Variability Modeling, Management, and Implementation

KV Product Line Engineering (343.354)

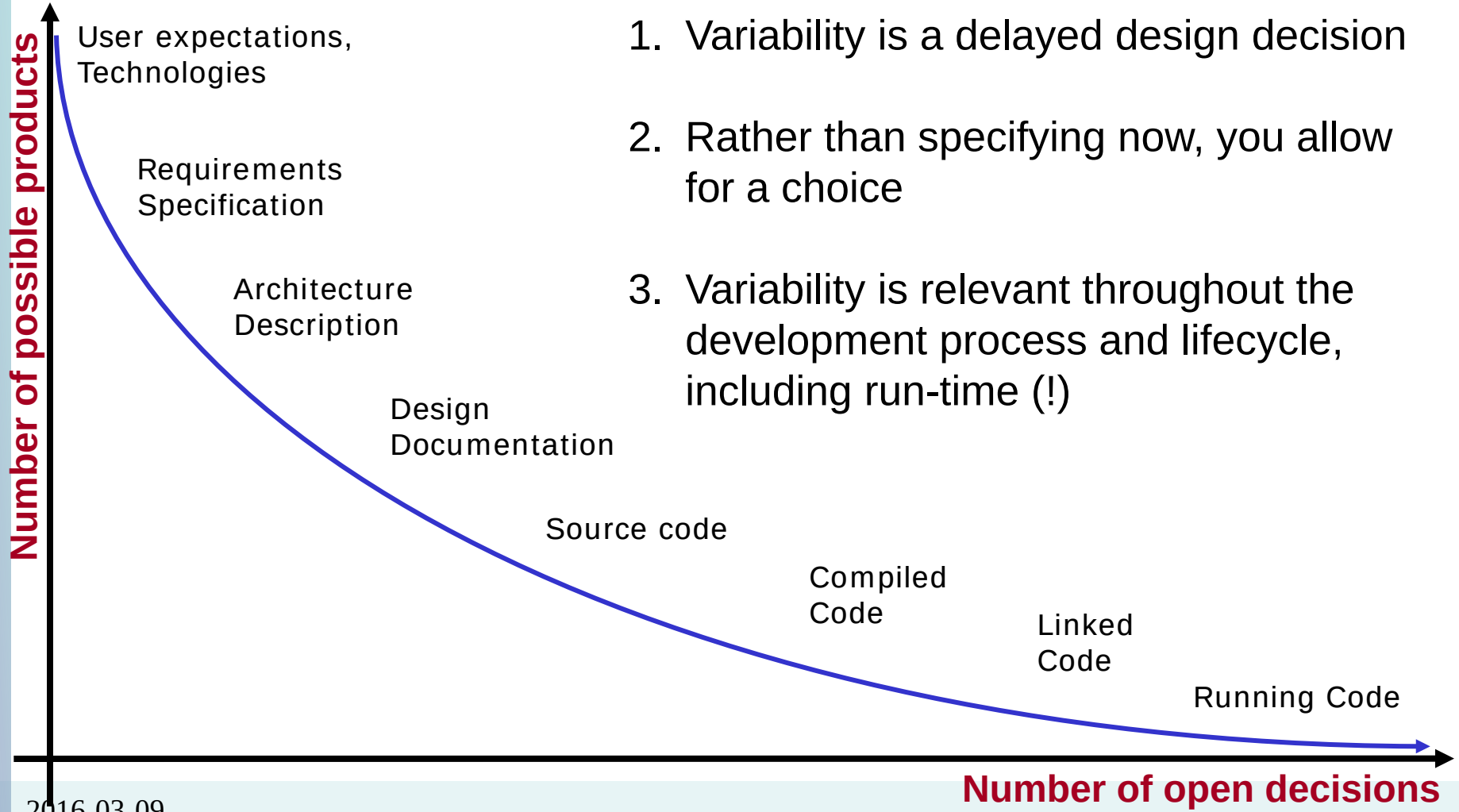
Dr. Roberto Lopez-Herrejon
Dr. Rick Rabiser

9.3.2016

Why do we need Variability?

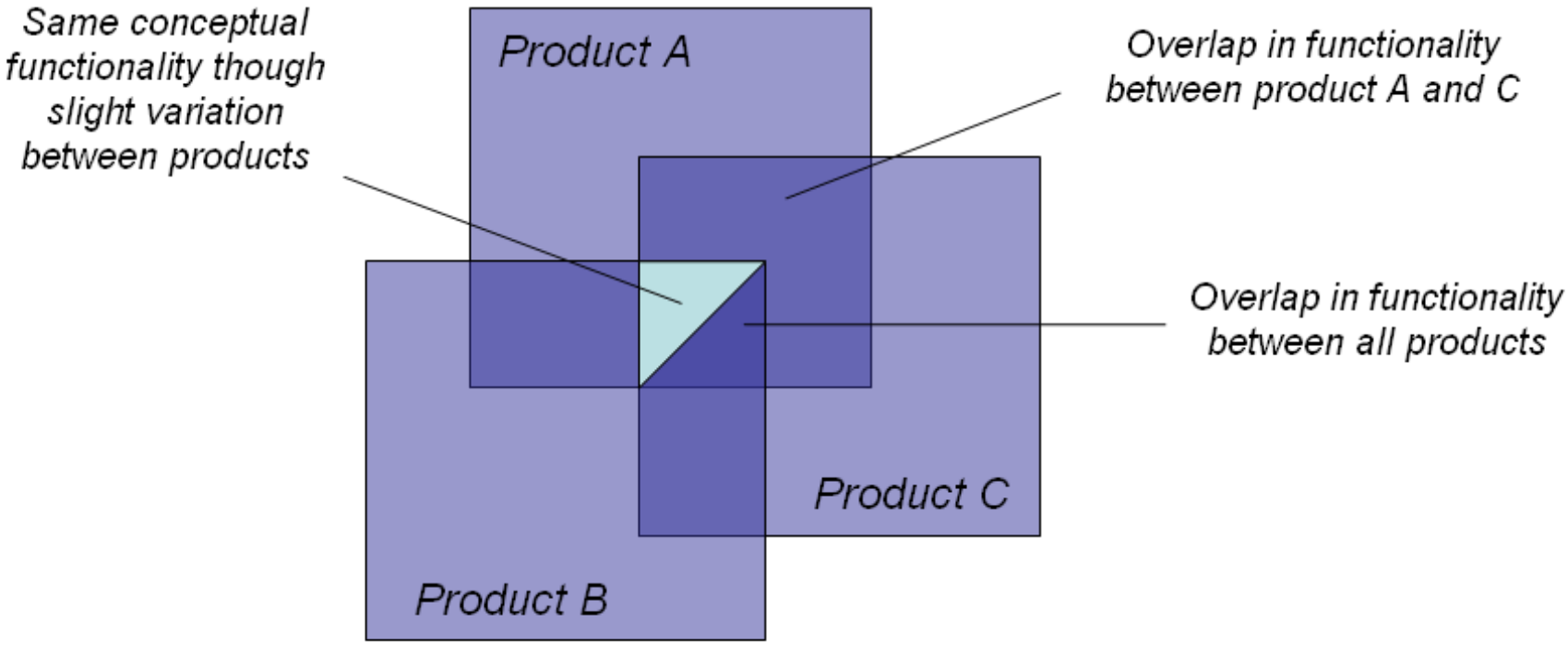
- Reuse = using an existing piece of software in a different context
  - Must be possible to adapt software to new contexts
  - Ad-hoc reuse does not work, the software needs to be prepared for reuse
- Software without variability is not reusable
- **Achieving reuse relies on understanding, documenting, and managing variability**

What is Variability?



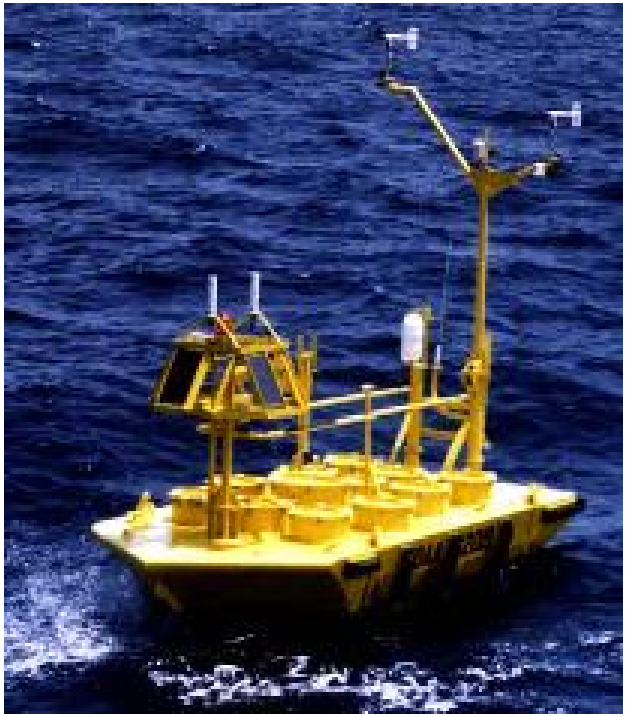
1. Variability is a delayed design decision
2. Rather than specifying now, you allow for a choice
3. Variability is relevant throughout the development process and lifecycle, including run-time (!)

Commonality and Variability Analysis



Domain Engineering: Product-Line Requirements

- Commonality Analysis:
  - Commonalities
    - “All FWS shall report the current temperature.”
  - Variability
    - “Some FWS may report the wind direction.”
  - Constraints
    - “Any FWS that reports the wind direction must also report the wind speed.”



Source: National Data Buoy Center

Types of Variability

- **External** Variability
  - Visible to the customer:
    - Example: manual vs. automatic transmission
    - Example: your cell phone may or may not have a camera and you may have different resolution options
- **Internal** Variability
  - Hidden from customer:
    - Example: battery technology in hybrid electric car
    - Example: communication protocol

Implementing Variability

- Diverse Techniques
  - Parameterization
  - Configuration Constants
  - Meta-tool + Code generation
  - OO Principles/Object-Oriented Frameworks
  - Dependency Injection
  - Aspects

Technique 1: Parameterization

- Change values of attributes
- Choose from a list of predefined options
- Example:

```
void foo(int i) {
    if (i > 0) {
        // do something
    } else {
        // do something else
    }
}
```

- Limitation: No new functionality without changing the component

Technique 2: Configuration Constants

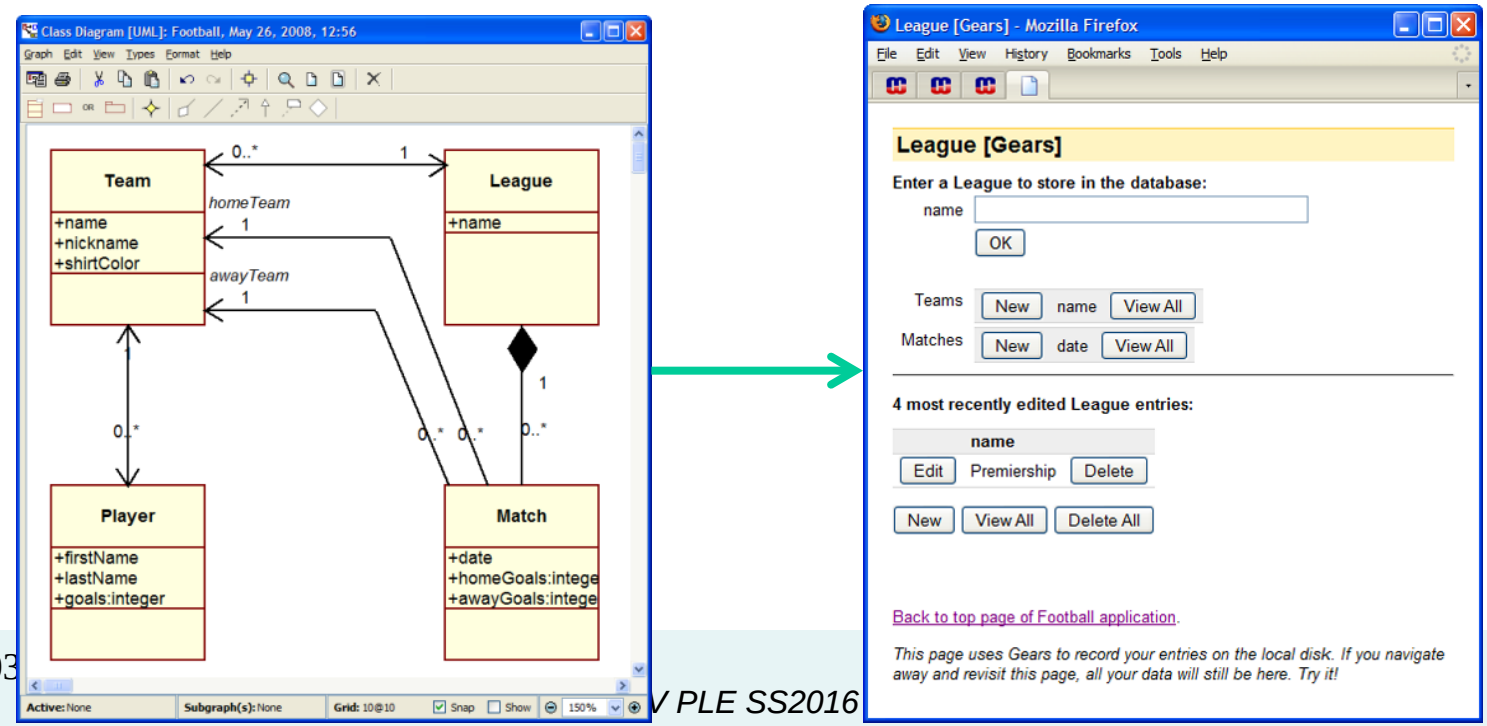
JOHANNES KEPLER UNIVERSITY LINZ

- Symbolic constants for constant values as a common practice
  - #define MaxSpeed 100;
- Conditional compilation
  - Source common to all variants

```
# ifdef configParam1
      …
# endif
# ifdef configParam2
      …
# endif
```

