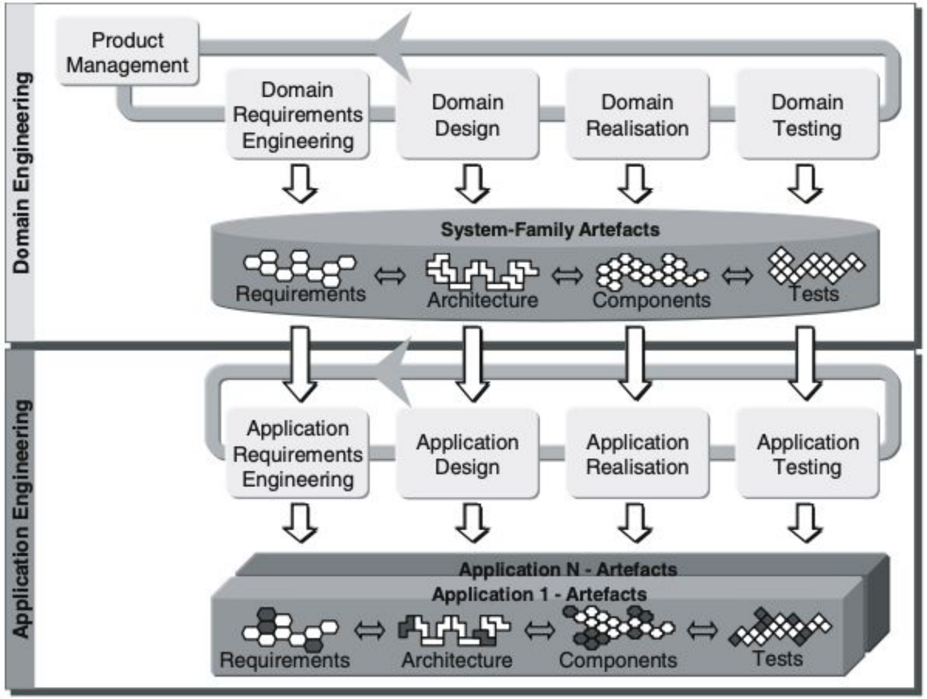
```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
refines class Edge{

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

```
class Edge{

  Node a, b;
  Edge(Node _a,Node _b)
  {
      a = _a; b = _b;
  }
 void print(){
      a.print(); b.print();
  }
}
```

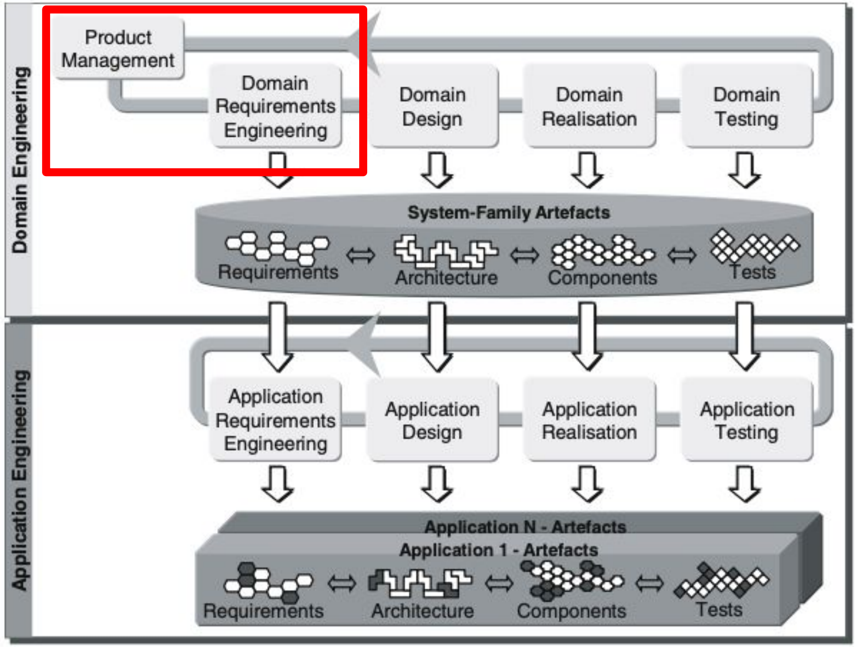
Two-Life-Cycle Approach

- Domain Engineering
  - Develop reusable assets
  - Designed for long-term, complex development.
