Advanced Software Engineering

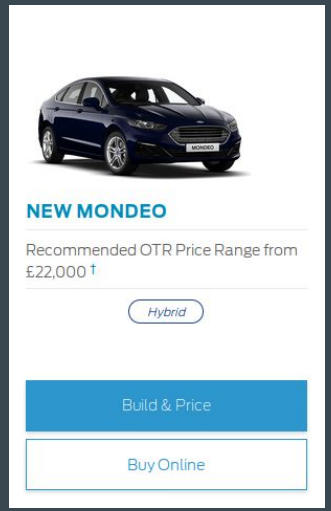
Product Line Engineering in Software-Intensive Systems

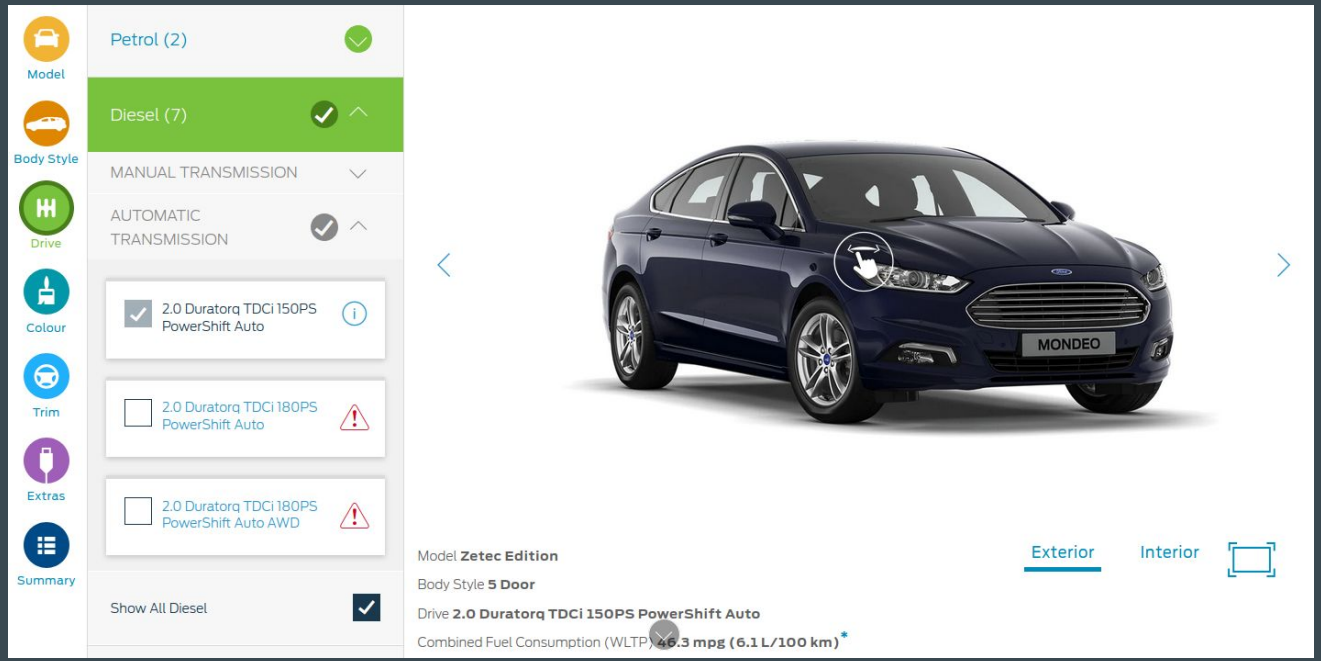
Variable Products

https://www.ebay.com

(Software) Products are getting increasingly variable, requiring flexible configuration and development/manufacturing

Product Configuration - Car





https://www.ford.co.uk/

Product Configuration - Computer

View gallery

Can't decide? We're here to help.

Processor

Which processor is right for you?

- 2.4GHz quad-core 8th-generation Intel Core i5 processor, Turbo Boost up to 4.1GHz
- 2.8GHz quad-core 8th-generation Intel Core i7 processor, Turbo Boost up to 4.7GHz + $300.00

Memory

How much memory is right for you?