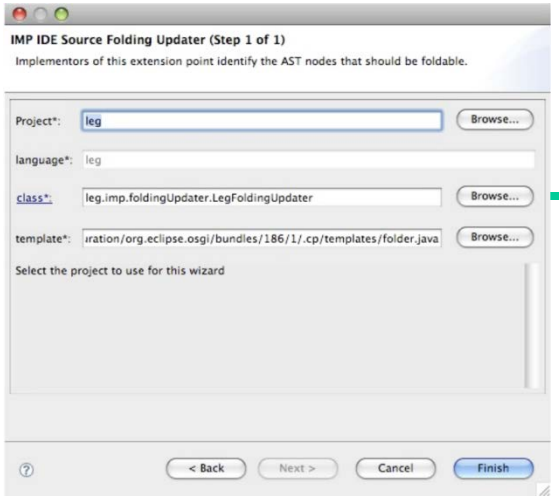
Technique 3: Meta-tools + Code Generation

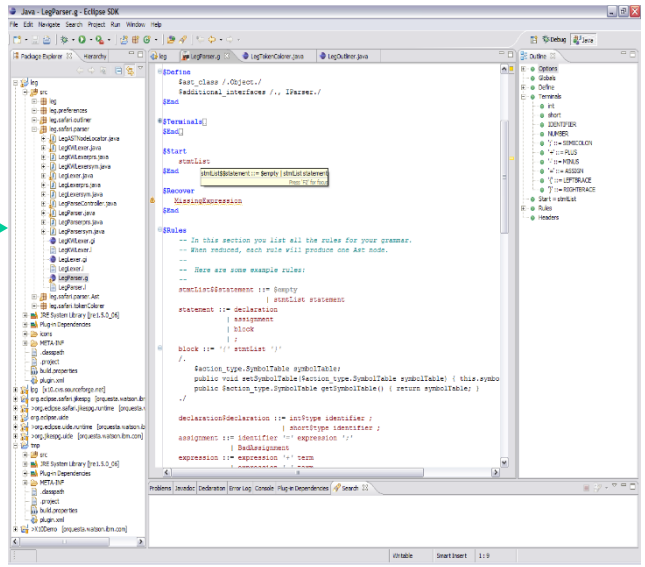
- DSL Creator: MetaEdit+
  - Graphical notation (concepts, rules, generators)
  - Automatic creation of editor and views
  - http://www.metacase.com/cases/dsm_examples.html



Technique 3: Meta-tools + Code Generation

- Eclipse IDE Creator
  - Textual notation (grammar, syntax highlighting, code completion, …)
  - Automatic creation of compiler, editor, debugger, etc.
  - http://eclipse-imp.sourceforge.net/

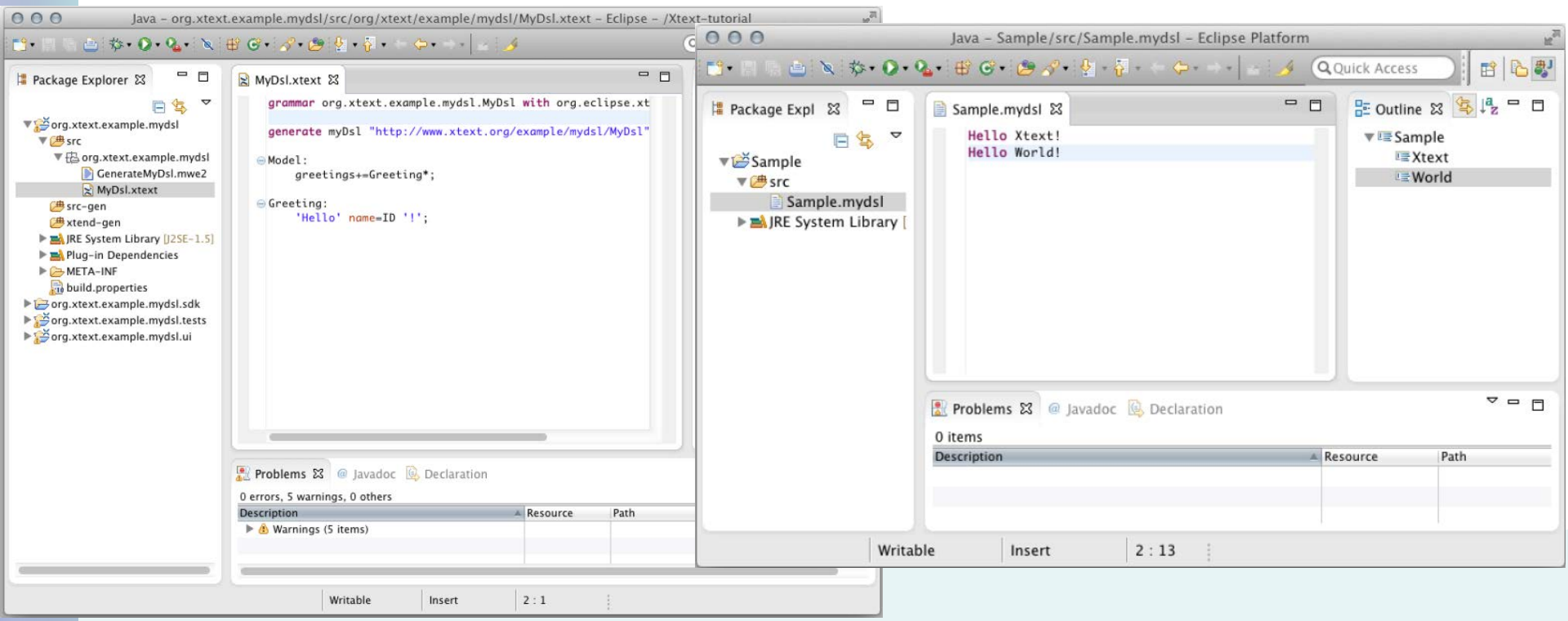




Technique 3: Meta-tools + Code Generation

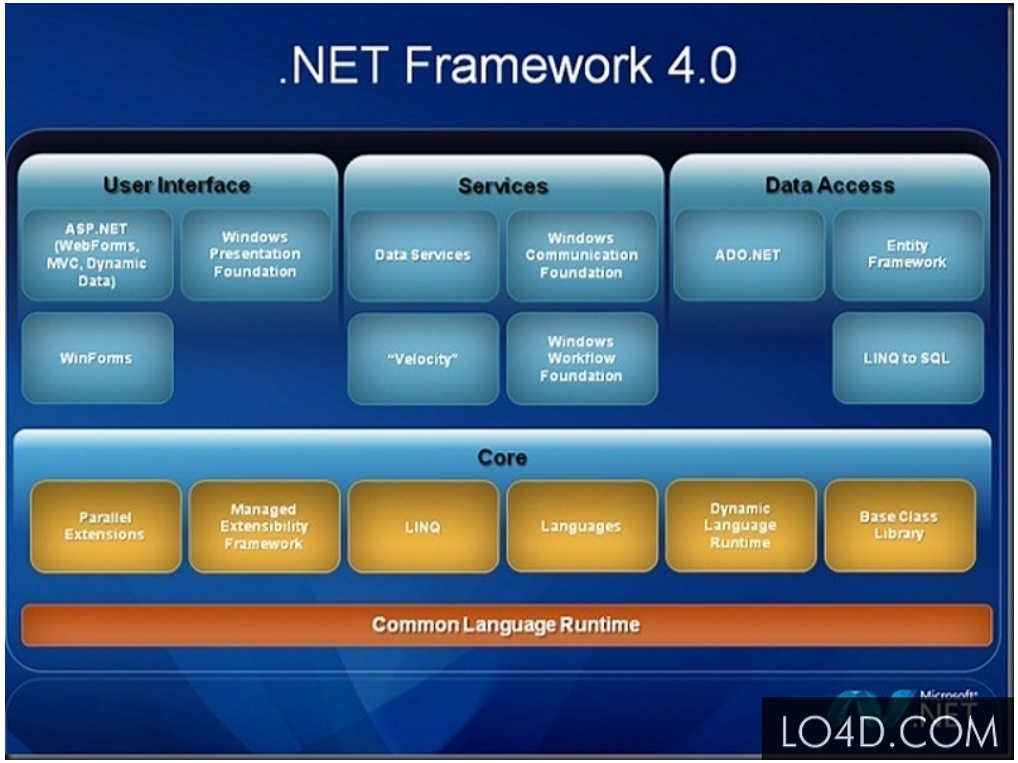
JOHANNES KEPLER UNIVERSITY LINZ

- Eclipse XText/XTend
  - Textual notation (grammar, syntax highlighting, code completion, …)
  - Automatic creation of compiler, editor, debugger, etc.
  - https://eclipse.org/Xtext/



Technique 4: Object-Oriented Frameworks

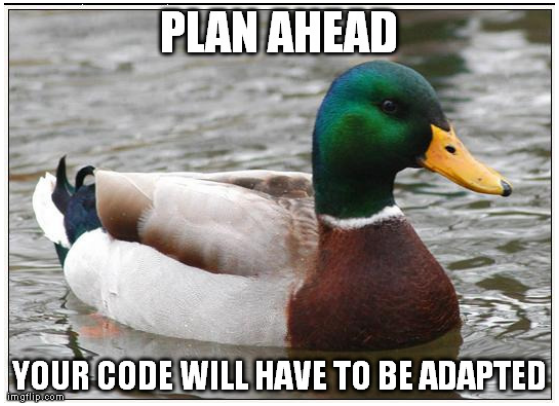
- Reusable design of a system or subsystem
- Implemented through a set of classes and their collaborations
- Users complete or extend it by adding or customizing application-specific components
- E.g., Microsoft .NET

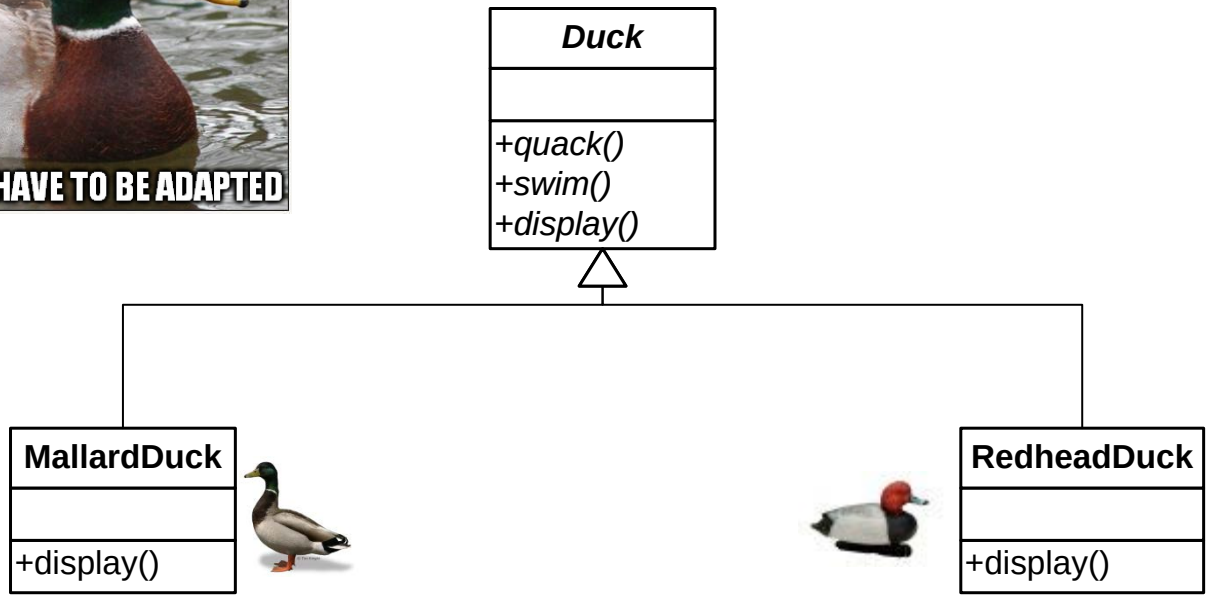


Technique 5: OO Principles

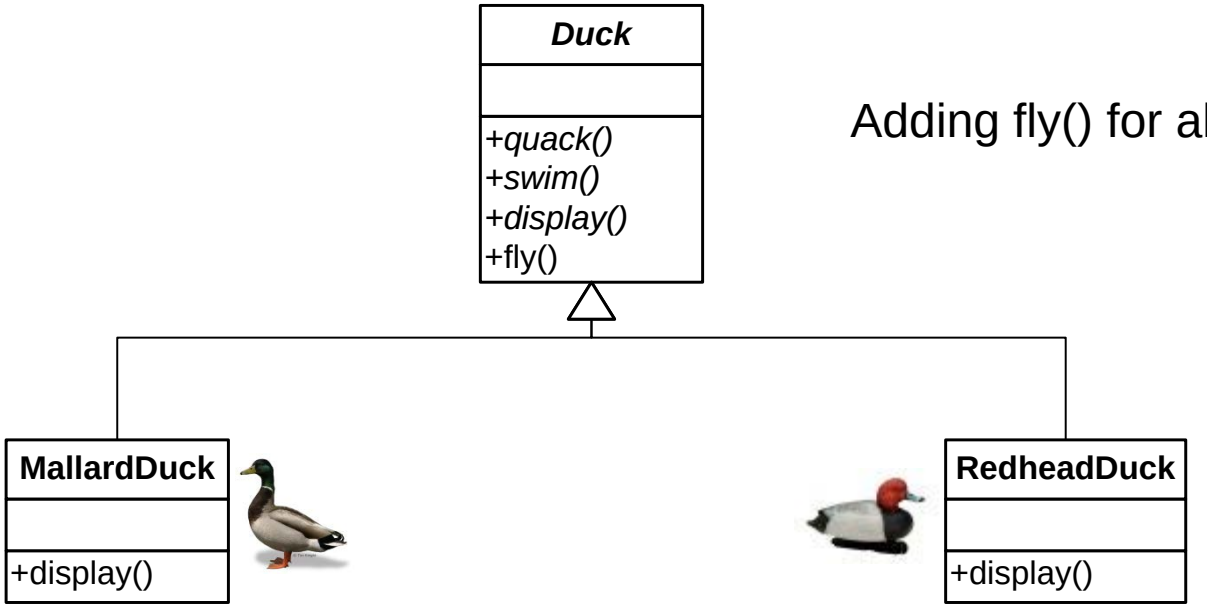
- Information hiding
  - Implementation details are hidden for clients
- Polymorphism (dynamic / late binding)
  - Runtime system determines the dynamic type of the object
  - Behavior can change at runtime
- Separation of concerns
  - Decompose system into subsystems, classes and class hierarchies
- Design patterns
  - Incorporate widely known, proven design elements

Inheritance: Duck Example



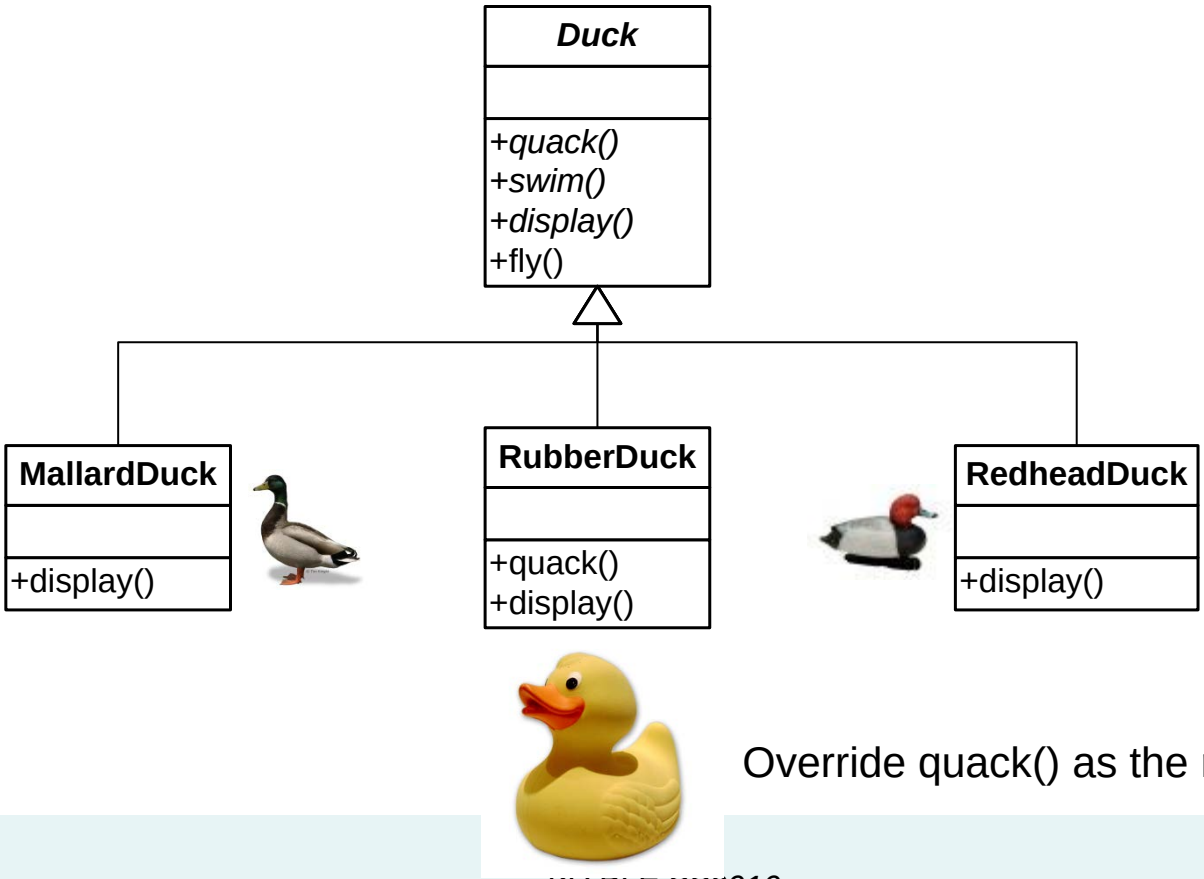


Inheritance: Duck Example



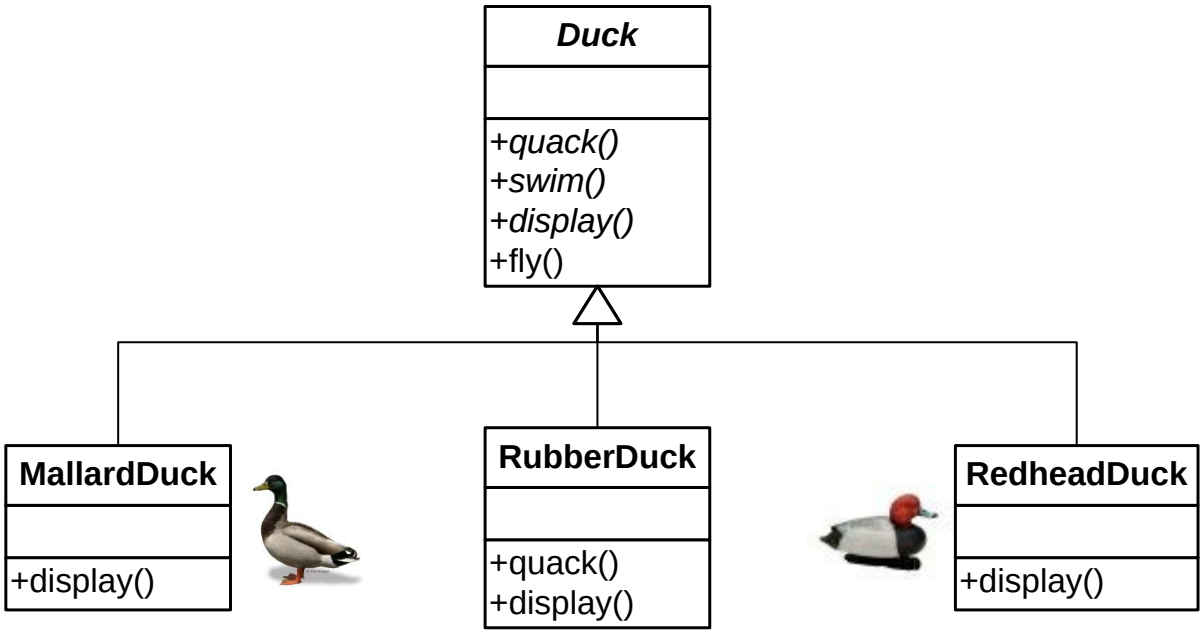
Adding fly() for all ducks

Inheritance: Duck Example



Override quack() as the rubber duck squeaks!

Inheritance: Duck Example



fly()?
Override?
A hole in hierarchy?

Technique 5: OO Principles

Inheritance is not THE answer…

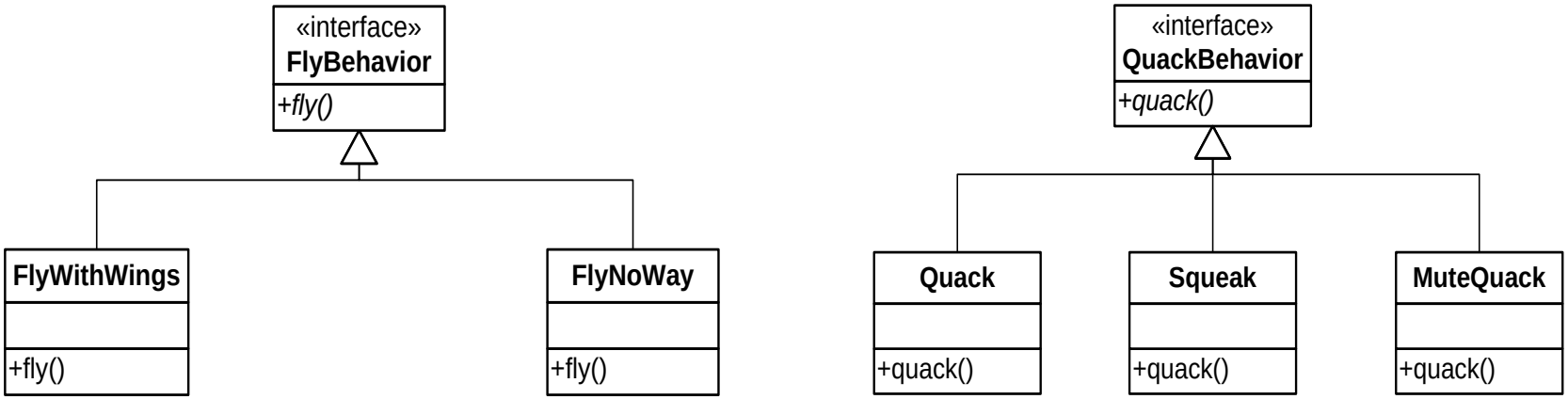
- Static specification
- Code must be changed

Encapsulate what varies and separate it from the rest of the code!

- Reduces coupling among classes
- Reduces chances of duplicating code later on

Commonality / Variability Analysis

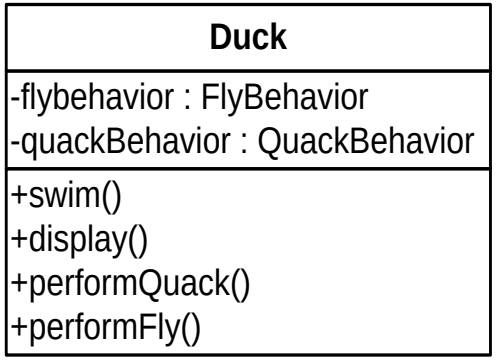
- Encapsulate Flying and Quacking in own classes with own hierarchies
- Duck implements the usage of interfaces



Technique 6: Inversion of Control

JOHANNES KEPLER UNIVERSITY LINZ

- Duck holds interface references




```
FlyBehavior dontFly = ...
QuackBehavior squeakQuack =
...

Duck rubberDuck = new Duck();
rubberDuck.setFlyBehavior(..);
rubberDuck.setQuackBehavior(..
   );
rubberDuck.performFly();
```

Technique 6: Spring Framework for IoC

JOHANNES KEPLER UNIVERSITY LINZ

- Java application framework – http://www.springsource.org

- Implements lookup
- Allows plugging and hot swapping
- Promotes good OO design
- Enables reuse
- Makes applications easy testable

Technique 6: Spring Framework for IoC

JOHANNES KEPLER UNIVERSITY LINZ

- Dependency injection
  - Defined through constructor arguments or properties
  - Injection at runtime
- Hollywood principle – “don’t call me, I will call you”
- Decouples object creators and locators from application logic

Technique 6: Spring Framework for IoC

JOHANNES KEPLER UNIVERSITY LINZ

Bean

- Unique ID
- Created as/from
  - Singleton – one instance for all references
  - Prototype – new instances created for each reference

- Created as late as possible

Technique 6: Spring Framework for IoC

```
FlyBehavior dontFly = ...
QuackBehavior squeakQuack = ...
Duck rubberDuck = new Duck();
rubberDuck.setFlyBehavior(dontFly);
rubberDuck.setQuackBehavior(squeakQuack);
rubberDuck.performFly();
```

```
<bean id="exampleDuck" class="RubberDuck">
 <property name="flyBehavior" ref="dontFly"/>
 <property name="quackBehavior" ref="squeakQuack"/>
</bean>
```

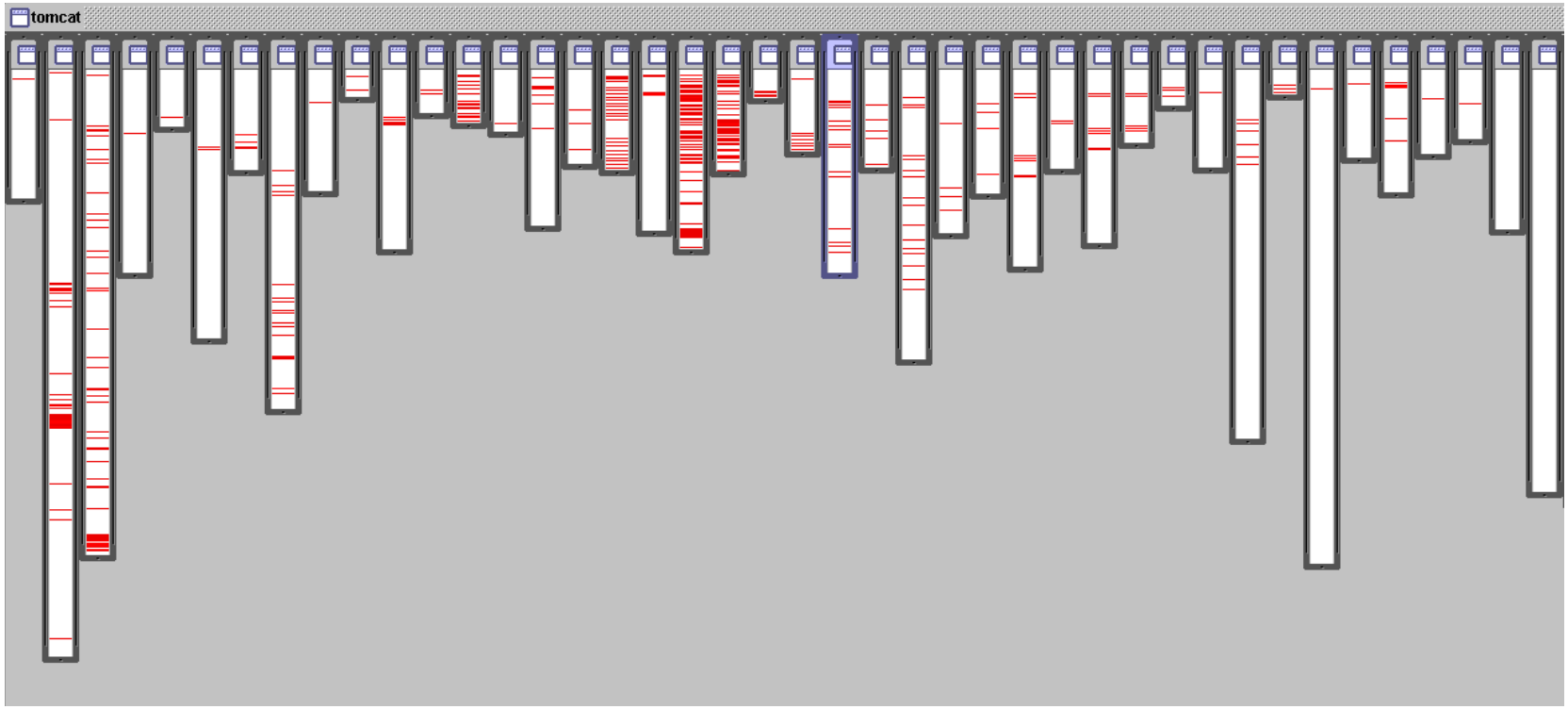
Technique 7: Separation of Concerns / Aspects

JOHANNES KEPLER UNIVERSITY LINZ

- Goal: Each program element does exactly only one thing
- However: Cross-cutting concerns
  - Implementation cuts across a number of source code positions, which leads to
    - Tangling (verwickelt)
    - Scattering (verstreut)

Technique 7: Separation of Concerns / Aspects

JOHANNES KEPLER UNIVERSITY LINZ



Technique 7: Aspects (e.g., AspectJ)

JOHANNES KEPLER UNIVERSITY LINZ

- **Advice** („what“) is woven into program according to **Pointcuts** („where“)
  - Before the execution of a specific method
  - After the normal or exceptional return from a method
  - When a field in an object is modified
- Approaches of **weaving**
  - Bytecode / Source code pre-processing
  - Class load time / Link-time weaving
  - Runtime / Dynamic, execution-time weaving

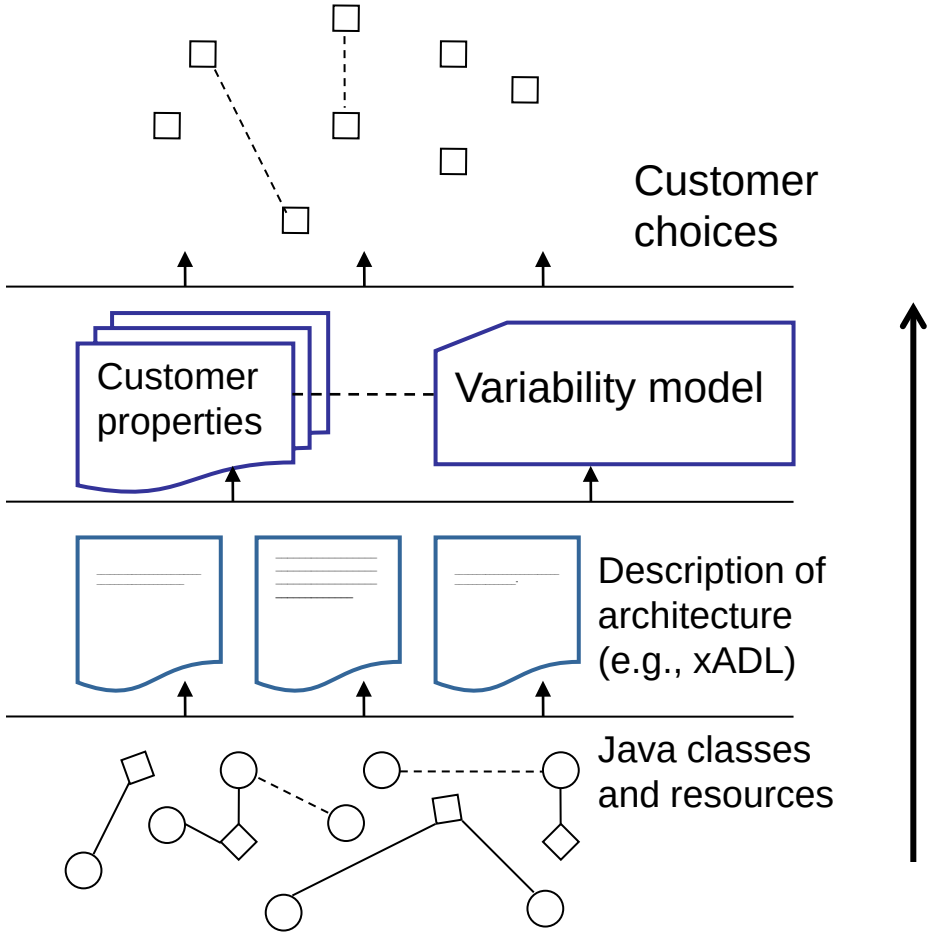
Technique 7: Aspect – Authentication Example

JOHANNES KEPLER UNIVERSITY LINZ

```
aspect authentication
{
// pointcut
  before: call (public void update* (..)) {
  // advice that should be executed when woven into the system
  int tries = 0 ;
  string userPassword = Password.Get(tries) ;
  while (tries < 3 && userPassword != thisUser.password()) {
    // allow 3 tries to get the password right
    tries = tries + 1 ;
    userPassword = Password.Get ( tries ) ;
  }
  if (userPassword != thisUser.password()) then
    //if password wrong, assume user has forgotten to logout
    System.Logout (thisUser.uid) ;
  }
} // authentication
```

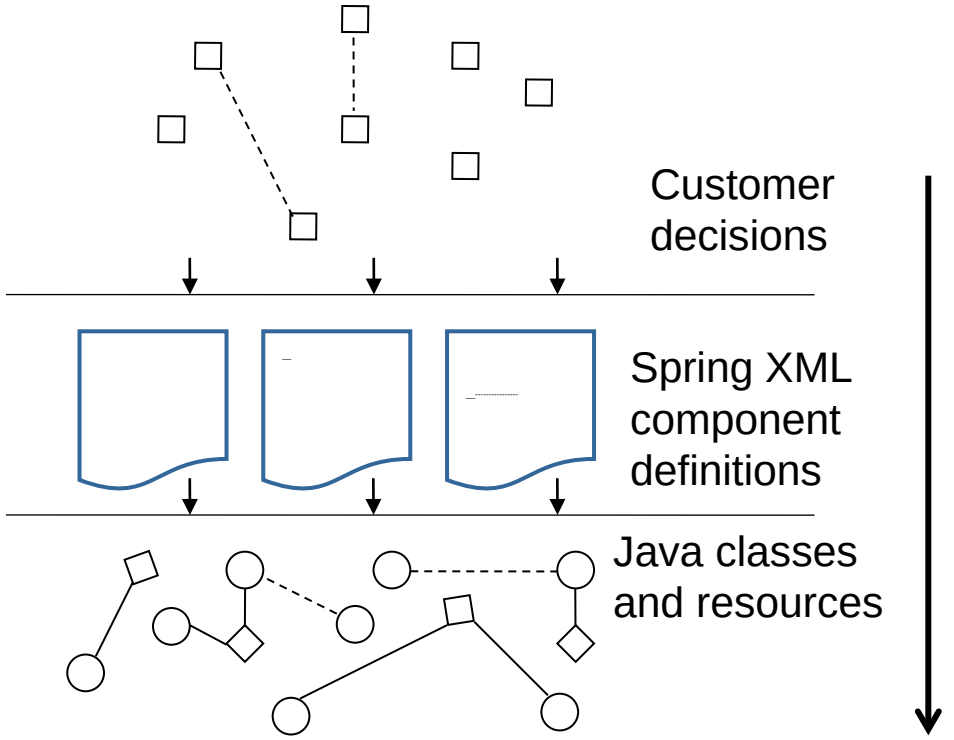
Eliciting Variability: Bottom-up Analysis

- What are the existing reusable assets and their variability?
- Which architectural elements are mandatory? Which are optional?
- How are the assets related to customer decisions?



Eliciting Variability: Top-down Analysis

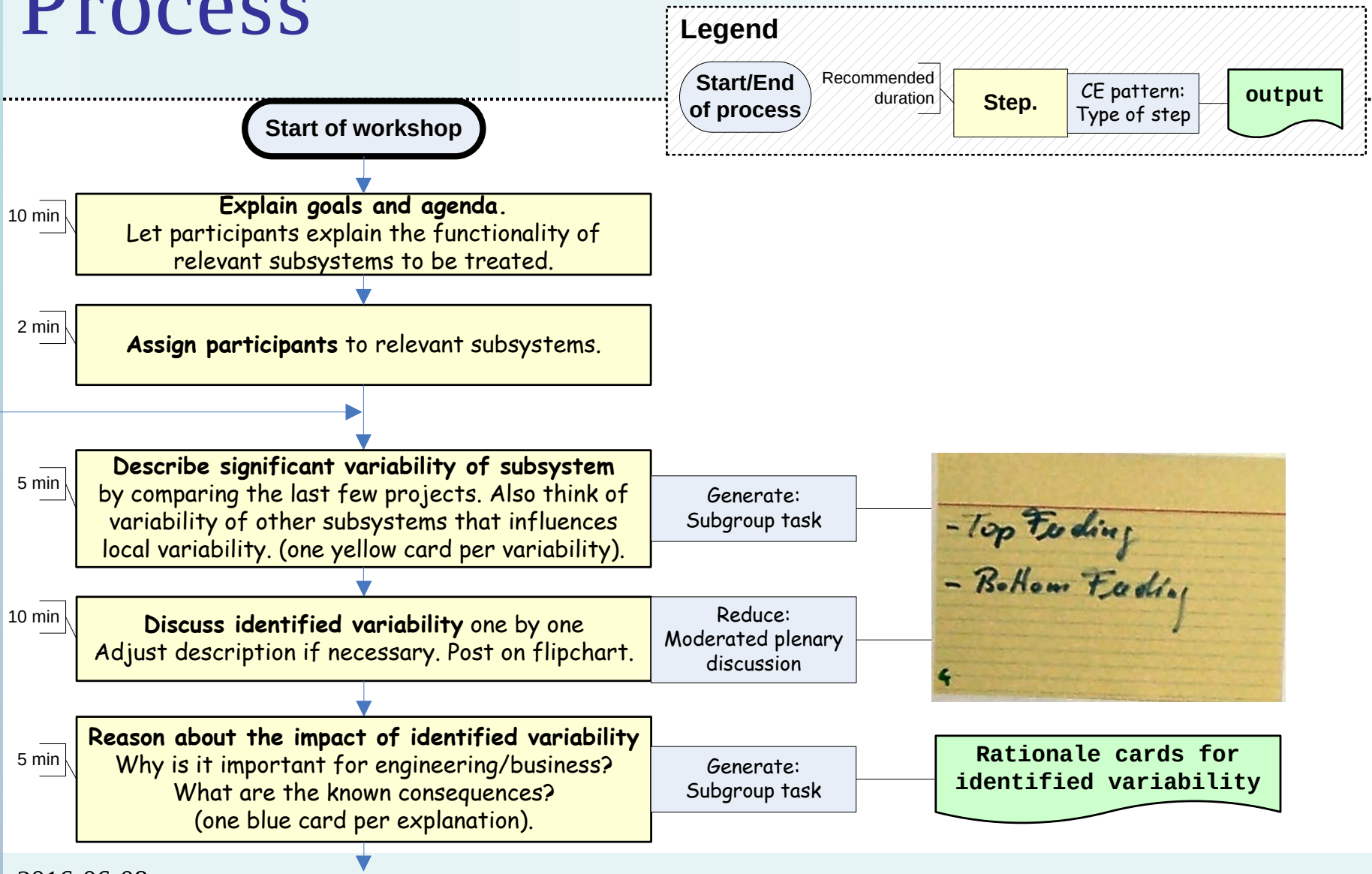
- Which configuration decisions have been taken by customers in the past?
- What are decisions related to the architecture of the deployed system?
- How are the decisions related with each other?



Top-down Analysis Process

- Value-Based Elicitation of Variability [Rabiser et al. 2008]
  1) Finding the most important differences in existing products
  2) Analyzing these differences to develop a shared understanding of the system's basic variability
  3) Documenting the rationale and importance (value, risk) of the identified variability
  4) Developing a shared understanding of the impact of the identified variability (e.g., on engineering, business)
  5) Defining the variability in understandable terms
  6) Prioritizing variability for product derivation

Process

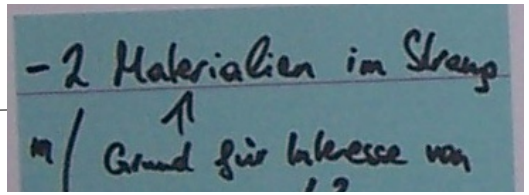

Legend
Start/End of process
Recommended duration
Step.
CE pattern: Type of step
output
Start of workshop
10 min
Explain goals and agenda.
Let participants explain the functionality of relevant subsystems to be treated.
2 min
Assign participants to relevant subsystems.
5 min
Describe significant variability of subsystem
by comparing the last few projects. Also think of variability of other subsystems that influences local variability. (one yellow card per variability).
Generate: Subgroup task
- Top Feeding
- Bottom Feeding
10 min
Discuss identified variability one by one
Adjust description if necessary. Post on flipchart.
Reduce: Moderated plenary discussion
5 min
Reason about the impact of identified variability
Why is it important for engineering/business?
What are the known consequences?
(one blue card per explanation).
Generate: Subgroup task
Rationale cards for identified variability

The Process

Legend

- **Start/End of process**
- Recommended duration
- **Step.** — CE pattern: Type of step
- **output**

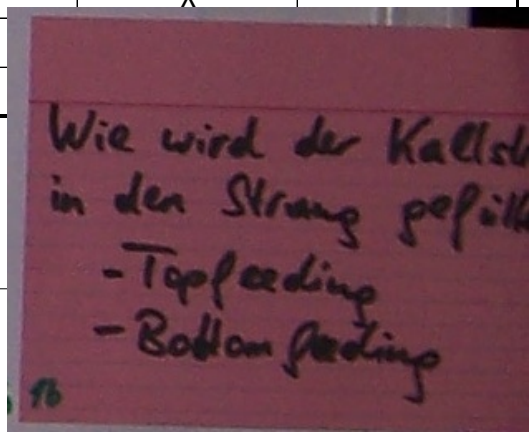
10 min — **Discuss rationale cards** and adapt where necessary. Post on flipchart.
Reduce: Moderated plenary discussion



20 min — **Define importance and impact of variability** using rationale cards.

Example:	*M1 Relevancy*	*Local Impact*	*System-wide Impact*
feedingType	D	X	
nrOfStrands	Y		
casterType	N		

5 min — **Find questions** to be answered to address identified variabilities. (one red card per question).

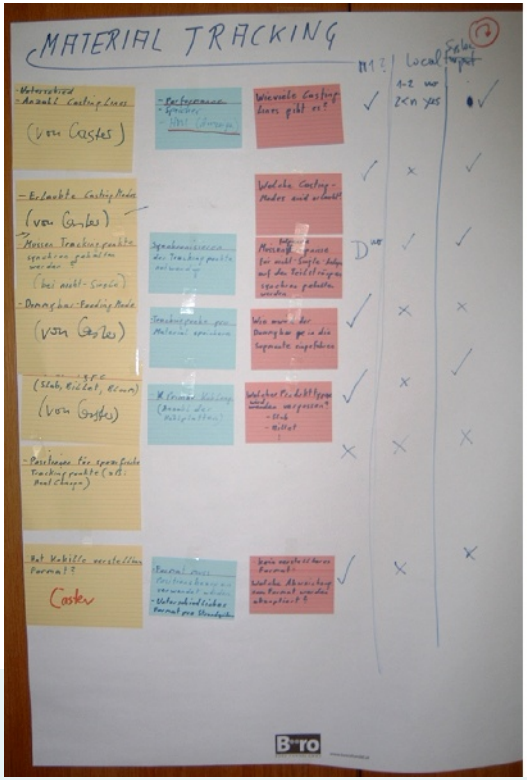


10 min — **Discuss questions** and adjust where necessary. Post on flipchart.
Reduce: Moderated plenary discussion

End of workshop

Facilitation Support

- Groups of engineers and project managers with experience in particular subsystems
- One moderator, two scribes
- Cards and flipcharts (GSS tools also possible)



- Variability
- Rationale
- Variation points
- Importance and impact table

Lessons Learned

- Define time boxes for each activity
- Ensure focus but don‘t limit the creative process
- Use results to provide feedback quickly
- Distinguish roles of moderator and scribe
- Complement results with variability recovery tools for ecliting “technical variability“
- There are different levels of variability

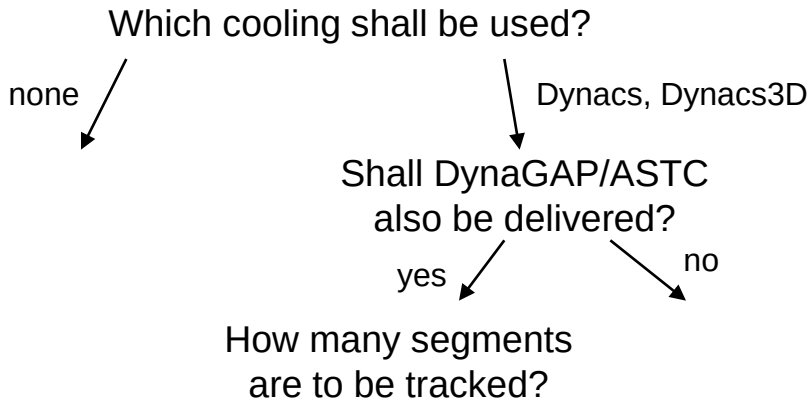
Variability Modeling

- Decision Modeling
- Feature Modeling
- Orthogonal Variability Modeling
- UML-based Variability Modeling
- Other approaches, e.g., CVL

Decision-Oriented Variability Modeling

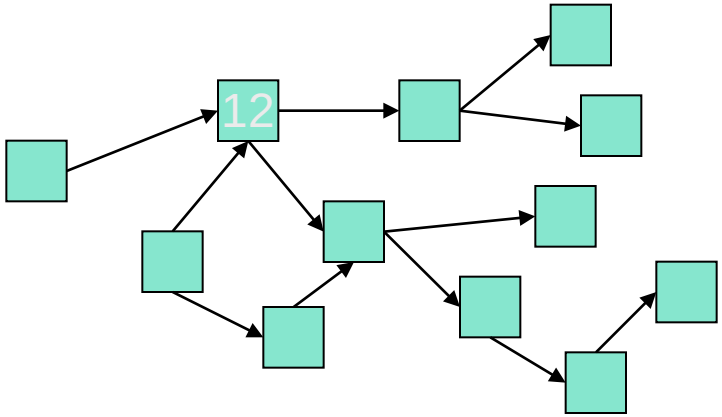
JOHANNES KEPLER UNIVERSITY LINZ

Decisions



- Variability
  - which components?
- Parameterization
  - Component configuration?

Components



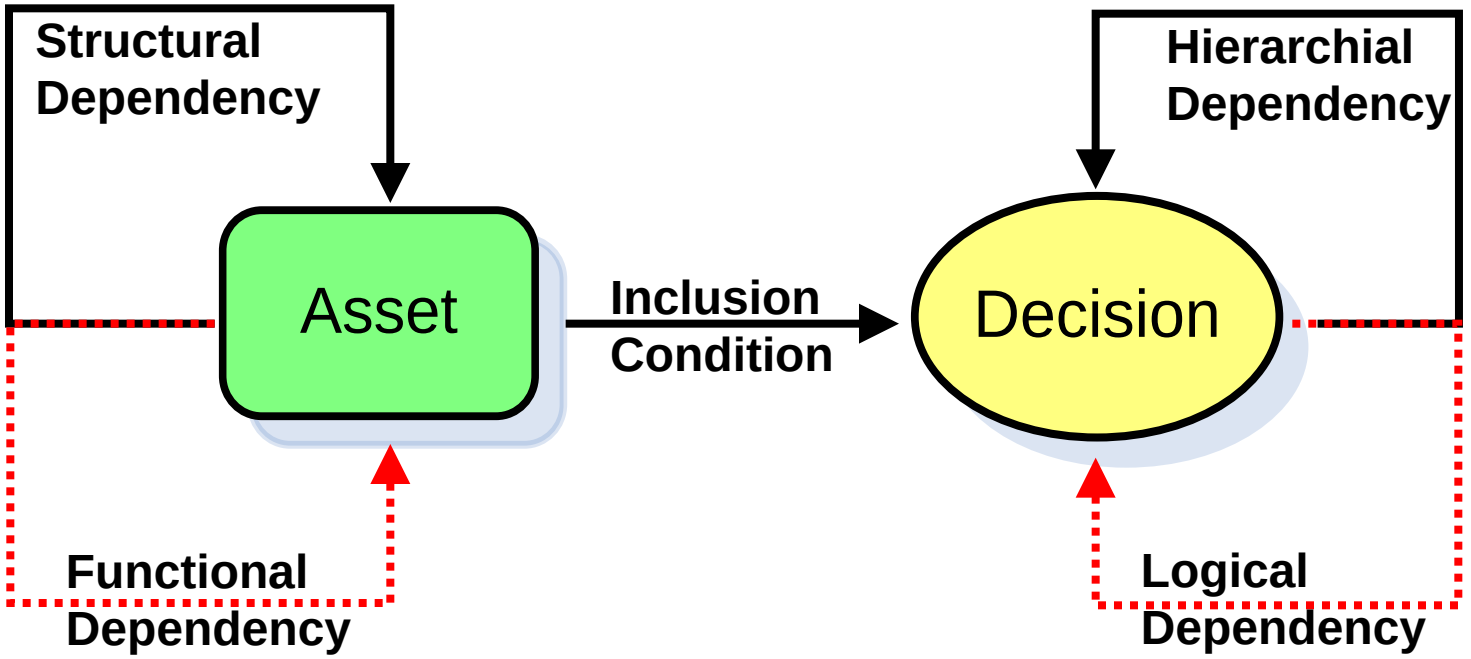
DOPLER Meta-Meta-Model

Asset

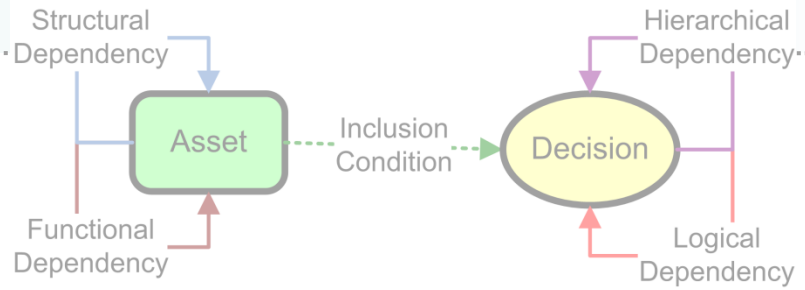
Describes existing reusable element

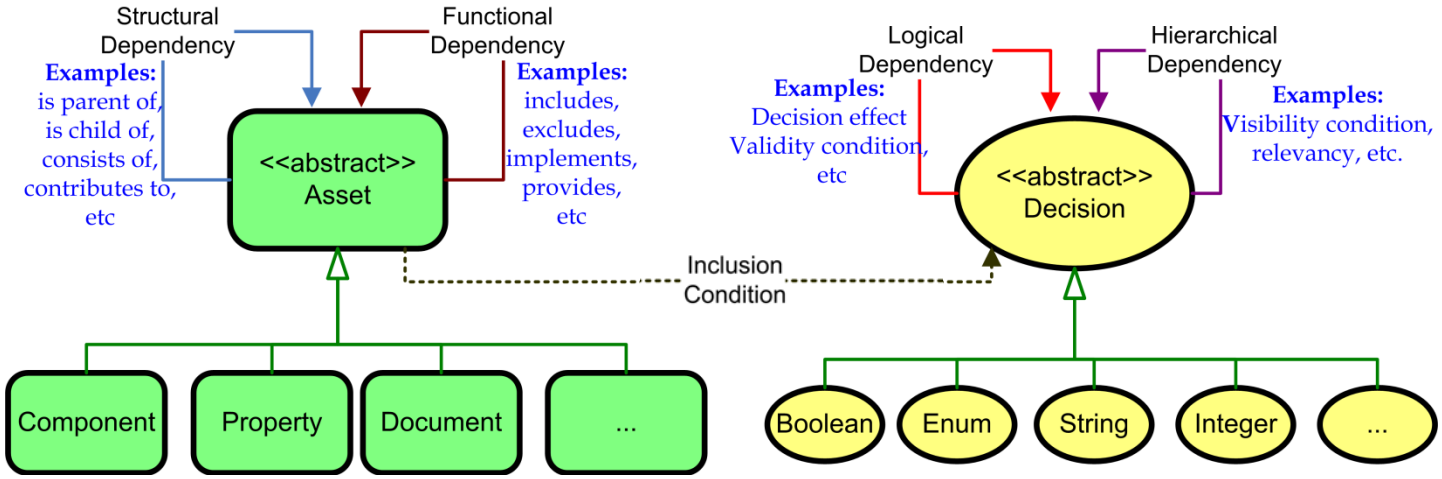
Decision

Describes external and internal variability

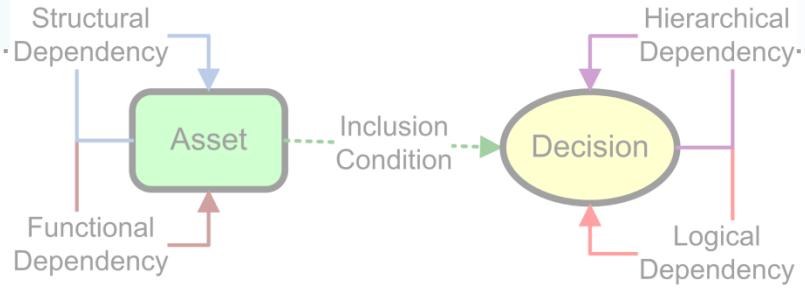


Dealing With Diverse Domains

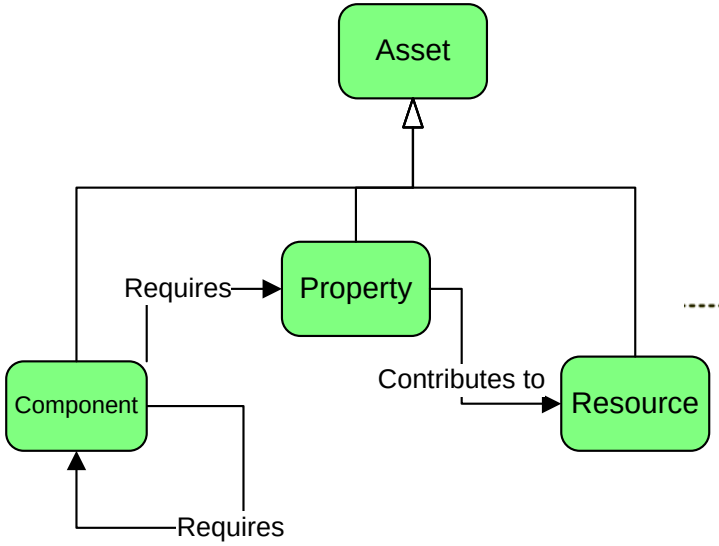


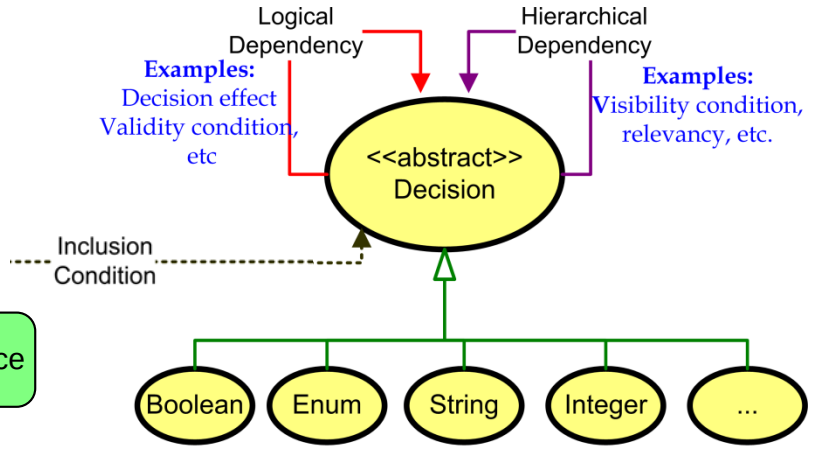


Dealing With Diverse Domains

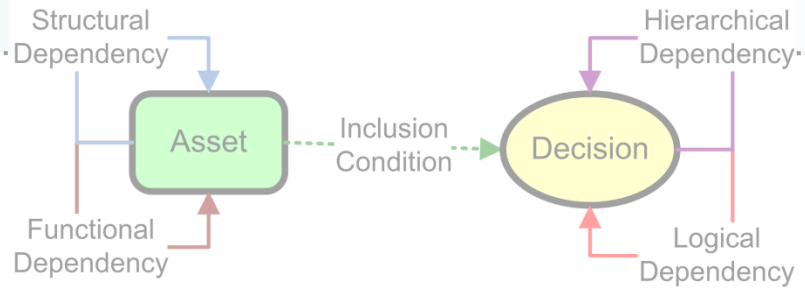


Component-based Software at Siemens VAI

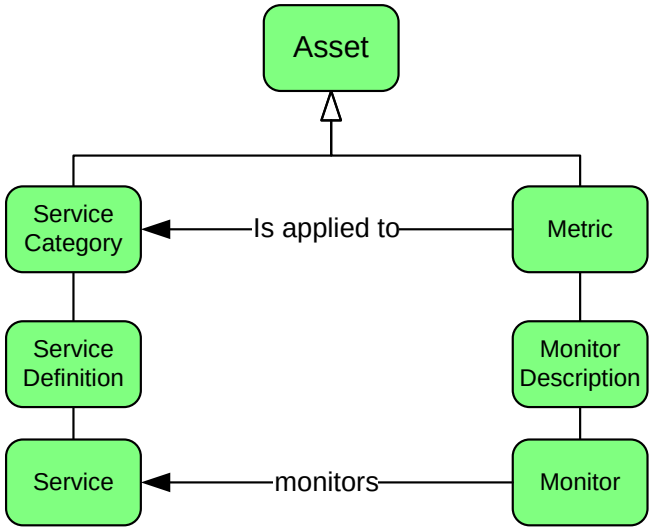


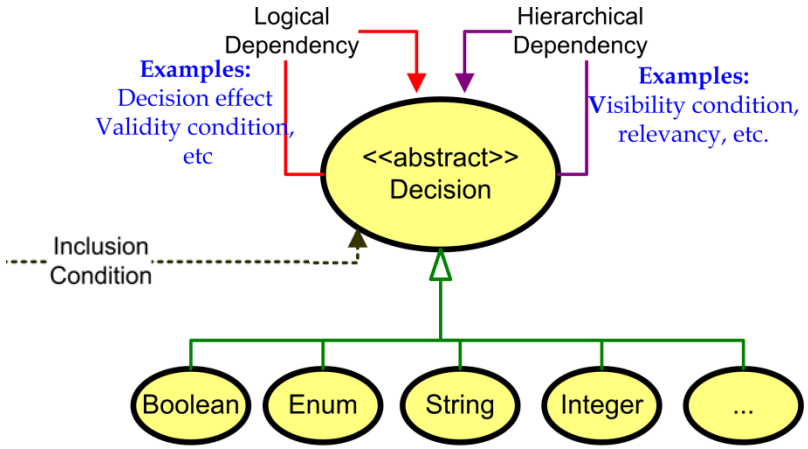


Dealing With Diverse Domains

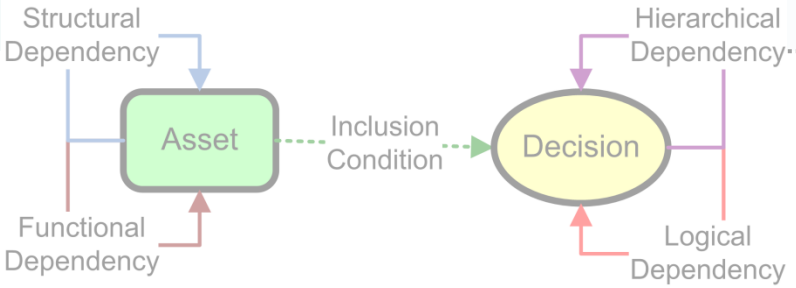


Runtime variability in service-oriented systems

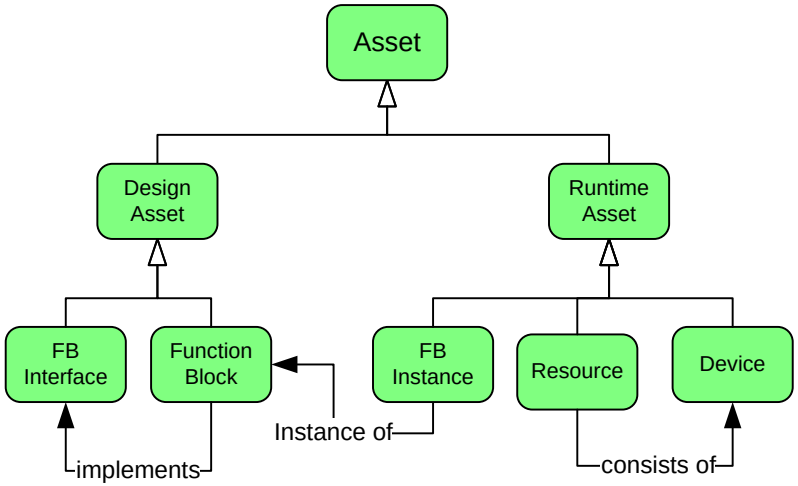


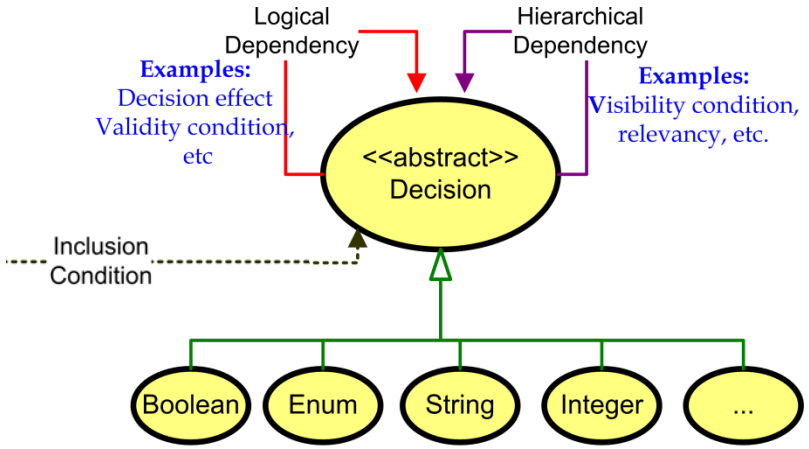


Dealing With Diverse Domains

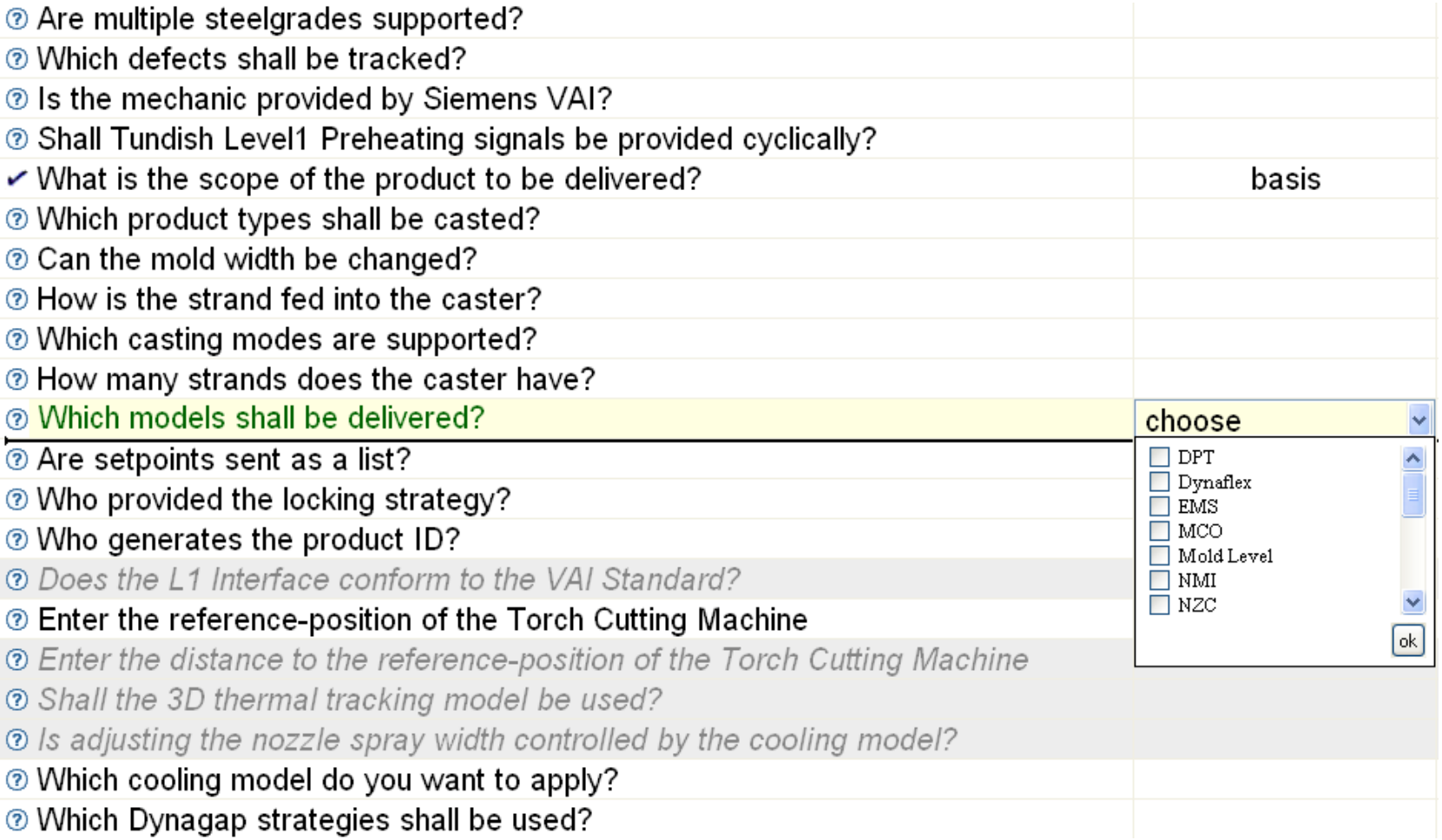


Variability in IEC-61499-based Industrial Automation Systems





Textual Representation

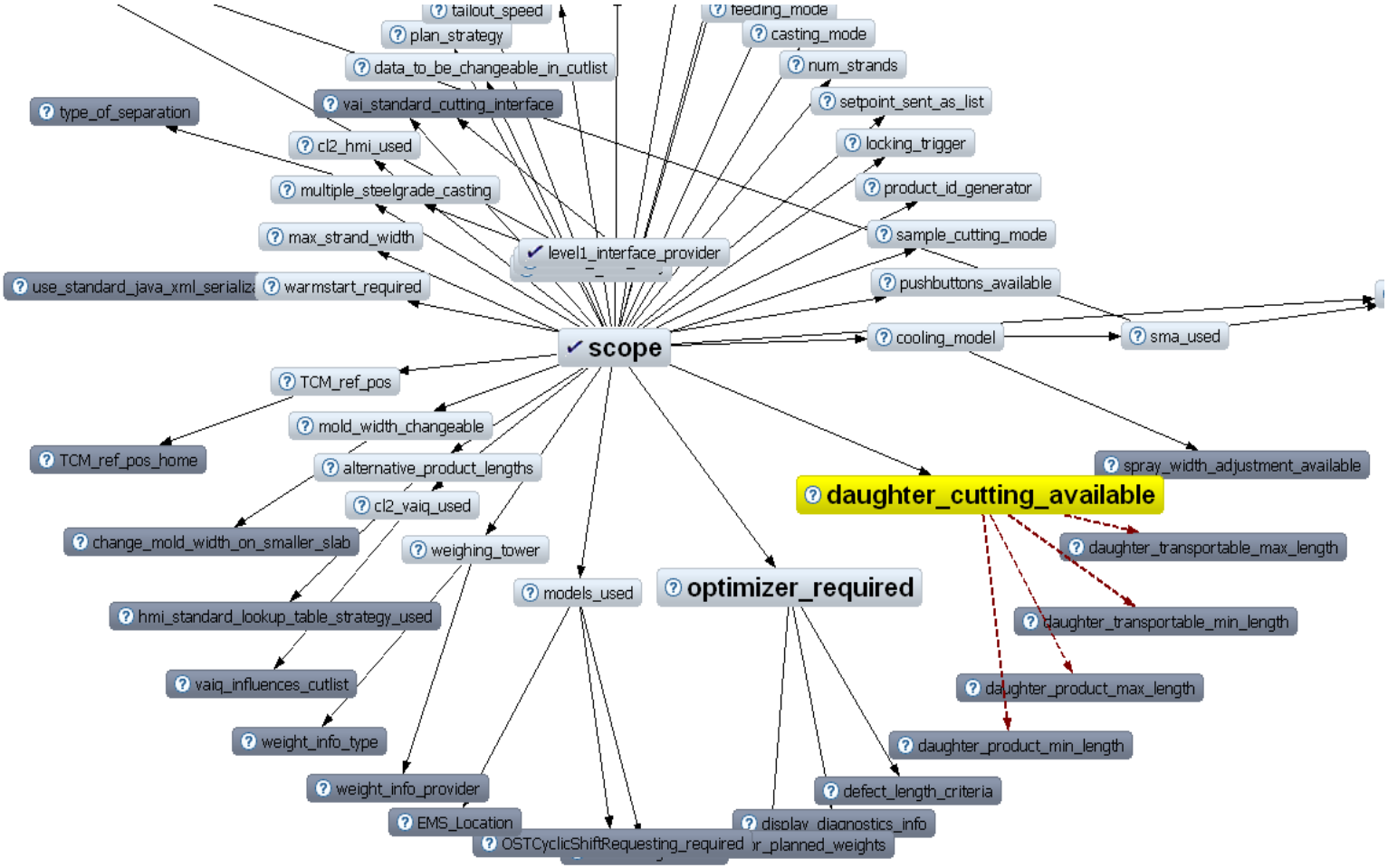


Decision Tables

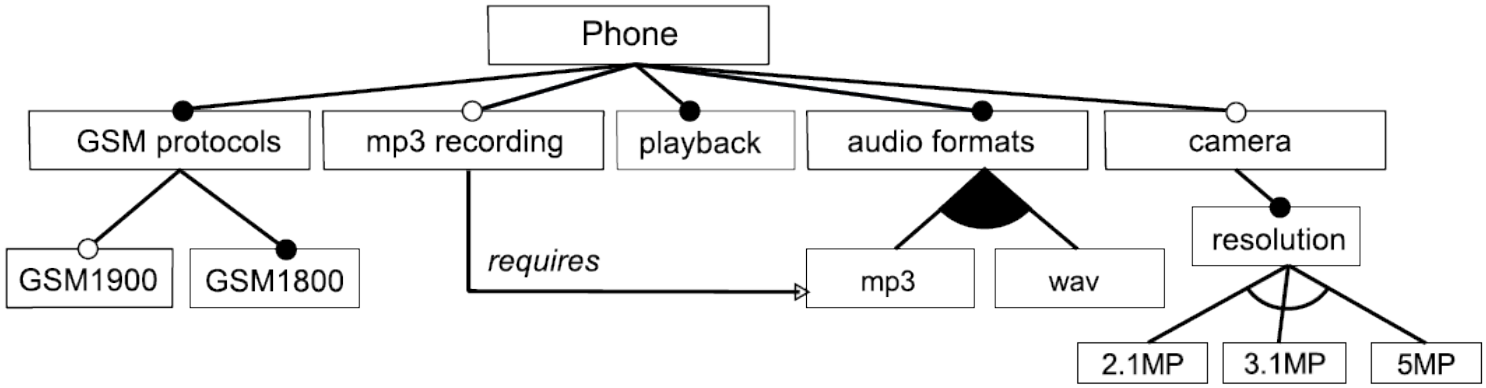
Name	Visibility	Description	Range		Constraints	Binding time
infoSource		What kind of information source does the Web platform have?	Static Dynamic External			Runtime
userManagement		Which kind of user management will be applied?	User, Group Hierarchy		Hierarchy => Group	Compile time
DatabaseSupport	infoSource== Dynamic	What concrete database support will be used?	Access MySQL Oracle			Install time / System initialization

name	relevance	question	range	cardinality	constraints
receiveProtocol		Which protocols do you want to be able to use in your mail client?	POP3, IMAP	1-2	
calendarIncl		Do you want to be able to manage your appointsments within your mail client?	yes, no	1	
taskMgmtIncl	calendarIncl == yes	Do you want to be able to manage tasks too?	yes, no	1	taskMgmtIncl => calendarIncl

Decision Graph



Decision Models (DM) vs. Feature Models (FM)



(a) Feature model in a tree notation—slightly adapted from FODA [42]

decision name	description	type	Range	cardinality/constraint	visible/relevant if
GSM_Protocol_1900	Support GSM 1900 protocol?	Boolean	true | false		
Audio_Formats	Which audio formats shall be supported?	Enum	WAV | MP3	1:2	
Camera	Support for taking photos?	Boolean	true | false		
Camera_Resolution	Required camera resolution?	Enum	2.1MP | 3.1MP | 5MP	1:1	Camera == true
MP3_Recording	Support for recording MP3 audio?	Boolean	true | false	ifSelected Audio_Formats.MP3 = true	

(b) Decision model in a tabular notation [59, 28]

Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, Andrzej Wąsowski, "Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches", *In: Proceedings 6th Int'l Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany*, 2012.

Feature modeling

***features – end user's understanding of the general capabilities of systems in the domain** – and the relationships among them*

- FODA method (1990)
- Fundamental to Feature-oriented Software Development
- Many extensions
  - Group cardinalities [Riebisch et al. '02]
  - Feature cardinalities [Czarnecki et al. '05]
  - Feature inheritance [Asikainen et al. '06]
- Surveys, e.g., [Hubaux et al. 2010, Schobbens et al. 2006, etc.]

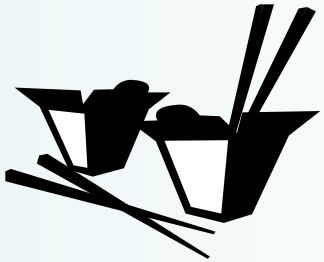
Decision modeling

*set of **decisions** adequate to **distinguish among the members** of a product family useful to **guide** the adaptation of **application engineering** work products*

- Synthesis method (1991)
- Inspired by industrial applications
- Diverse approaches
  - FAST [Weiss and Lai 1999]
  - DOPLER [Dhungana et al. 2011]
  - Schmid and John [Schmid and John 2004]
- Surveys, e.g., [Schmid et al. 2011]

Dimension	Feature Modeling	Decision Modeling
Applications	**div. applications**: concept modeling, variability and comm. modeling; derivation support	variability modeling; **derivation** support
Unit of variability	**features**	**decisions**
Orthogonality	**mostly** used in **orthogonal** fashion	**orthogonal**
Data types	**comprehensive set of basic types**	
Hierarchy	**essential** concept, single appr.	**secondary** concept, div. appr.
Dependencies and Constraints	**no standard** constraint language but similar range of approaches (Boolean, numeric, sets)	
Mapping to artifacts	**optional** aspect (no standard mechanism)	**essential** aspect (no standard mechanism)
Binding time and mode	not standardized, **occasionally supported**	
Modularity	**no standard** mechanism; feature hierarchy plays partly this role	**no standard** mechanism; decision groups play partly this role
Tool aspects	mainly **trees**	div. vis. incl. tree, **workflow**

4 key differences



JOHANNES KEPLER UNIVERSITY LINZ

Feature Modeling

- Focus on modeling commonalities and differences
- Hierarchy essential with uniform semantics
- Mapping to artifacts optional
- Focus on analysis and modeling

Decision Modeling

- Focus on modeling differences
- Hierarchy secondary with varied semantics
- Mapping to artifacts essential
- Focus on application engineering

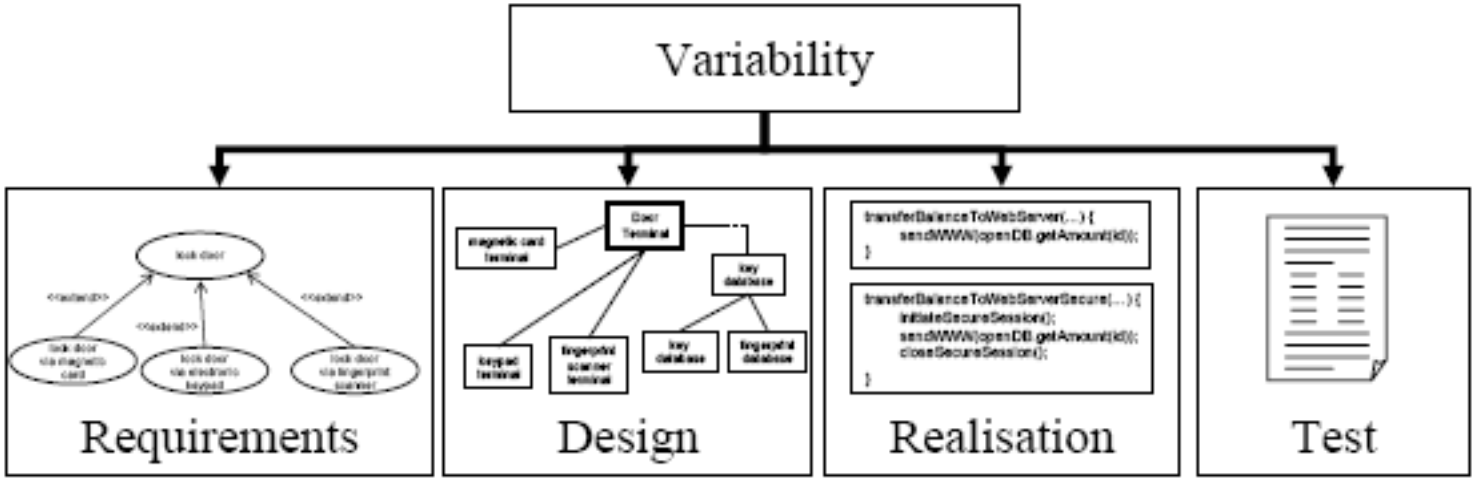
More commonalities than differences; differences are mainly historical!

Specific capabilities of approaches are much more important when selecting an approach than classification as DM or FM

Orthogonal Variability Modeling

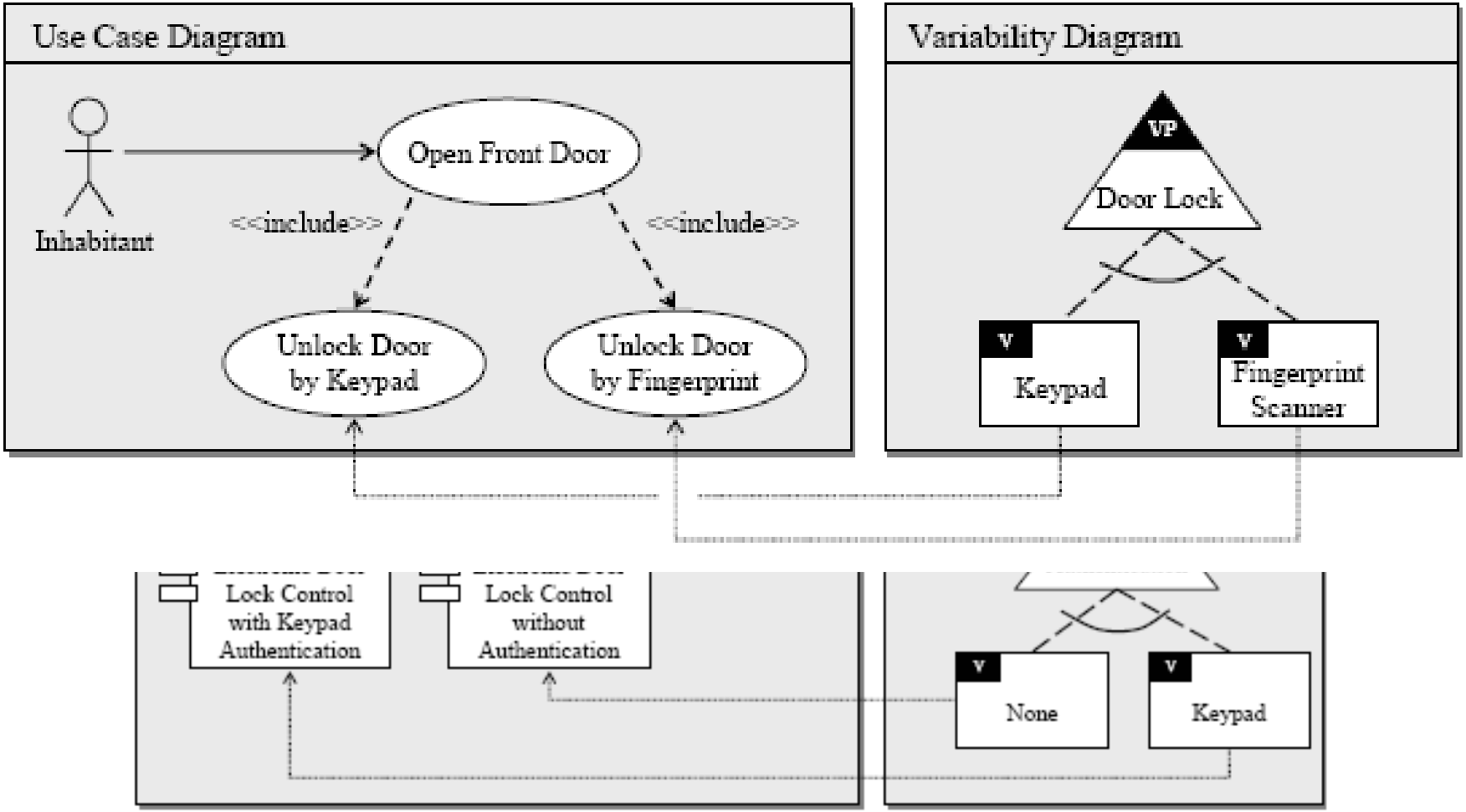
[Pohl et al. 2005]

- Motivation: Feature models quickly get huge and complex
- Main aim of OVM: reduce model size and complexity by „only“ documenting the variability aspects of arbitrary PL artifacts
- Basic concepts
  - Variation point – describes „what does vary“
  - Variant – shows „how does it vary“
  - Variation points offer variants



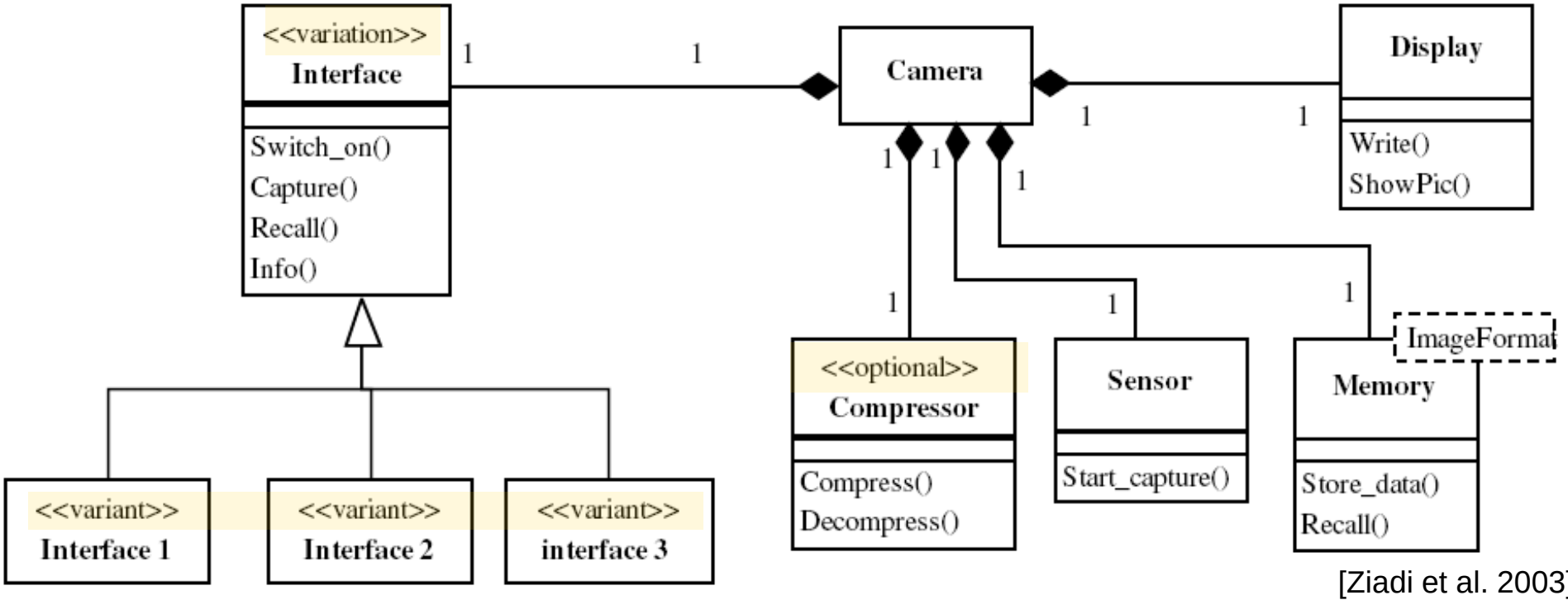
OVM – Example

[Pohl et al. 2005]



UML-based approaches

- e.g., KobrA *[Atkinson et al. 2002]*, PLUS *[Gomaa 2005]*, …
- Variability is handled by extending UML using UML stereotypes and (OCL) constraints



[Ziadi et al. 2003]

Exercise 1

- **List of capabilities of an Instant Messaging Applications Software Product Line**
- **Draw a feature diagram *or* create a decision model**
  - For feature diagrams, use a feature modelling tool, e.g., Feature IDE (http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/)
  - For decision models, use a tabular notation
  - Capabilities are not organized or sorted in any way. Try to figure out the dependencies among the given capabilities
    - Which capabilities are mandatory? Which are optional? Which alternative? Which capabilities require/exclude each other?

Exercise 1

- Explicitly **document and describe your design ideas**: Why did you model features/decisions and dependencies the way you did?
- Also shortly **describe the benefits and drawbacks of the notation you used**.
- Send your work (variability model + text describing design ideas and benefits/drawbacks) **as one PDF** (**Lastname_MatrNr_E1.pdf**) to **rick.rabiser@jku.at** no later than **4th of April 2016 (12:00, noon)**
  - Please take care of the layout and resolution of your model! It should be readable in print out form and not require zooming to 500% in the PDF!

Exercise 1: The fictitious software product line and its capabilities

JOHANNES KEPLER UNIVERSITY LINZ

- All instant messaging applications allow the user to manage his/her account (account management), to manage his/her contacts (contact management), to send messages to each contact (instant messaging), and to (Audio) call each contact via VOIP.
- Some instant messaging applications also allow the user to define status messages (via the account management feature).
- Some instant messaging applications also support Video calls among contacts.
- Every instant messaging application supports participating in group chats (several contacts exchanging instant messages with each other concurrently). Only some instant messaging applications allow users to organize (initiate) such group chats themselves.
- Some instant messaging applications support importing and exporting contacts via the contact management feature.
- All instant messaging application support searching for contacts via the contact management.
- Contacts can be stored locally or, alternatively, online. If online contact management is chosen, searching for contacts provides additional capabilities (extended web search).
- Some instant messaging applications allow users to send arbitrary files to contacts (file sharing).
- Some instant messaging applications also allow users sharing their screen (screen sharing) with selected contacts. This feature only works if file sharing is available.

Next Lecture (16.3.)

- Feature Modeling (RL)