- Application Engineering
  - Develop products.
  - Designed for current customer, rapid changes.



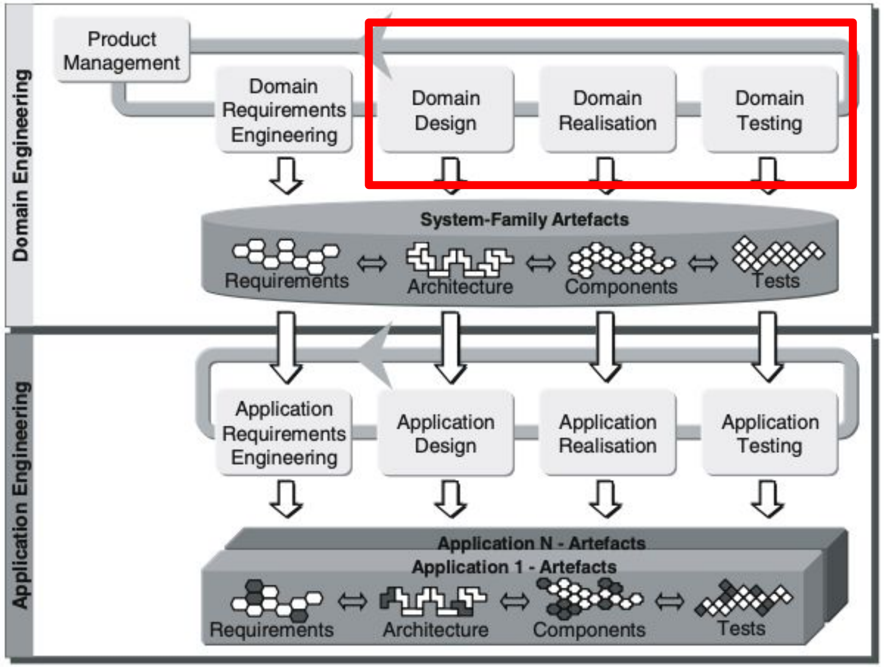
Domain Engineering Activities

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



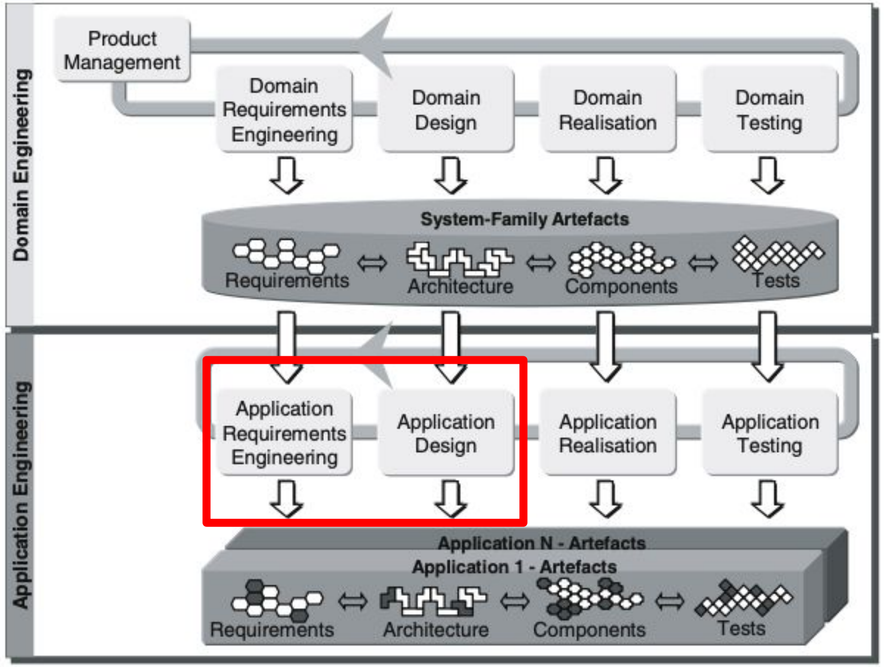
Domain Engineering Activities

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



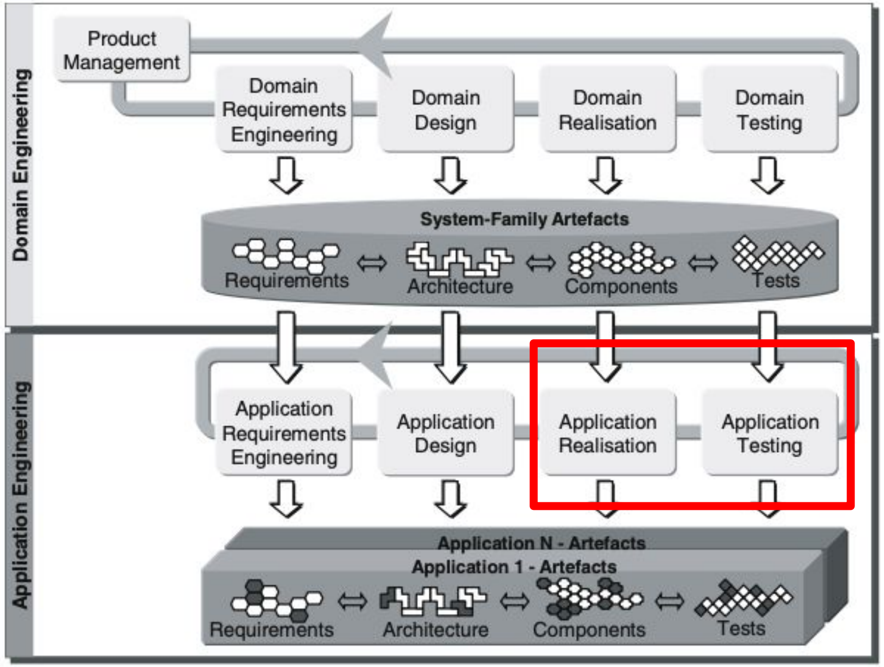
Application Engineering Activities

- 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.



Let’s take a break!

Additional SPLE Concerns

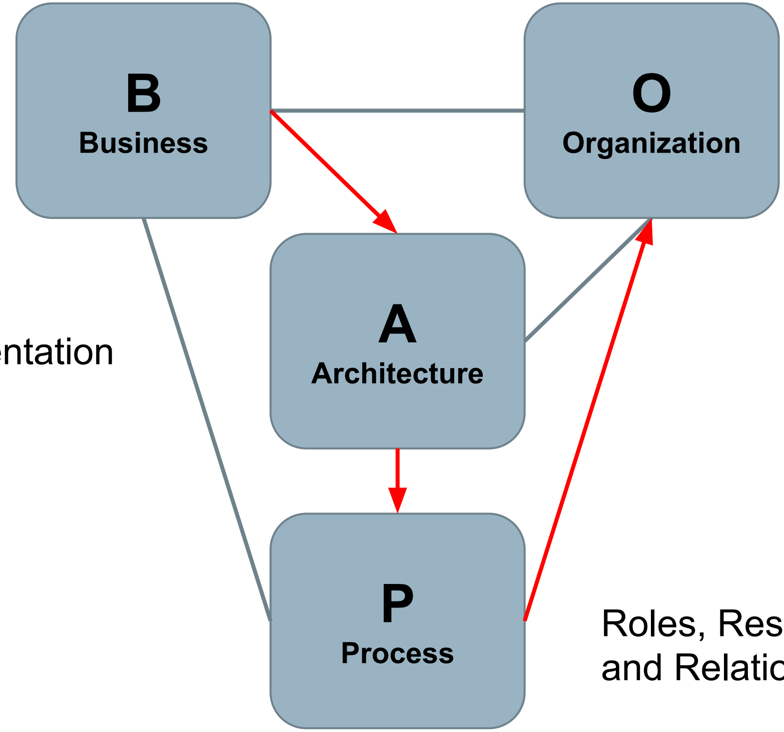
BAPO Model

Strategy and Planning