- 8GB 2133MHz LPDDR3 memory
- 16GB 2133MHz LPDDR3 memory + $200.00

Storage

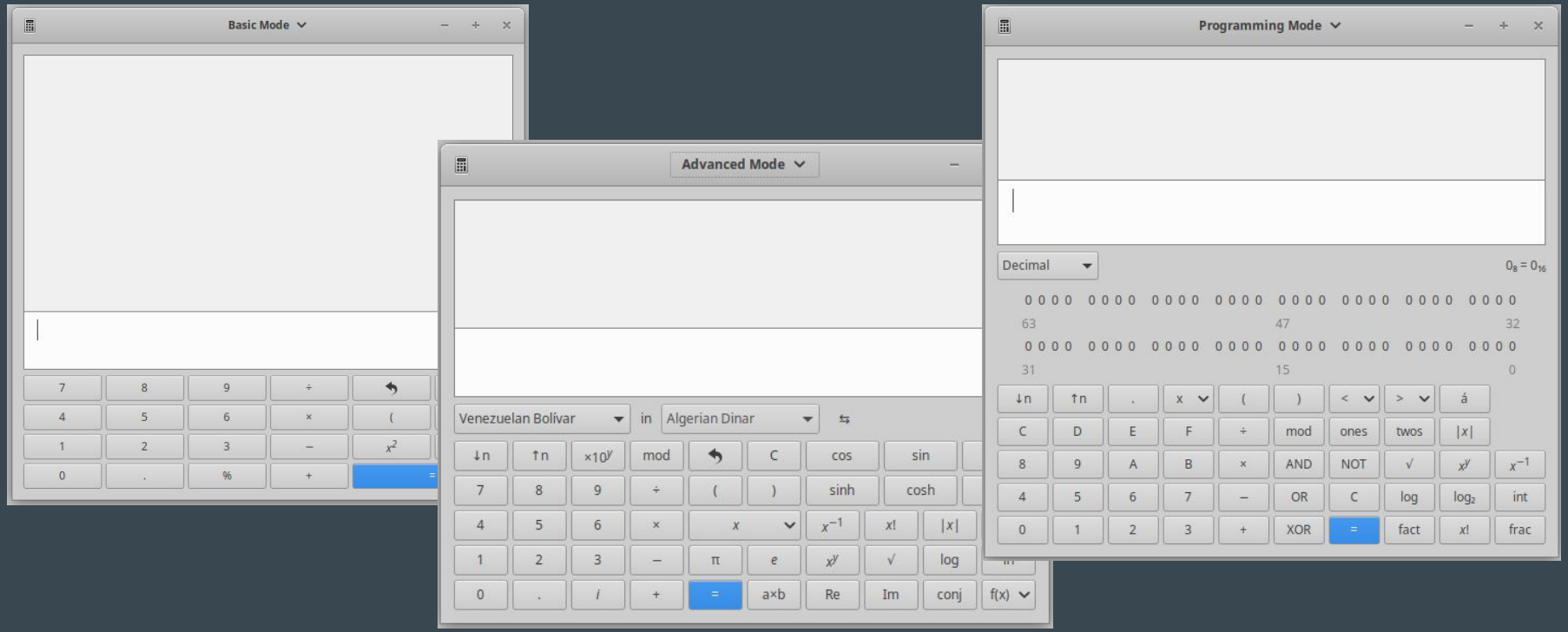
How much storage is right for you?

- 256GB SSD storage
- 512GB SSD storage + $200.00

https://www.apple.com/

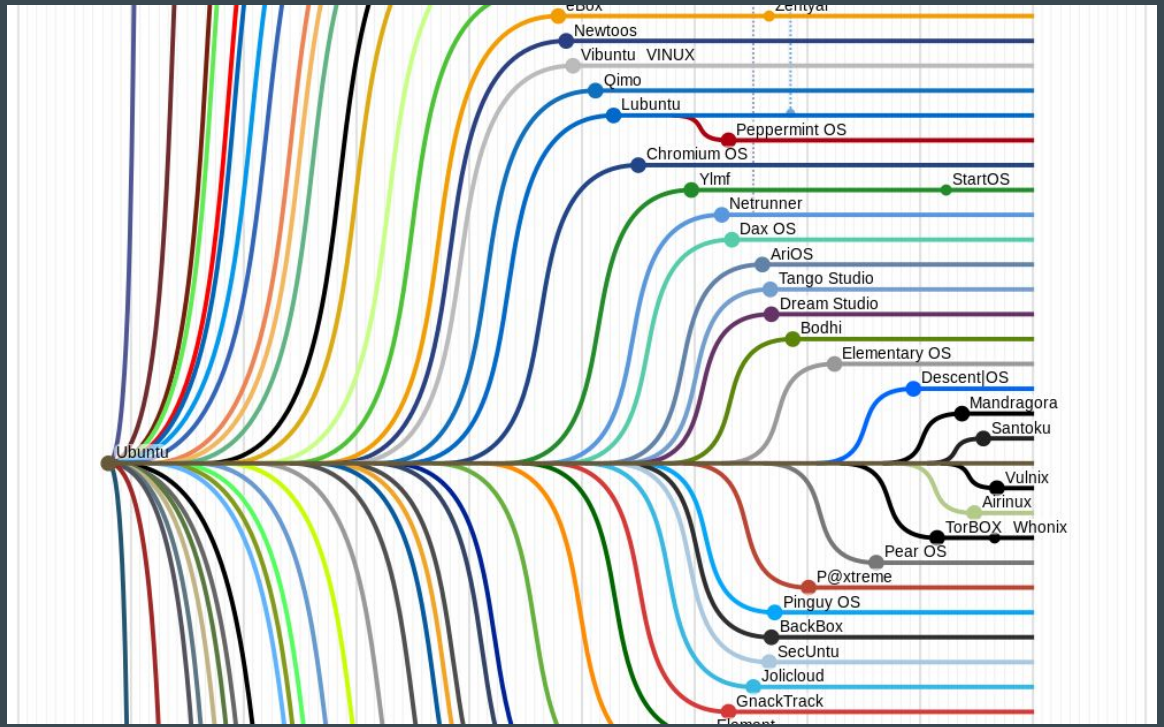
Variability in Software

Calculator Variants



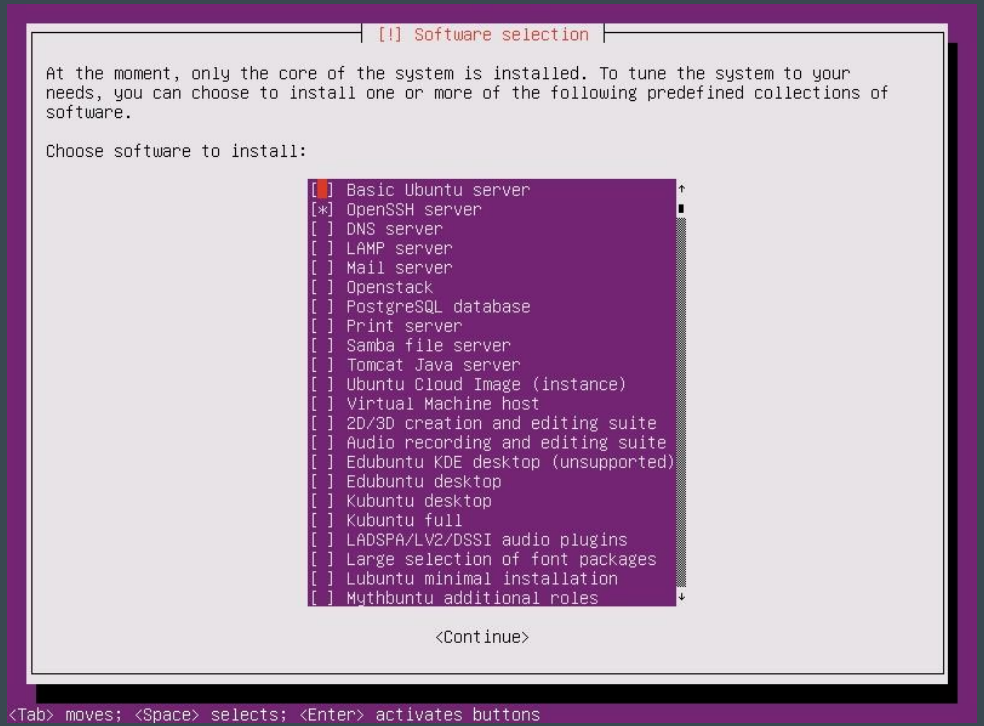
Linux Calc

Ubuntu Family Tree



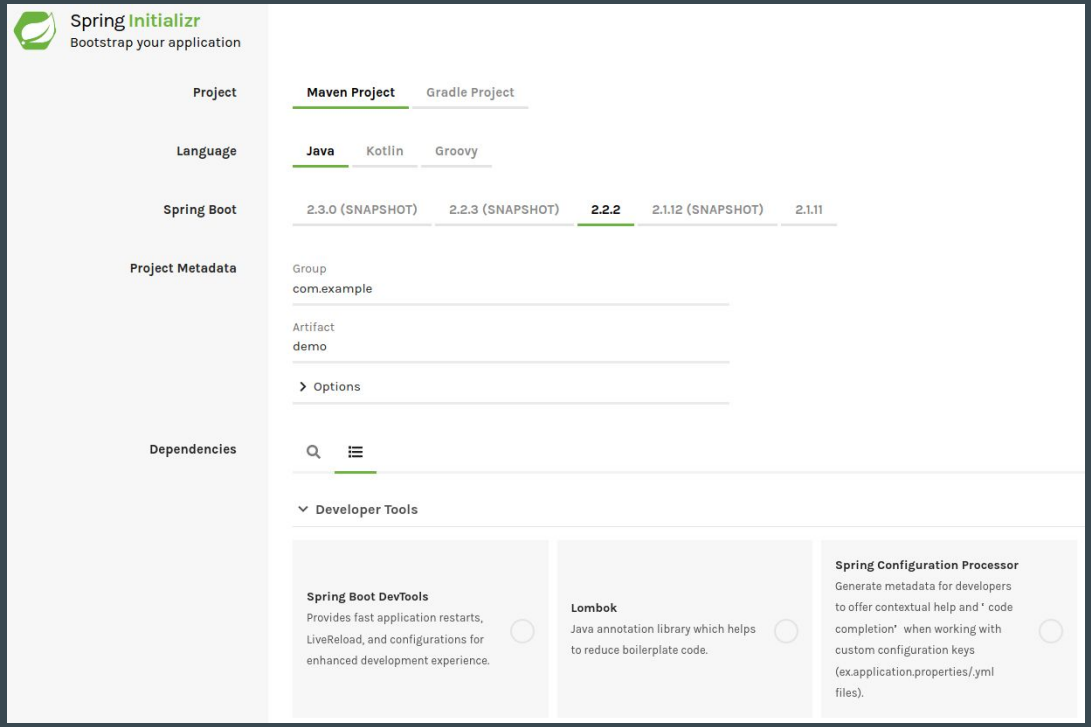
https://commons.wikimedia.org/wiki/File:UbuntuFamilyTree1210.svg

Ubuntu Server Configuration



https://askubuntu.com/questions/153265/what-does-the-basic-ubuntu-server-package-contain-in-software-selection-during

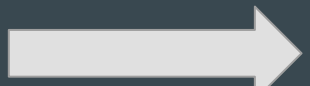
Spring (Boot) Initializer

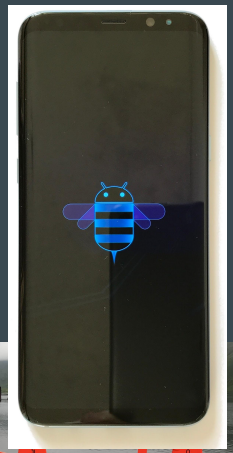


https://start.spring.io/

Software-Intensive Systems

Many variants of Software-Intensive Systems





Inspired by Mathieu Archers slide; Pictures: Wikipedia/Freepik/Saildrone

(Software) Product Line Engineering

What is a Software Product Line?

“A ***software product line*** is a set of ***software-intensive systems*** that ***share*** a common, managed ***set of features*** satisfying the specific needs of a ***particular market segment*** or mission and that are developed from a ***common set of core assets*** in a prescribed way.”

Carnegie Mellon Software Engineering Institute - https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513819

What is a Software Product Line?

“A ***set of programs is considered*** to constitute ***a family***, whenever it is ***worthwhile*** to study programs from the set by ***first studying the common properties*** of the set and ***then determining the special properties*** of the individual family members.”

David L. Parnas - "On the design and development of program families" in Transactions on Software Engineering, SE--2(1):1–9, 1976

What is a Feature?

“A prominent or distinctive ***user-visible/-perceivable aspect***, quality, or characteristic of a (software) system.”

Sidenote: *Several definitions of a feature, Rabiser 2016 states 28 definitions*

Rabiser, D. (2016). Multi-Level Feature Modeling in Industrial Software Ecosystems, 1–192.

What is Software Product Line Engineering?

“Software product line engineering refers to software engineering methods, tools and techniques for creating a ***collection of similar software systems*** from a ***shared set of software assets*** using a common means of production.”

Inspired by Jessie Carbonnels slide; Van Vliet et al. - Software engineering : principles and practice, 1993

Software Product Line Characteristics

collection of similar software systems - family

- operating system, ERP system, ...

shared set of software assets - reuse of commonalities

- code, architecture, requirements, ...

for a specific domain - domain engineering

- steel milling, calculator, cell phone OS, ...

Benefits

- Improved productivity
- Increased quality
- Increased reliability
- Decreased cost
- Decreased certification costs - Really? -> Boeing 737Max disaster
- Decreased labor needs
- Decreased time to market

Carnegie Mellon Software Engineering Institute - https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513819

SPLs induce Complexity

- 33 independent, optional features construct a ***unique variant*** for ***every person on this planet***

- 320 independent, optional features construct ***more variants*** than ***atoms estimated in the universe***

Based on Christian Kästners slide

Types of Variability

Variability in Time vs. in Space

Variability in Time - ***Releases***

- Different ***versions*** of an artifact that are ***valid from a different point in time***

Variability in Space - ***Variants***

- ***An artifact*** that comes in ***different forms*** at the ***same time***

Based on Mathieu Archers slide

Variability by Time

Compile Time - e.g. variability within the source code

Build Time - e.g. variability via build tools

Linking/Binding Time - e.g. variability via Dependency Injection

Runtime - dynamic variability via e.g. OSGI or service discovery

Implementing Variability

Implementing Variability

Parameterization

Configuration Constants

Object Orientation

Dependency Injection

Aspect Orientation

Based on Rick Rabisers slide

Parameterization

```
ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
```

```
public void ls(String... args) {
  List<String> arguments = Arrays.asList(args);
  if (arguments.contains("-a")) {
    doSomething();
  } else if (arguments.contains("-b")) {
    doSomethingElse();
  }
}
```

Clone & Own ‘Variability’

```java
List<String> readFile(String fileName) throws IOException {
  List<String> lines = new ArrayList<>();
  Path filePath = Paths.get(fileName);
  lines = Files.readAllLines(filePath);
  return lines;
}
```

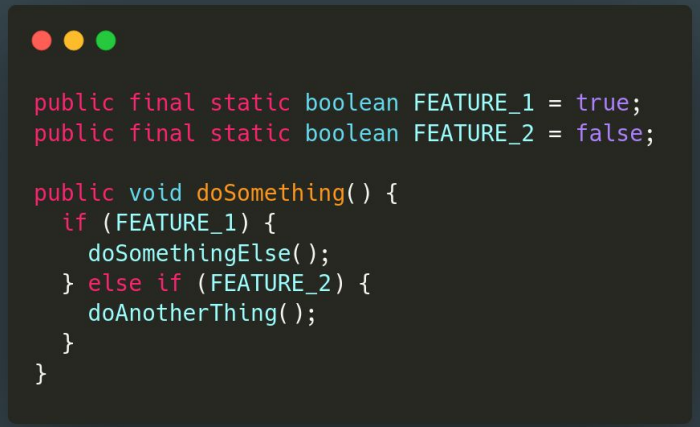
```java
List<String> readFile1(String fileName) throws IOException {
  List<String> lines = new ArrayList<>();
  Path filePath = Paths.get(fileName);
  Charset charSet = Charset.forName("UTF-8");
  lines = Files.readAllLines(filePath, charSet);
  return lines;
}
```

Build Tool Variability

```xml
<profile>
  <id>distribution</id>
  <properties>
    <provided.by>Provided by CDLFlex</provided.by>
    <distribution.release.url>http://cdl.at/arti/libs-release-local</distribution.release.url>
    <distribution.snapshot.url>http://cdl.at/arti/libs-snapshot-local</distribution.snapshot.url>
    <download.url>http://cdl.at/arti/libs-release-local/org/cdlflex/ahy/cdlflex-ahy</download.url>
    <download.user>ahy_update</download.user>
    <download.pw>XXX</download.pw>
  </properties>
  <activation>
    <activeByDefault>true</activeByDefault>
  </activation>
</profile>
```

```
mvn clean install -Pdistribution -DskipTests=true
```

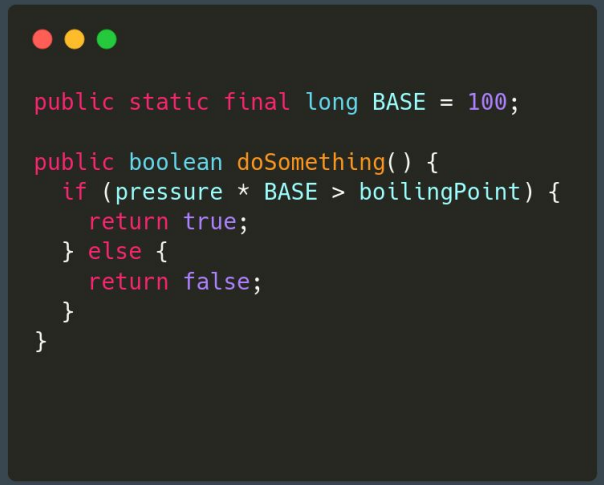
Conditional Compilation



```
public final static boolean FEATURE_1 = true;
public final static boolean FEATURE_2 = false;

public void doSomething() {
  if (FEATURE_1) {
    doSomethingElse();
  } else if (FEATURE_2) {
    doAnotherThing();
  }
}
```

Configuration Constants



```
public static final long BASE = 100;

public boolean doSomething() {
  if (pressure * BASE > boilingPoint) {
    return true;
  } else {
    return false;
  }
}
```

E.g. in combination with external definition files

Object Orientation

Template

Interface

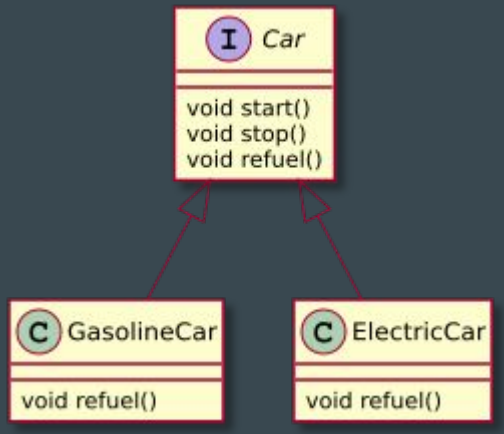
Factory

Strategy

Decorator

Based on Mathieu Archers slide

Interface

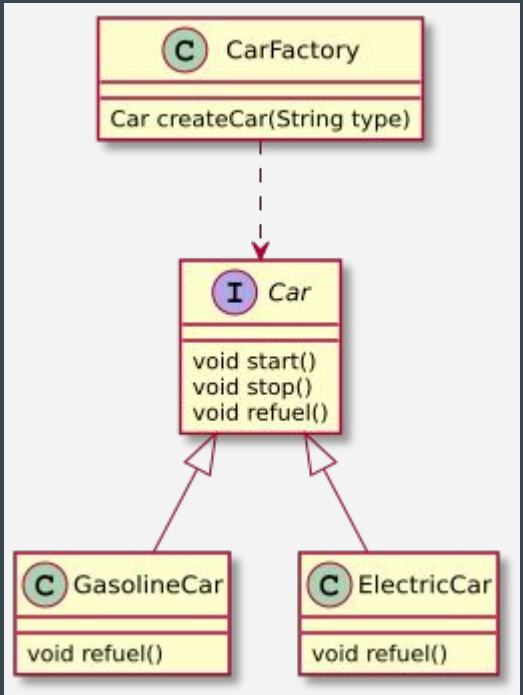



```
interface Car {
  void start();
  void stop();
  void refuel();
}

public class ElectricCar implements Car {
  void refuel() {
    chargeBattery();
  }
}

public class GasolineCar implements Car {
  void refuel() {
    fillupTank();
  }
}
```

Factory Pattern



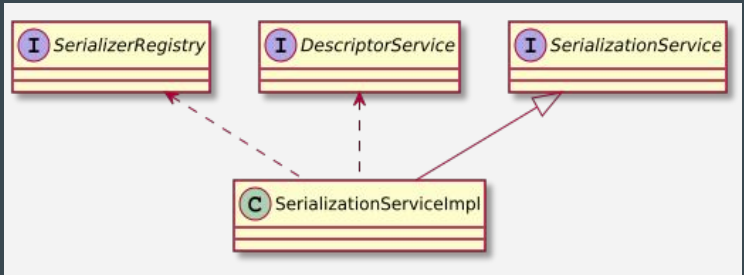

```java
public class CarFactory {
    Car createCar(String type) {
        if (type.equals("Electric")) {
            return new ElectricCar();
        } else if (type.equals("Gasoline")) {
            return new GasolineCar();
        } else { }
    }
}

interface Car {
    void refuel();
}

public class ElectricCar implements Car {
    @Override
    public void refuel() {
        doSomething();
    }
}

public class GasolineCar implements Car {
    @Override
    public void refuel() {
        doSomething();
    }
}
```

Dependency Injection




```xml
<bean id="serializationService"
      class="org.openengsb.SerializationServiceImpl">
  <property name="serializerRegistry" ref="serializerRegistry"/>
  <property name="descriptorService" ref="descriptorService"/>
</bean>
```

But wait... there's more

Variability Modeling

Variability Modeling

Feature Modeling - Kang et. al. 1990

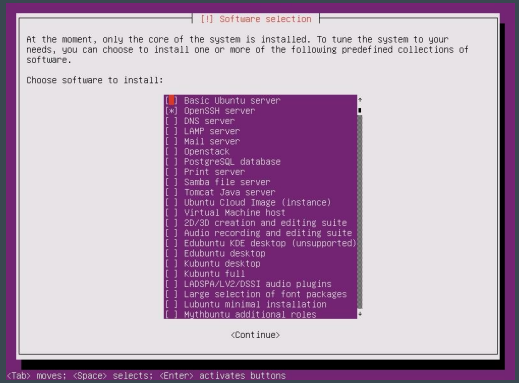
Decision Modeling - McCabe et.al. 1993

Orthogonal Variability Modeling - Pohl et.al. 2005

UML-based Variability Modeling

Other approaches - Superimposed Modeling, Delta Modeling, CVL,TVL

Based on Rick Rabisers slide



Variability Model

```java
public void ls(String... args) {
  List<String> arguments = Arrays.asList(args);
  if (arguments.contains("-a")) {
    doSomething();
  } else if (arguments.contains("-b")) {
    doSomethingElse();
  }
}
```

Base Artifacts e.g. code

Configuration

✓ ✗

Software Generation

Based on Mathieu Archers slide

The 2 Lifecycles

Domain Engineering

- Domain Analysis
- Domain representation
- Domain implementation
- Development ***for Reuse***

Application Engineering

- Product selection
- Product derivation
- Development ***by Reuse***

Based on Jessie Carbonnels slide

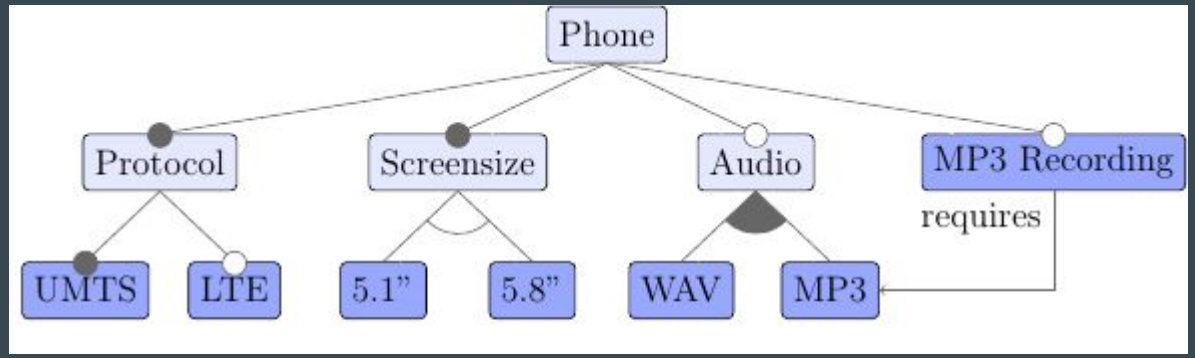
Decision Modeling

Name	Description	Range	Cardinality	Relevancy/Constraints
LTE	Is LTE supported?	true | false		
Screensize	How big is the screen?	5.1" | 5.8"		
Audio	Which audio types are supported?	WAV | MP3	1:2	
Recording	Is MP3 recording supported?	true | false		requires Audio.MP3

Decision models cover ***variability only***

Based on Jessie Carbonnels slide

Feature Modeling



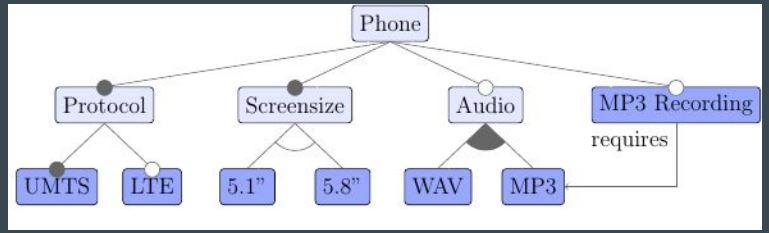
Based on Jessie Carbonnels slide

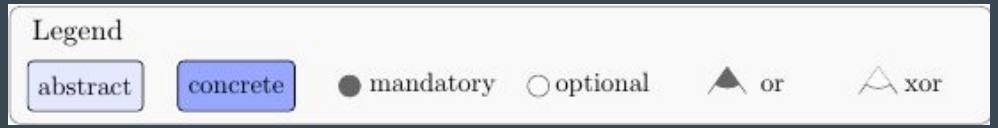
Feature Models

Feature Models

Feature Models

- Family of visual description languages
- allows to describe a set of features and their dependencies
- depict a set of valid combinations of features = ***configurations***
- → requires / ↛ excludes



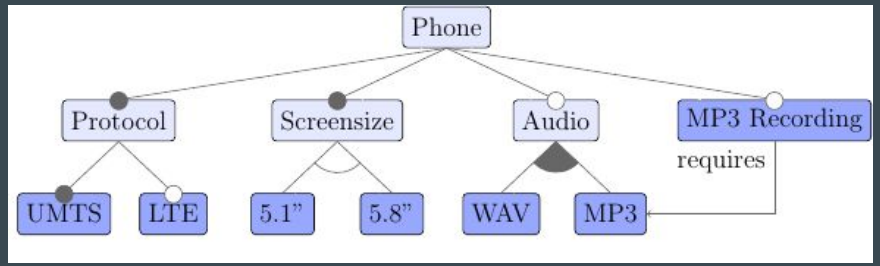


→ requires / ↛ excludes

Based on Jessie Carbonnels slide

Feature Tree

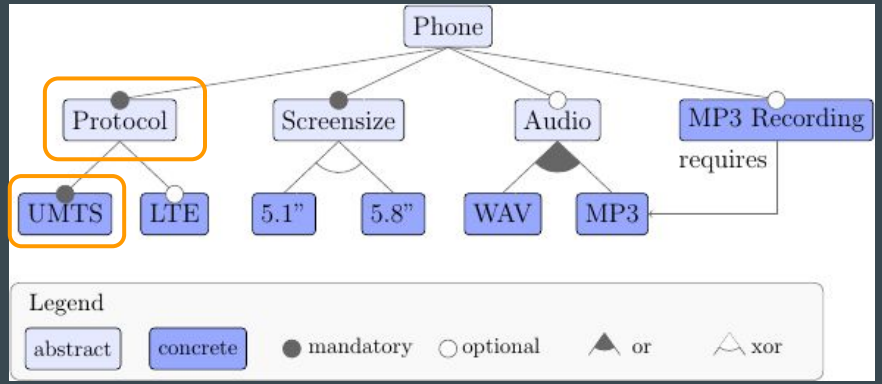
- structure hierarchically the set of features in a tree = ***feature tree***
- ***root feature*** = name of the modeled system
- (top to bottom) from most generalised features to most specialized one
- describe the system in several level of increasing details
- express refinement relationships



Based on Jessie Carbonnels slide

Feature Models

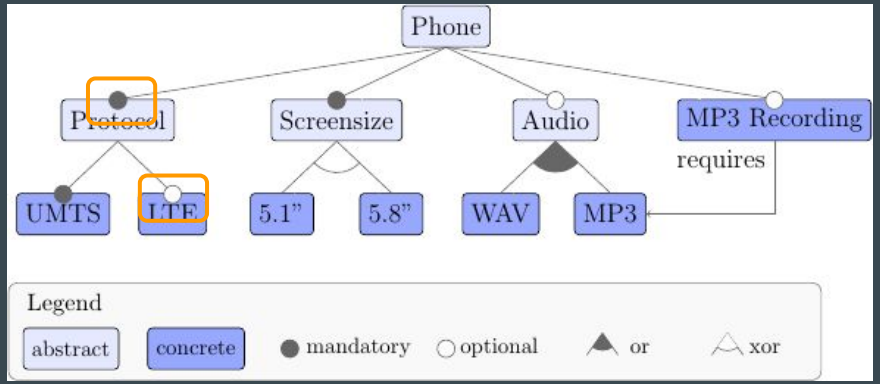
- Abstract Features
  - Act as ***nodes for structuring*** the feature tree
  - ***Cannot be selected*** in a configuration
- Concrete Features
  - Represent specific features that ***can be selected*** in a configuration



Based on Jessie Carbonnels slide

Feature Models

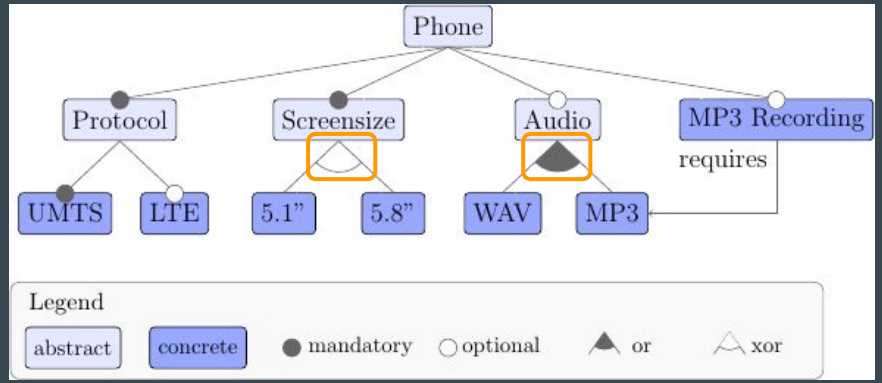
- Mandatory features
  - if the parent is selected, the ***child feature is necessarily selected***
- Optional Features
  - if the parent is selected, the ***child feature can be selected***, or not



Based on Jessie Carbonnels slide

Feature Models

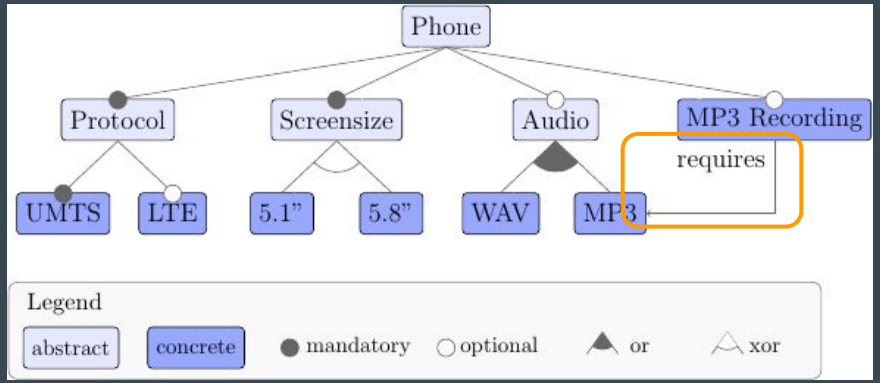
- OR Features
  - if the parent is selected, ***at least one feature*** involved in the group ***has to be selected***
- XOR Features
  - if the parent is selected, ***exactly one feature*** involved in the group ***has to be selected***



Based on Jessie Carbonnels slide

Feature Models

- ⟶ requires
  - if the premise is selected, the ***conclusion is also selected***
- ↛ excludes
  - the two features are ***mutually exclusive***



Based on Jessie Carbonnels slide

Configurations

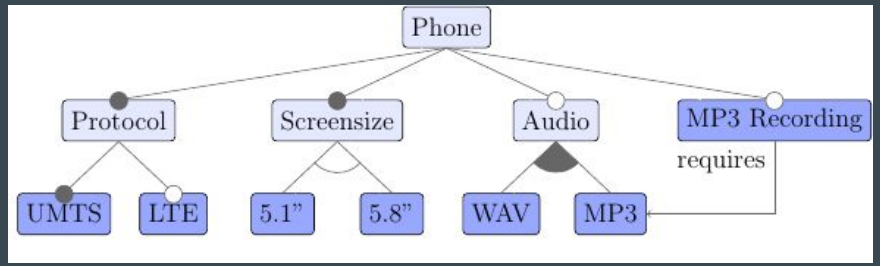
1. {phone, protocol, umts, screensize, 5.1}
2. {phone, protocol, umts, screensize, 5.8}
3. {phone, protocol, umts, lte, screensize, 5.1}
4. {phone, protocol, umts, lte, screensize, 5.8}
5. {phone, protocol, umts, screensize, 5.1, audio, wav}
6. {phone, protocol, umts, screensize, 5.8, audio, wav}
7. {phone, protocol, umts, lte, screensize, 5.1, audio, wav}
8. {phone, protocol, umts, lte, screensize, 5.8, audio, wav}

- Configurations are validated using SAT Solvers
- Problems to solve are NP-complete

Based on Jessie Carbonnels slide

Feature Models

- ***understandable and compact*** way to express variability
- ***combinatorial explosion*** of the possible software variants
- ***potentially large number*** of represented software systems
- ***mapping*** to artifacts needs to be ***defined explicitly***
- ***Software generator*** needs to be implemented



Based on Jessie Carbonnels slide

Take away messages

- ***Variability is everywhere***
  - Like software is everywhere
  - Numerous artefacts, domains, kinds of systems are subject to customization
- ***Engineering variability can be hard***
  - Exponential number of products
  - Basic or sophisticated techniques (than span your entire cursus) exist
- ***You're now aware of that!***
  - Recognize variability
  - Re-visit your cursus and body of knowledge under the angle of variability
  - Apply state-of-the-art techniques

Based on Mathieu Archers slide