CHALMERS
UNIVERSITY OF TECHNOLOGY

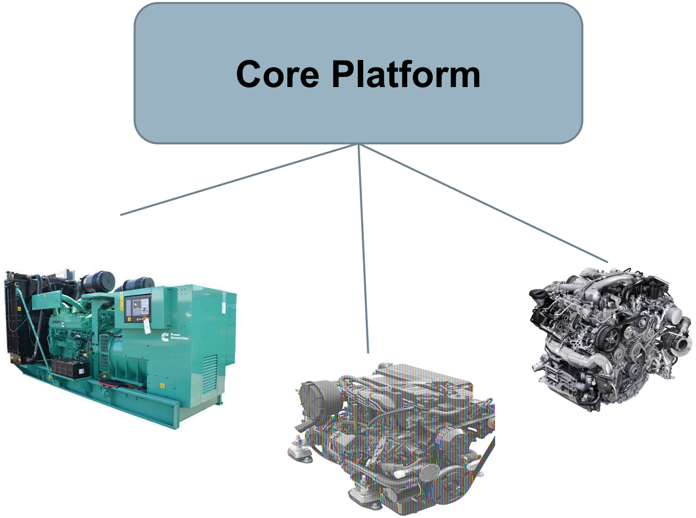
UNIVERSITY OF GOTHENBURG

Lecture 2: Domain Engineering

Gregory Gay
TDA 594/DIT 593 - November 3, 2022

Software Product Lines (SPLs)

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



Today's Goals

- Introduce Domain Engineering
  - Domain and Application Engineering
    - Platform vs Specific Product
    - Design FOR and WITH reuse
  - Principles of SPLE
    - BAPO: Business, Architecture, Process, Organization
  - Domain Modelling

Domain and Application Engineering

Core Development Activities

Requirements:
- Planned Behavior and Constraints

Design:
- System Architecture
- Classes
- Packages
- Communication Between Components
- Interfaces

Realization:
- Software Development

Testing:
- Observe the System to Ensure Requirements Met

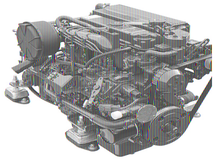
Domain and Application Engineering

Requirements
Design
Realization
Testing

Core Platform

Domain Engineering

Requirements
Design
Realization
Testing



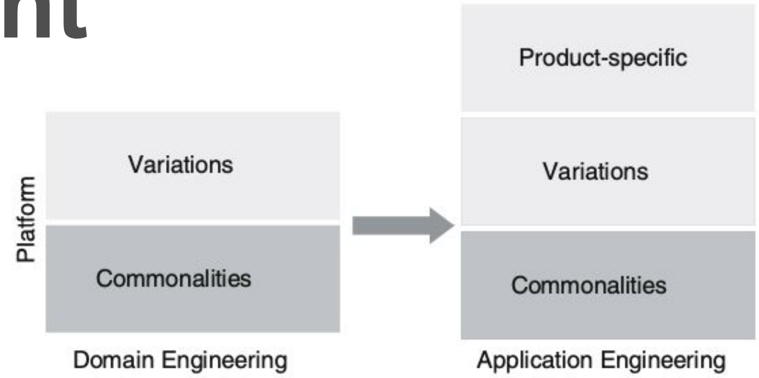
Requirements
Design
Realization
Testing

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.

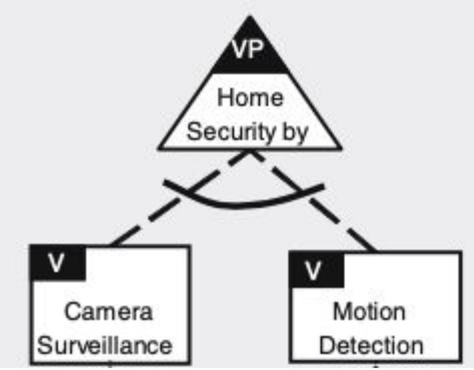
Variability Management



- Commonality
  - Shared between all products.
  - Implemented in core platform.
- Variations
  - 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

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.**

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 among features at *any variation point*.
  - Choosing one feature requires choosing or excluding another feature.

Fate of New Requirements

- 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

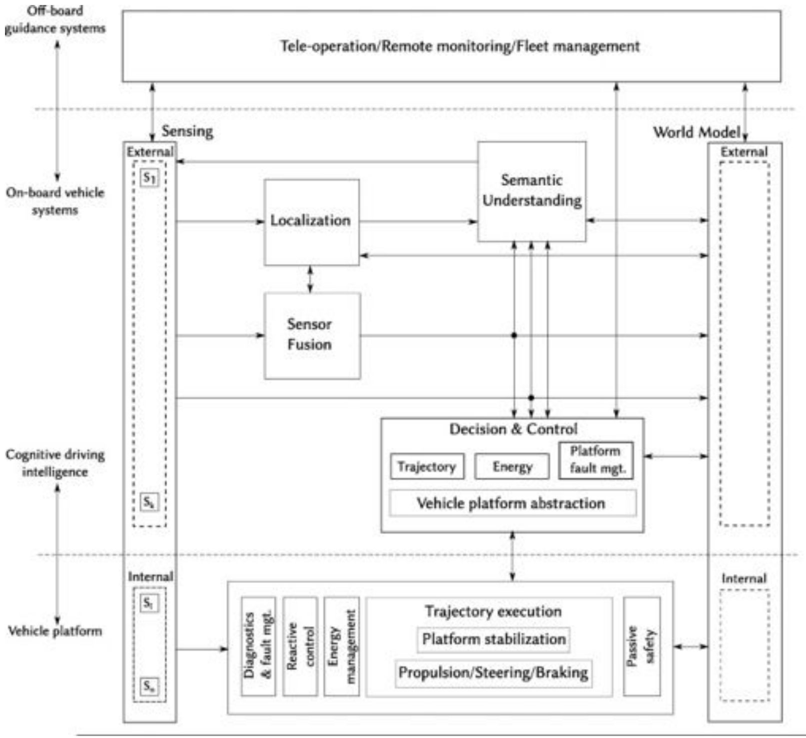
- Up-front planning and investment required.
- Long-term return on investment?
  - Implement requirement as part of platform or in a product?
  - 3+ concrete products: make it part of platform.

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?

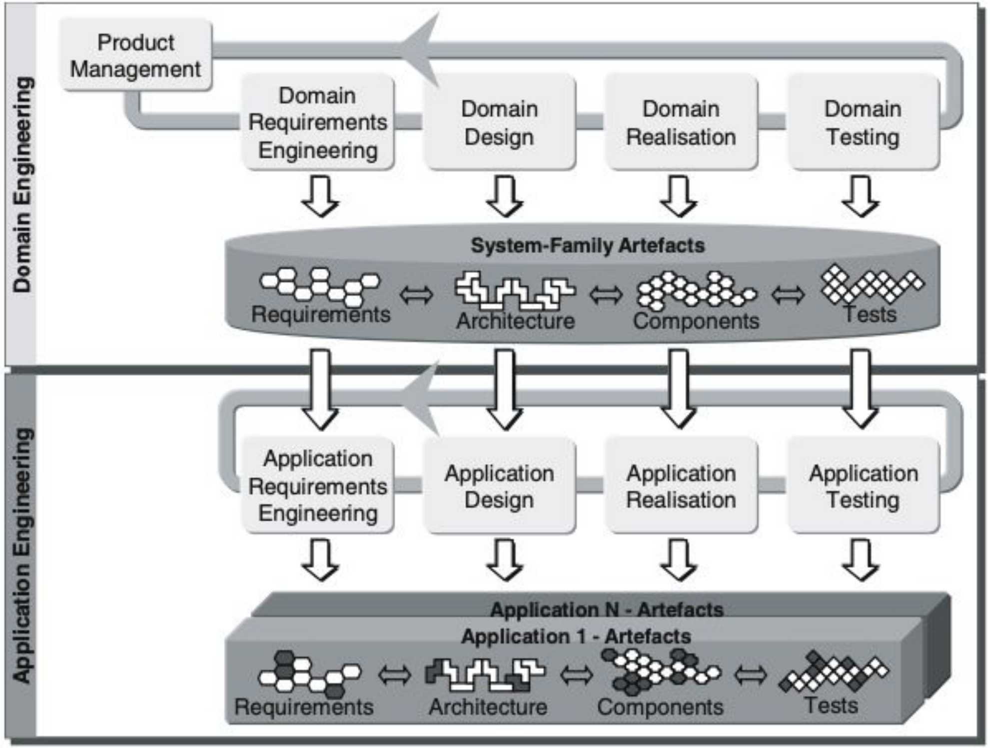
Architecture-Centric Development

- 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.



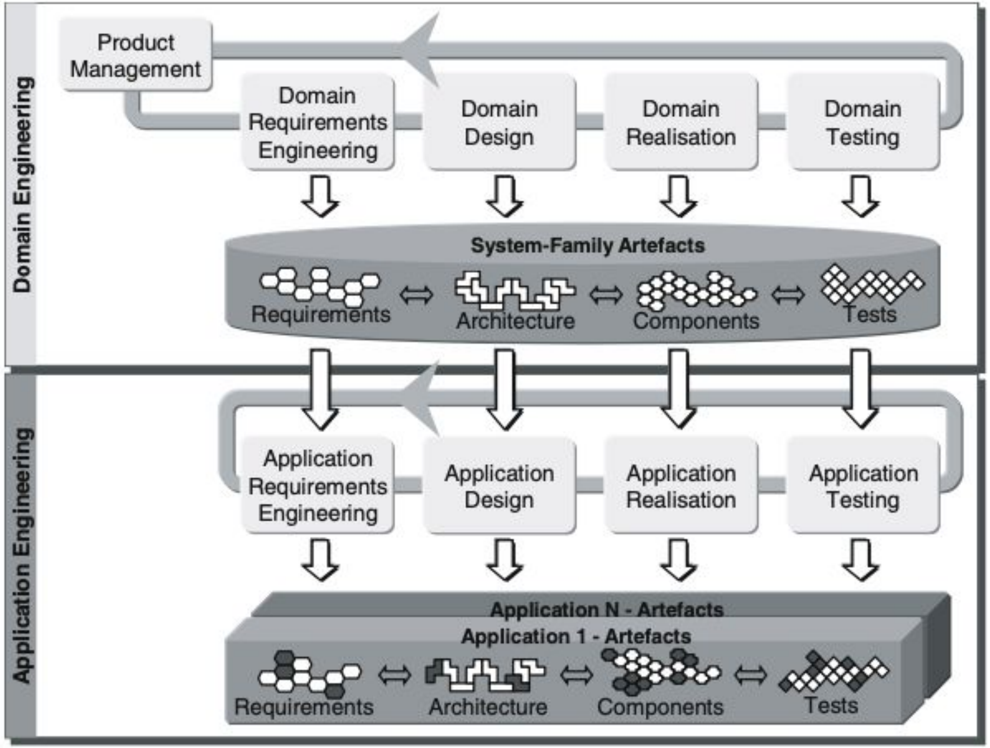
Domain and Application Engineering

- **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, supervised learning, …
  - Social network subdomains: message board, text chat, voice chat, video streaming

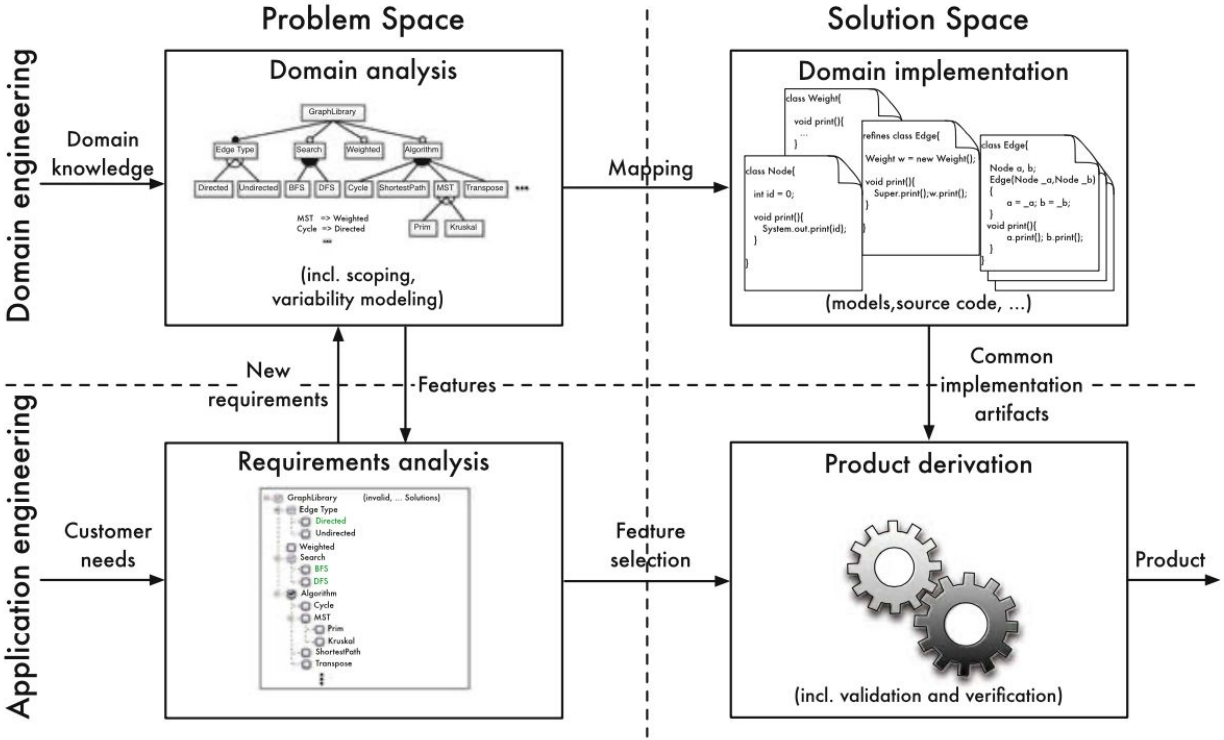
Problem and Solution Space

Problem Space

- Stakeholder view
- Characterized by features

Solution Space

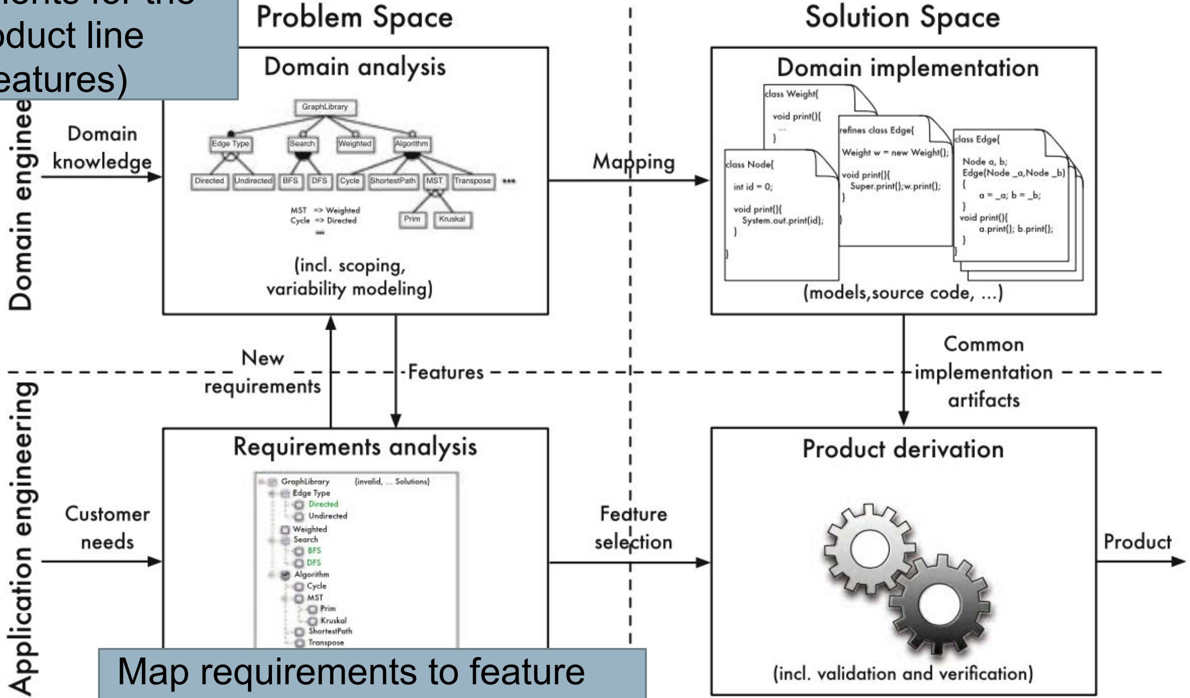
- Developer view
- Characterized by code structure
- Implementation of features.



Key Task Clusters

Develop reusable assets.

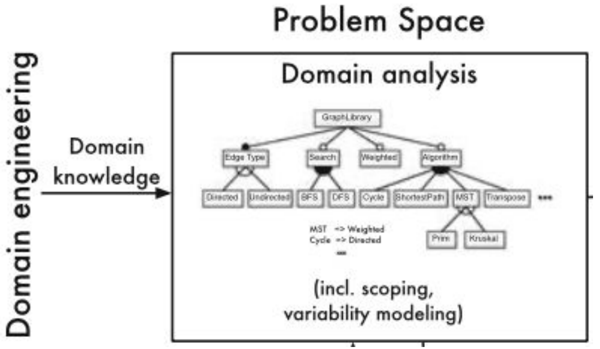
Requirements for the entire product line (scope, features)



Assets combined to form new concrete product.

Map requirements to feature selection, assess new requirements

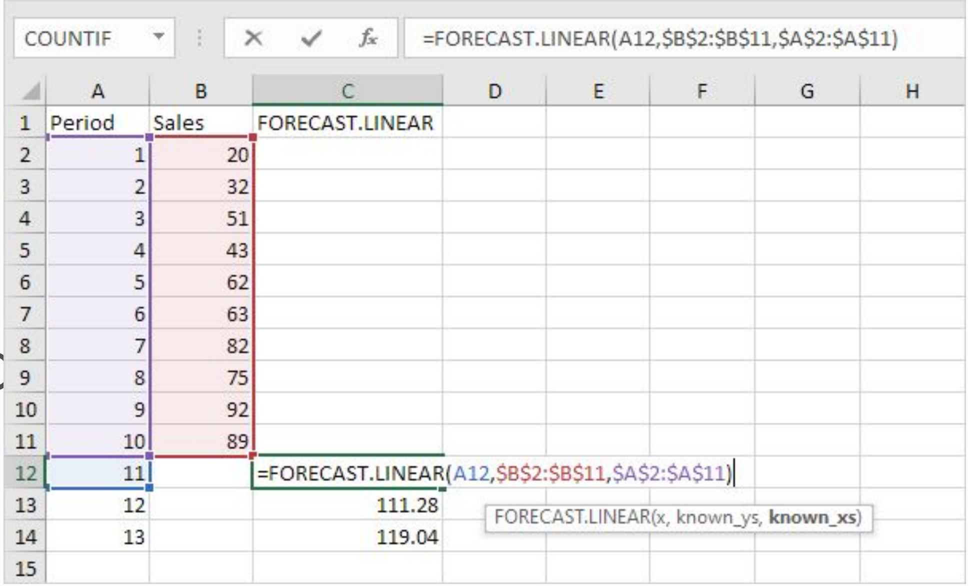
Domain Analysis



- 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

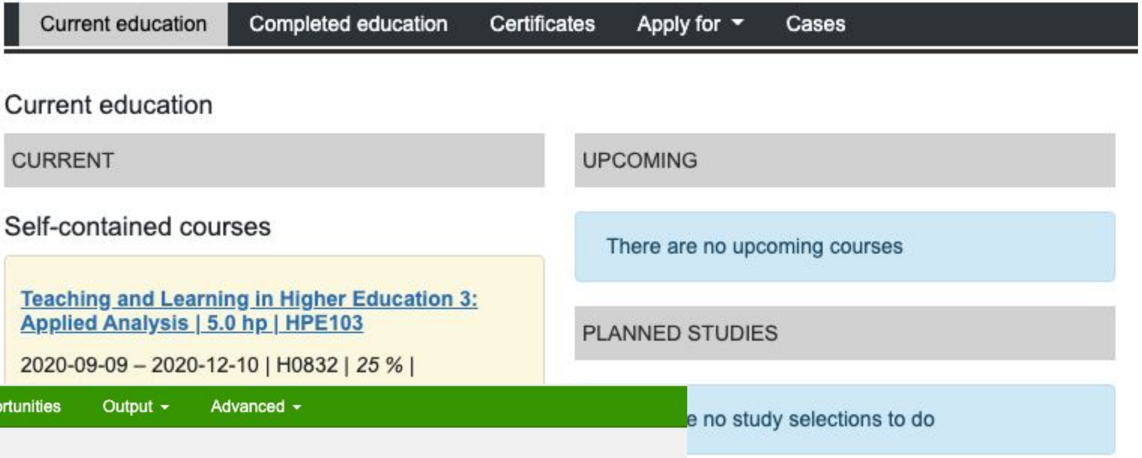
Example: Spreadsheets

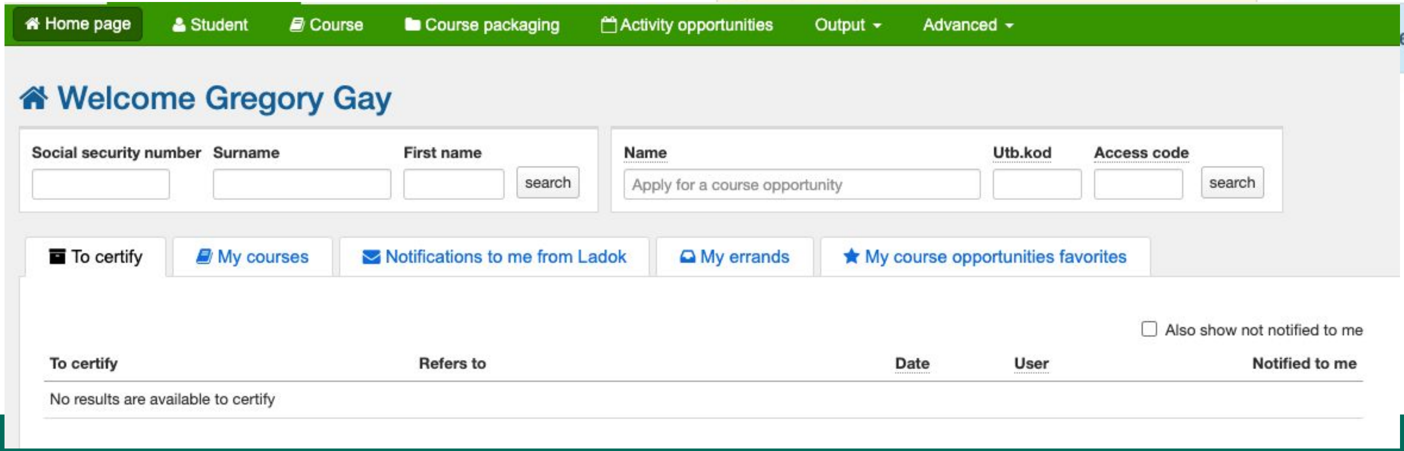
- Look at existing products: Excel, Google Sheets, …
- What are some features a user would expect?



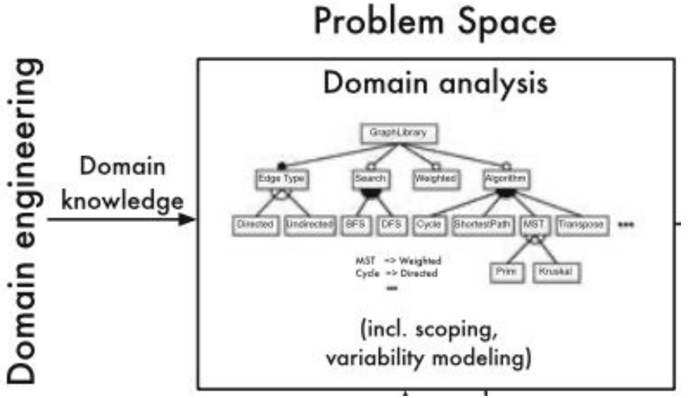
Example: Student Data Management (Ladok)

- Product Line: Student App, Teacher App





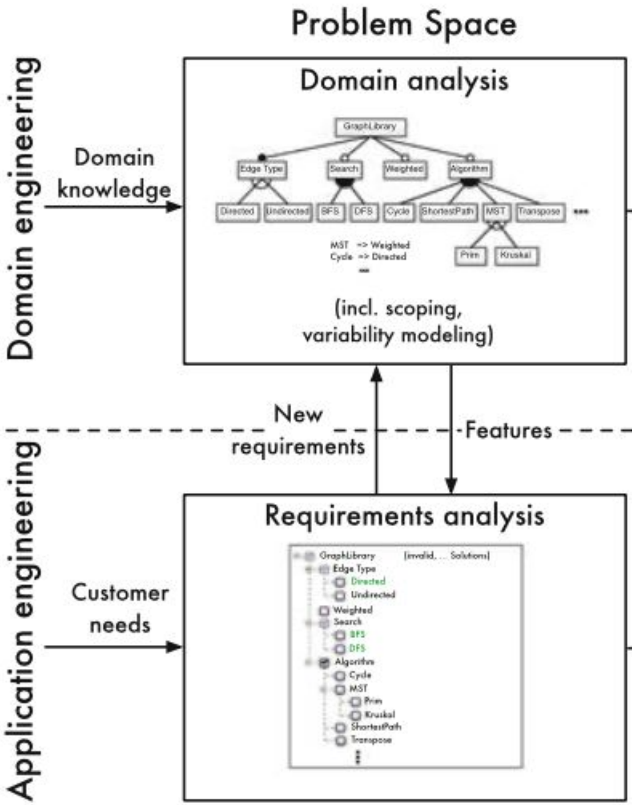
Domain Analysis



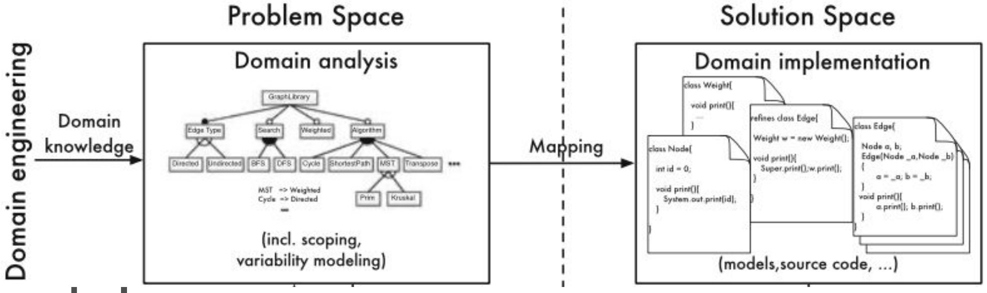
- Domain Modeling
  - Document commonalities and differences between products in terms of features and dependencies.
  - Ex: Embedded Database
    - Features: Storage, Transactions, OS (Android, Linux), Encryption
    - Storage, OS are mandatory.
    - Only one OS selection supported per product.

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.



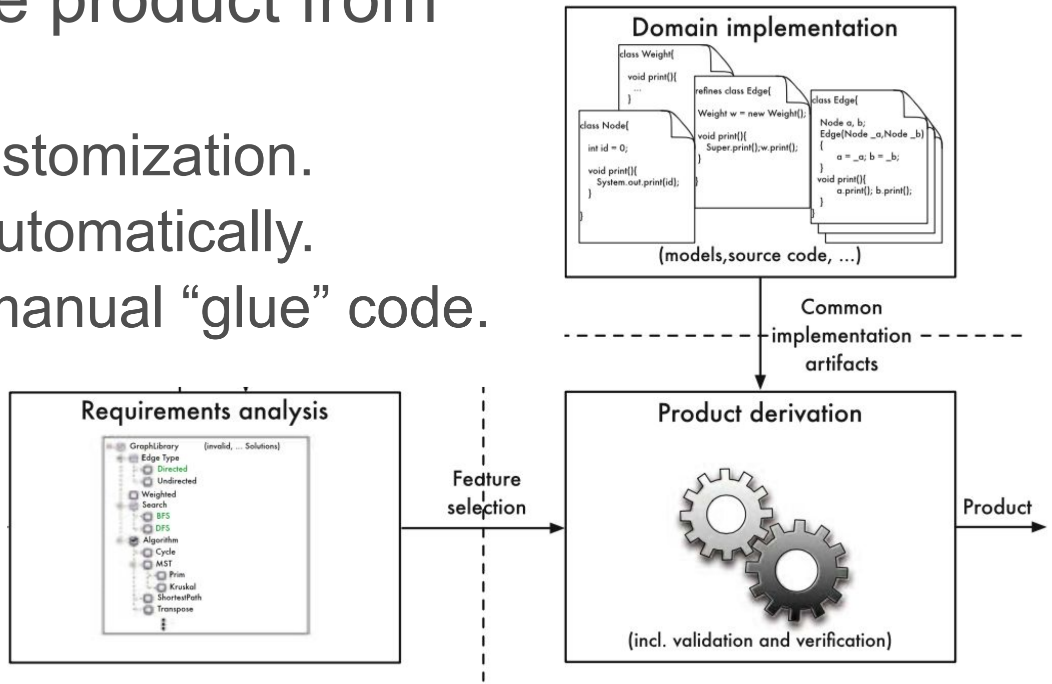
Domain Implementation



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

Product Derivation

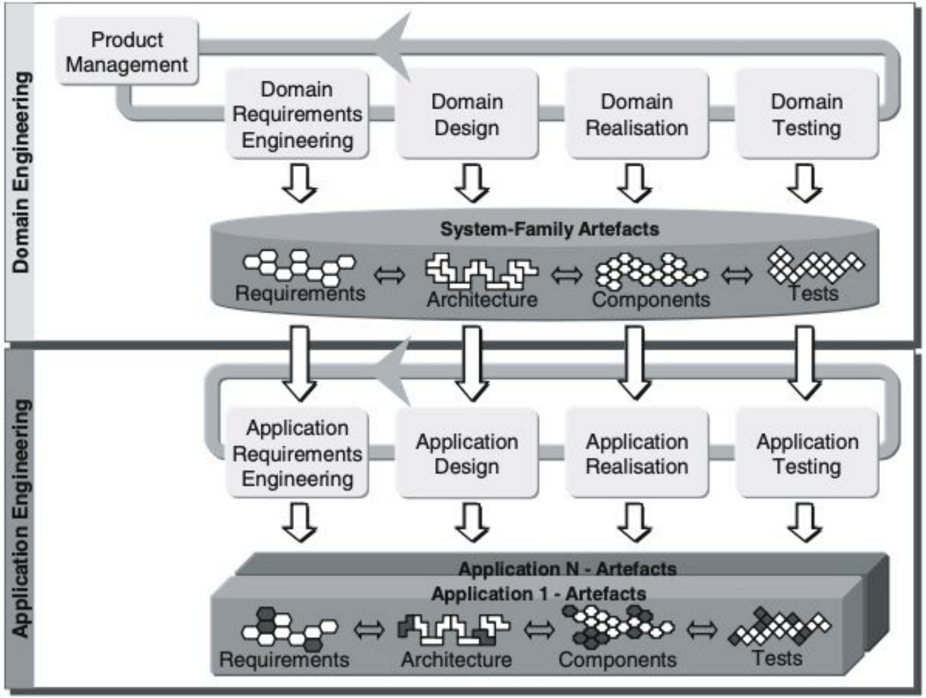
- Build the final concrete product from reusable assets.
  - Add any necessary customization.
  - Ideally, can be done automatically.
  - Often requires some manual “glue” code.



Let’s take a break!

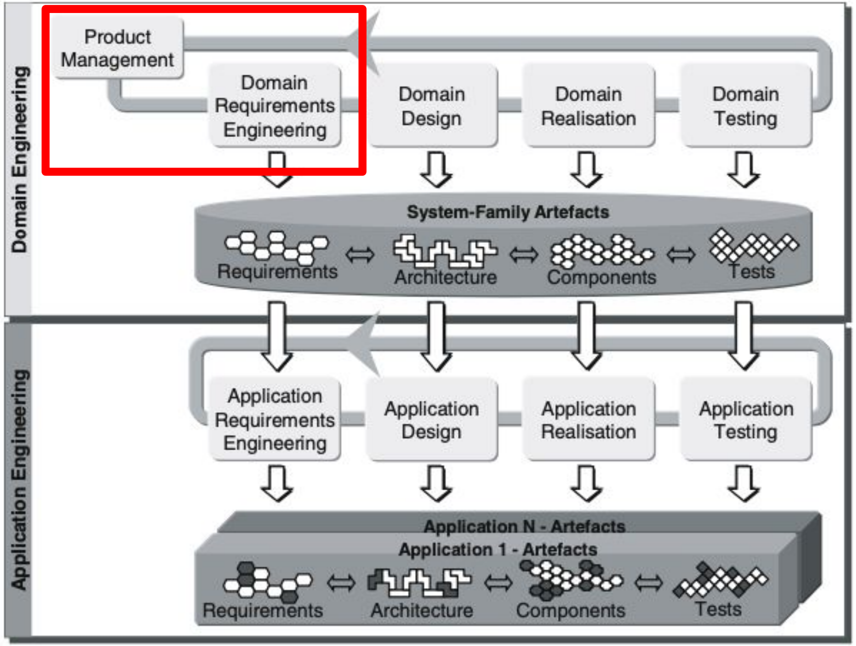
Two-Life-Cycle Approach

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



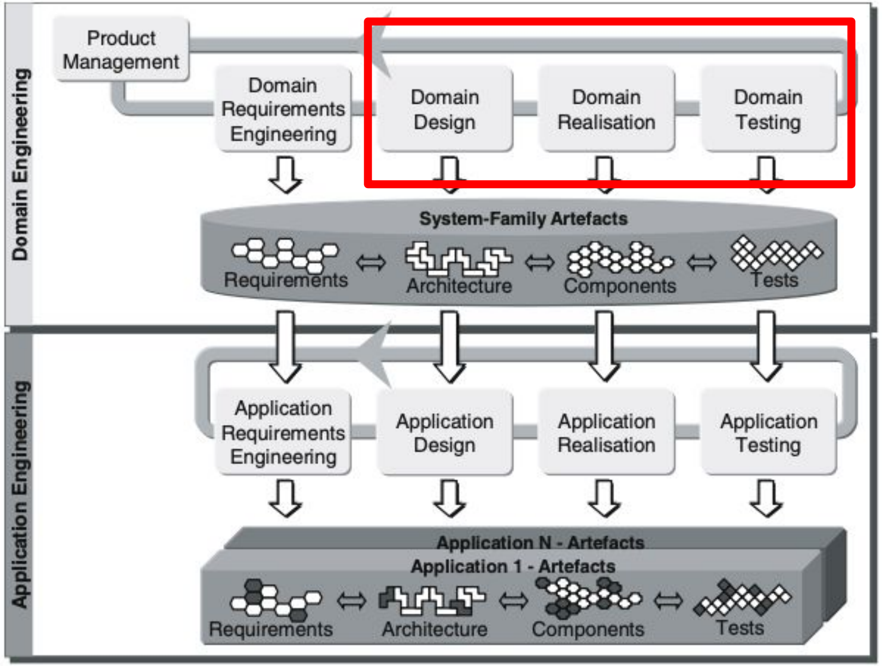
Domain Engineering Activities

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



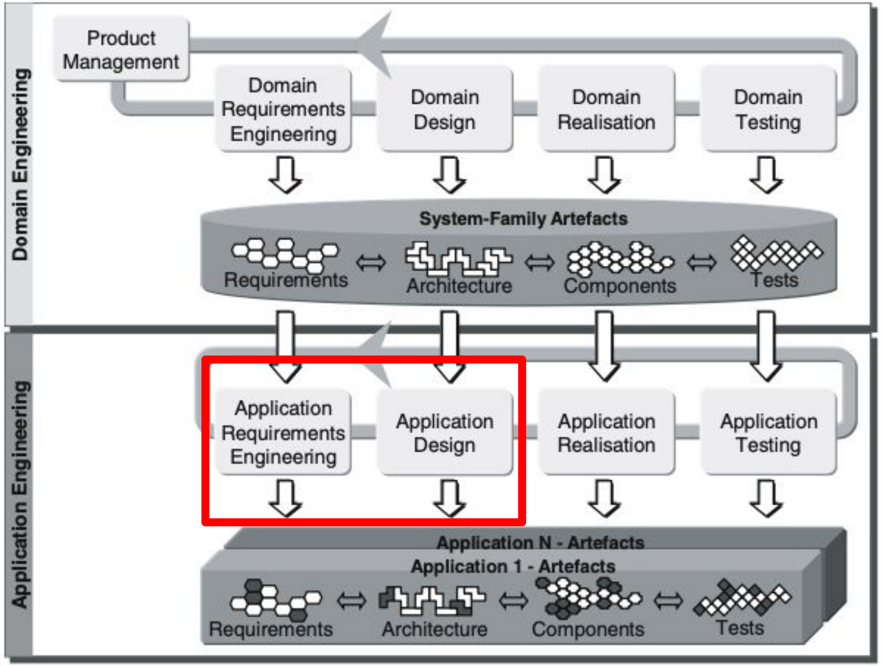
Domain Engineering Activities

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



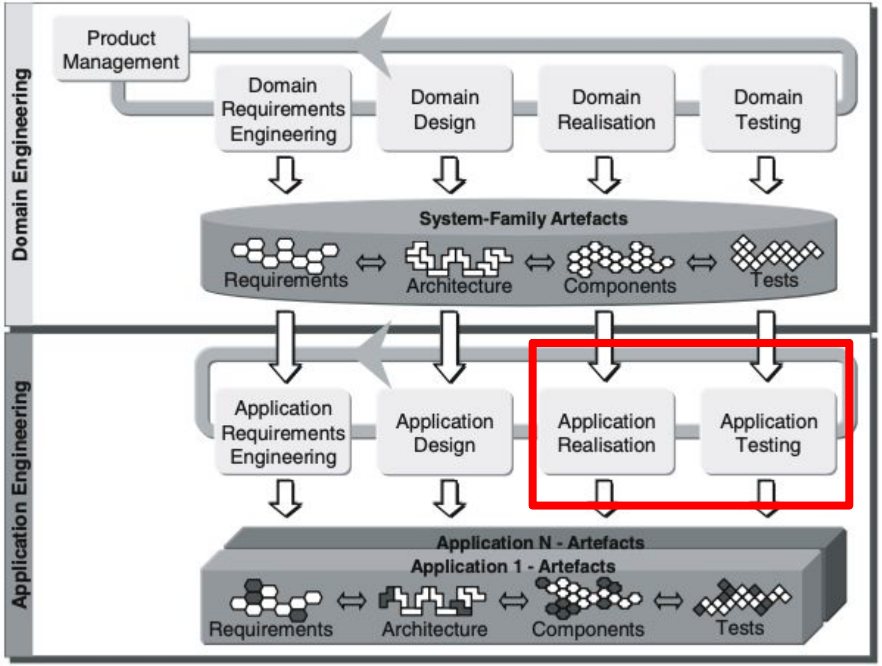
Application Engineering Activities

- 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.



Feature Diagrams

Features and Feature Dependencies

- Generally a functionality of the software.
- Can be **mandatory** or **optional**.
- Features are connected by their **relationships**.
  - Selecting A *allows* B to be selected.
  - Selecting A *requires* B to be selected.
    - **Variation Point:** Selecting A requires selecting one of (B, C, D).
- A feature model describes these relationships.

Identifying Features

- Aspects of the domain reflected in the software.
  - Externally-visible functions of software.
  - Aspects of non-functional behavior that can be controlled.
    - (energy consumption) “Precision” vs “Battery-Preserving”
    - (disk usage, memory) How often data is saved
- Must represent a **distinct** and **well-understood** aspect of the system.

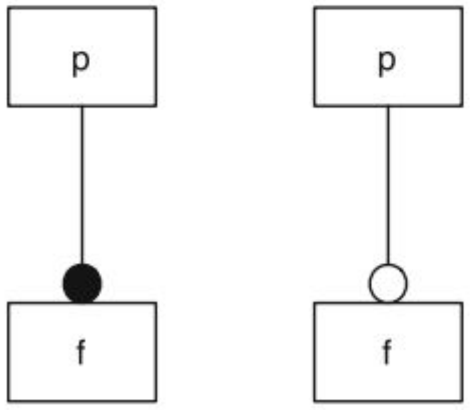
Understanding a Feature

- To model a feature, consider:
  - Description and requirements
  - Relationship to other features
    - (hierarchy, ordering, grouping)
  - External dependencies (hardware, software)
  - Configuration knowledge (activated by default?)
  - Constraints (requires feature X, excludes Y)
  - Effect on non-functional properties
  - Attributes (number, parameters)
  - Potential feature interactions.

Feature Diagrams

Mandatory Feature

Optional Feature

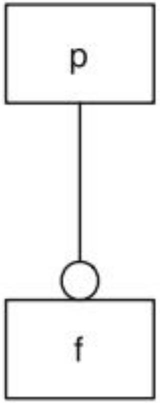


- Tree where nodes represent features.
- Shows parent-child relationship.
  - F can only be selected when P is selected.
  - Parent tends to be more general, child is more specific.
    - Parent - Sensor, Child - RADAR

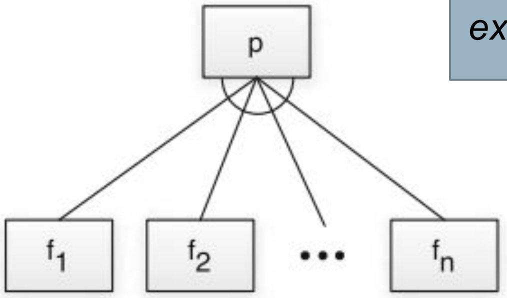
Feature Diagrams

Mandatory Feature

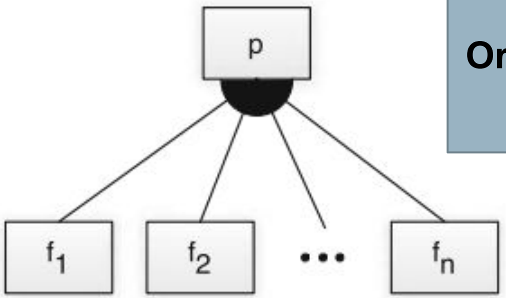
Optional Feature



Alternative (mutually exclusive choice): Choose *exactly* one



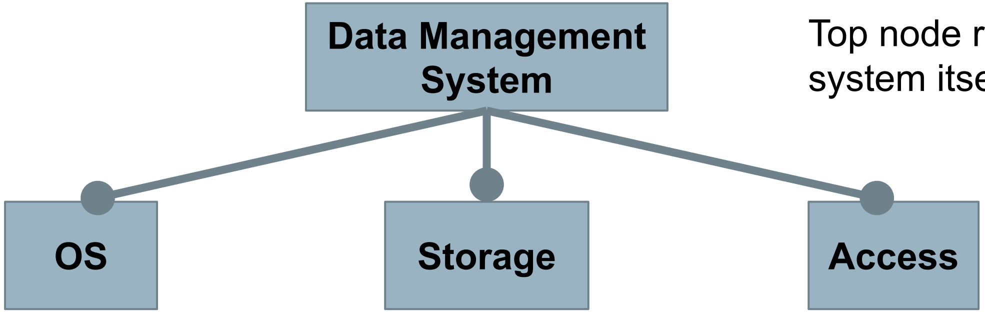
Or: Choose **at least** one



Cross-Tree Constraints

- **Cross-tree Constraints** are predicates imposing constraints between features.
  - `DataDictionary ⇒ String`
    - (Storing a data dictionary **requires** support for strings)
  - `MinimumSpanningTree ⇒ Undirected ∧ Weighted`
    - (Computing a Minimum Spanning Tree **requires** support for undirected **and** weighted edges)
  - Constraints over Boolean variables and subexpressions.
    - (i.e., (`NumProcesses >= 5`))

Example - Data Management

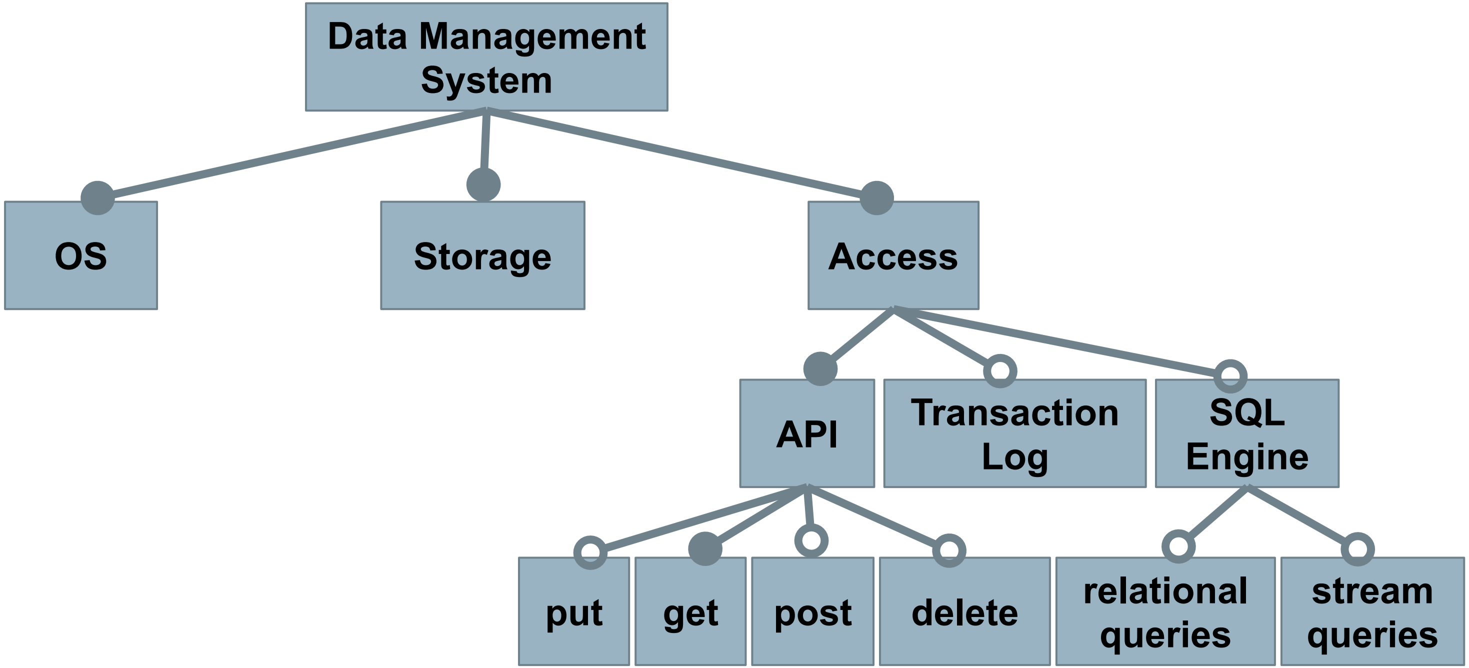


Top node represents the system itself.

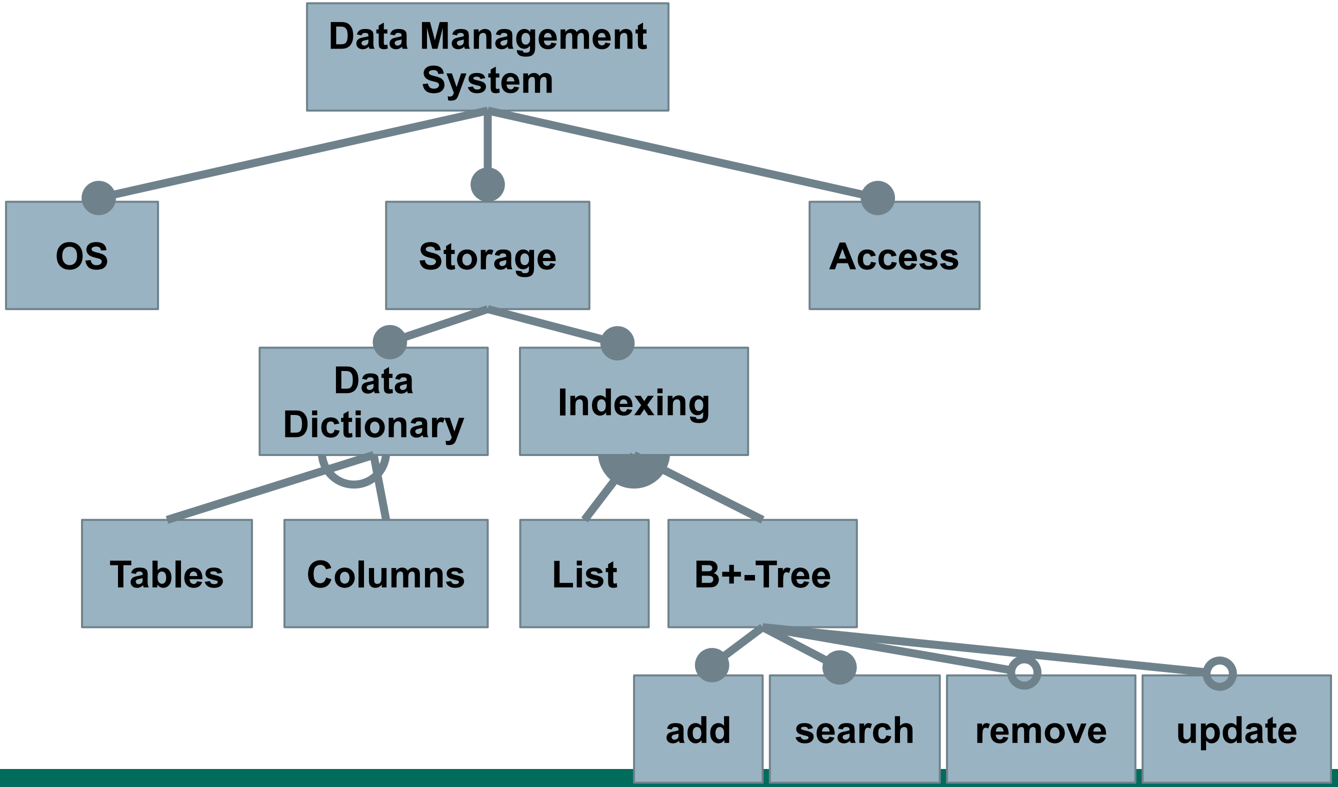
Hierarchy goes from general/abstract to specific.

First layer represents "types" of functionality.

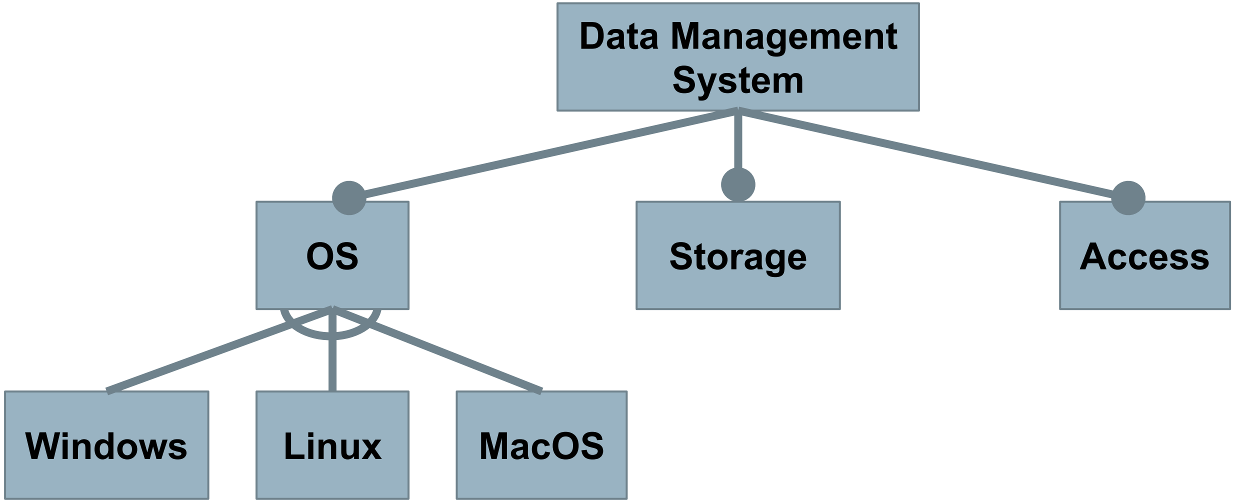
Example - Data Management



Example - Data Management

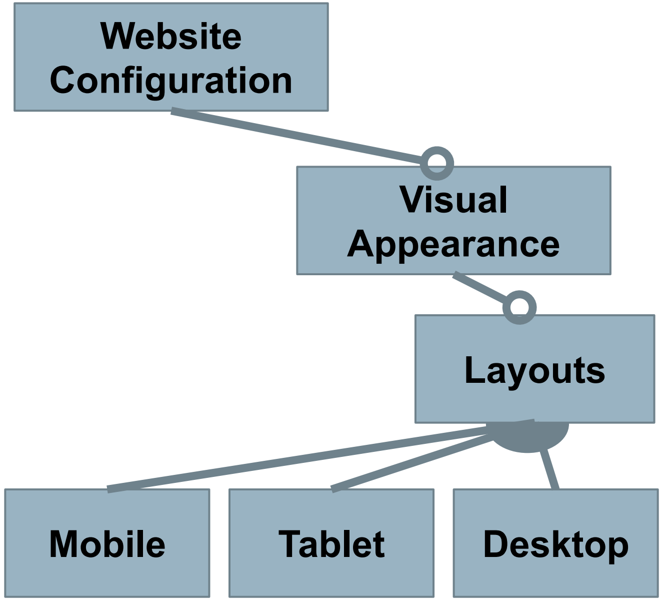


Example - Data Management

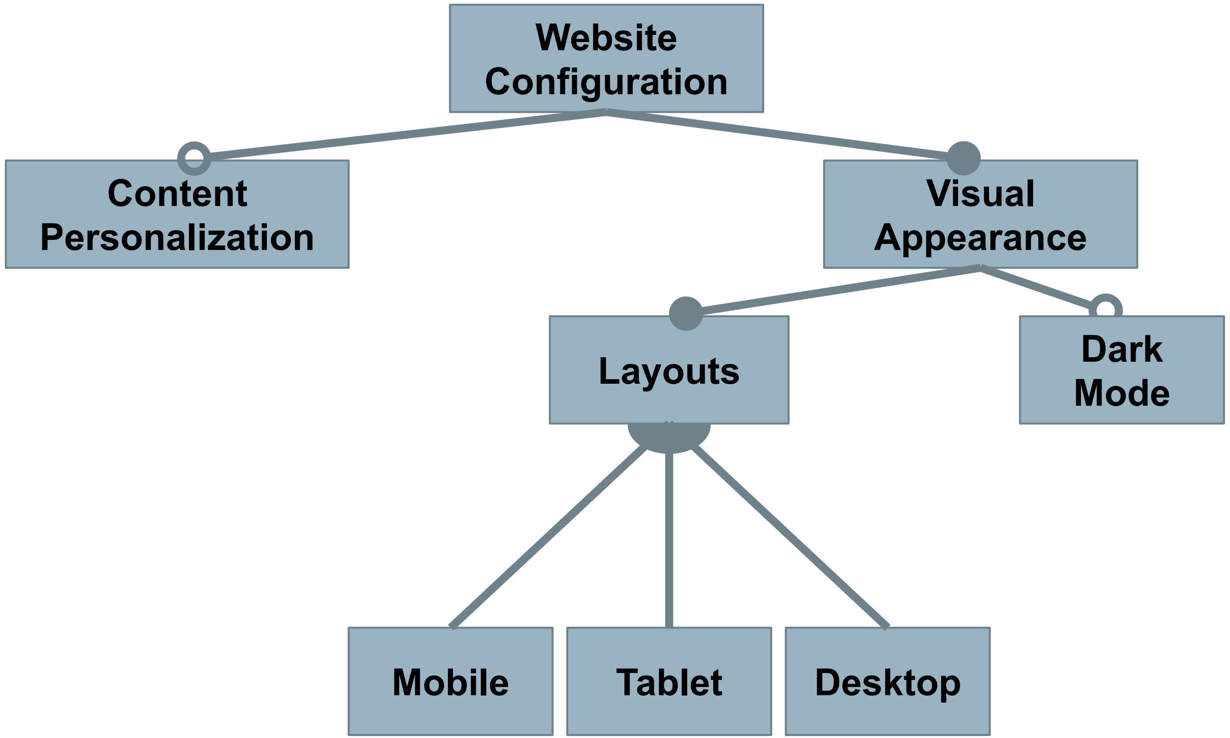


Example - Website Configuration

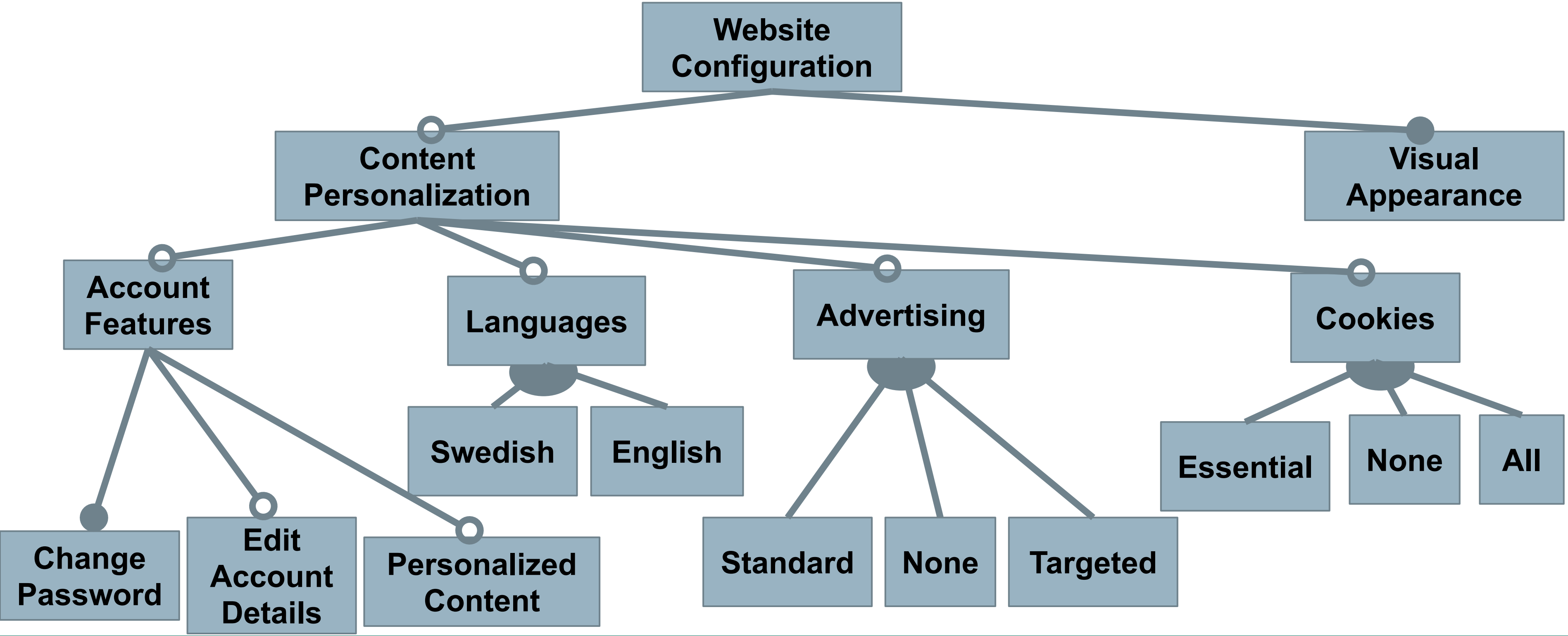
- SPL that provides website functionality.
- One feature - adjusts layout based on the device.
- What other aspect of the site could be features?
  - Consider visual appearance and personalized content.



Example - Website Configuration



Example - Website Configuration



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.

We Have Learned

- A product is a **valid** selection of features.
- Feature models capture the constraints that define whether a selection is valid.
  - Feature diagrams represent feature relationships visually.
  - Propositional logic represents feature relationships as formulae that can be used in analyses.

Next Time

- Feature Modelling and Analysis
- 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 13, 11:59 PM**
- Case study examining development of a SPL or other reuse-driven system.
  - **Choose a system from case studies on Canvas**

Assignment 1 - Case Study

- 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