Hosting and Assignment of Responsibility

Technical Implementation

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

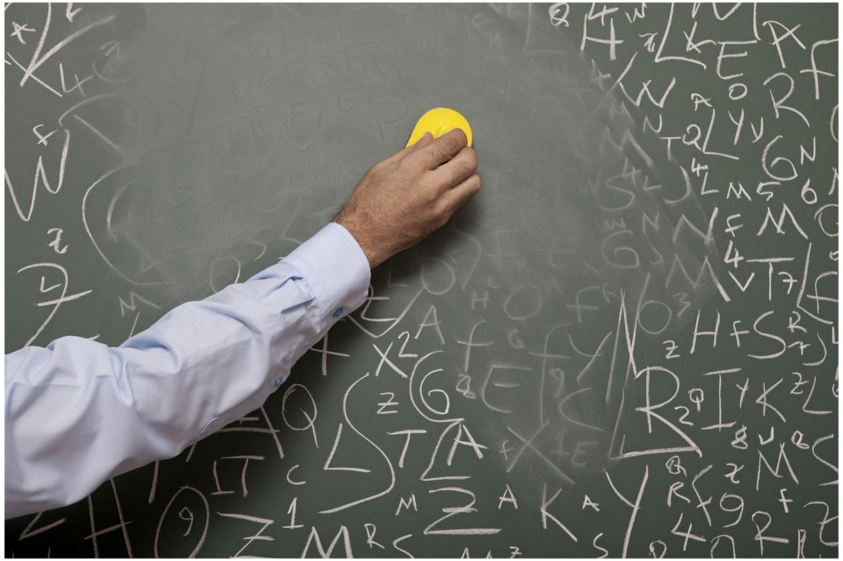
- Coordination needed between domain and application engineering.
- Often separate domain and application engineers.
  - Domain engineers develop and maintain assets.
  - Application engineers quickly combine assets.
  - Specialists coordinate between domain and application.

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 ***<u>SIGNIFICANT</u>*** up-front investment.

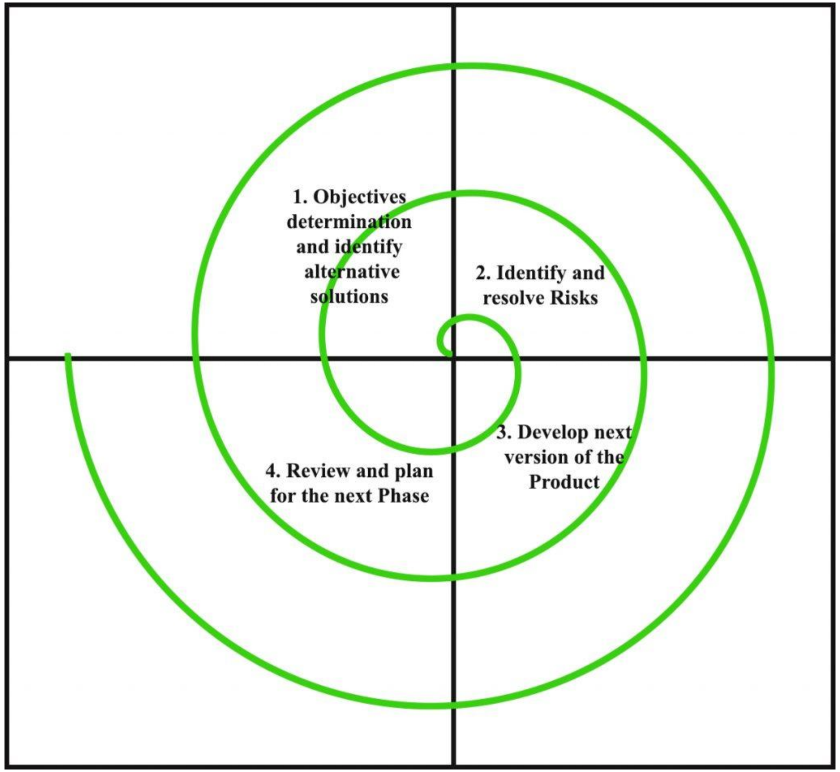


Extractive Approach

- 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.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

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 Modelling
  - Models that define and constrain variability.
  - Basis for planning a SPL.
- Team Selection Due Tonight!
  - 6-7 people, one email per team to ggay@chalmers.se
  - Complete assignment in Canvas
    - (include either team number given to you, or if you want to be assigned to a team)
- Assignment 1 out now!

Assignment 1 - Case Study

- **Due November 14, 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?

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY