31. From Values to Features to Components

Maintaining and Measuring a Product Family of Canvases for the MVFS

Selecting features by Lean (Canvas) Modeling - Grading and Metrics on Canvases

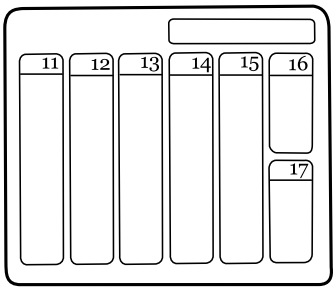
Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Software Engineering Group
http://st.inf.tu-dresden.de
Version 20-0.13, 09.01.21

Literature

- [CM03] Sitt Sen Chok, Kim Marriott. Automatic Generation of Intelligent Diagram Editors. ACM Transactions on Computer-Human Interaction, Vol. 10, No. 3, September 2003, Pages 244–276.
  - This paper introduces (Constraint Multiset Attributed Grammars, CMAG) that describe structure and constraints of hierarchic artefacts
  - https://www.researchgate.net/profile/Kim_Marriott2/publication/220286256_Automatic_generation_of_intelligent_diagram_editors/links/02bfe511a59b70f9ec000000/Automatic-generation-of-intelligent-diagram-editors.pdf
- CMAG are generalized to Constraint Part Attributed Grammars (CPAG) by adding collection-constructors such as tuples, sets, lists grouped by and-ior-xor constructors.
- Bernd Meyer, Kim Marriott, Adrian Bickerstaffe, Lars Knipping. Intelligent diagramming in the electronic online classroom. Human System Interactions, 2009. HSI '09.
  - DOI: 10.1109/HSI.2009.5090975
  - https://www.researchgate.net/publication/224517241_Intelligent_diagramming_in_the_electronic_online_classroom
- Hans de Bruin and Hans van Vliet. Quality-driven software architecture composition. Journal of Systems and Software, 66(3):269--284, 2003. From features to solutions.
  - https://doi.org/10.1016/S0164-1212(02)00079-1
- Jaime Chavarriaga, Carlos Noguera, Rubby Casallas, and Viviane Jonckers. Managing Trade-offs among Architectural Tactics using Feature Models and Feature-Solution Graphs. IEEE. doi:10.1109/ColumbianCC.2015.7333406

Shortcomings of Lean Startup from the Viewpoint of Software Product-Line Engineering

No support for consistent modeling of product lines (no support for feature modeling and feature variation)



No support for canvas modeling (composition and engineering)

No support for staged feature configuration with suppliers

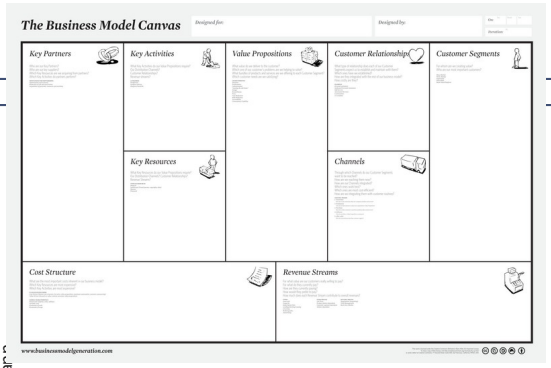
No support for grading and metrics

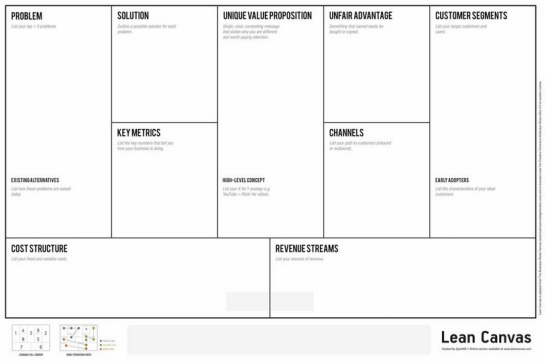
Objectives

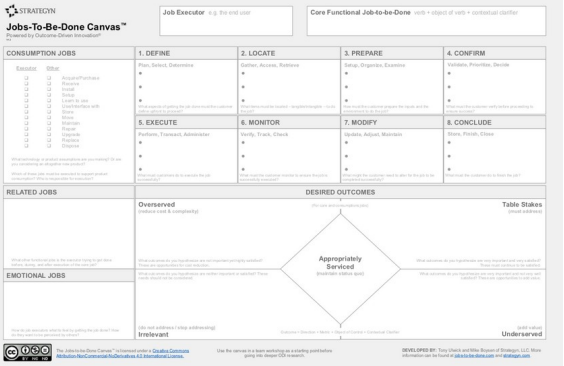
- Know how to compute the cost of an MVFS and MVP
- Know how to use grammars to structure value trees, feature trees, and product component trees
- Know how to bridge the worlds of values, features, and product components
- Know how to prepare scaling with a multi-product feature model

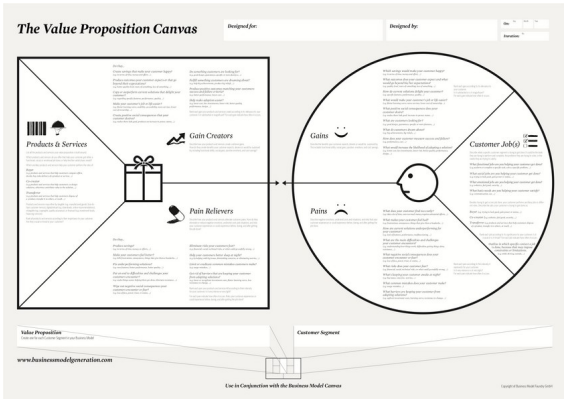
31.1 From Values to Features to Solutions

- Three worlds have to be combined:
  - Values and value propositions of the customer
  - Features of the product or service
  - Solutions to implement the product

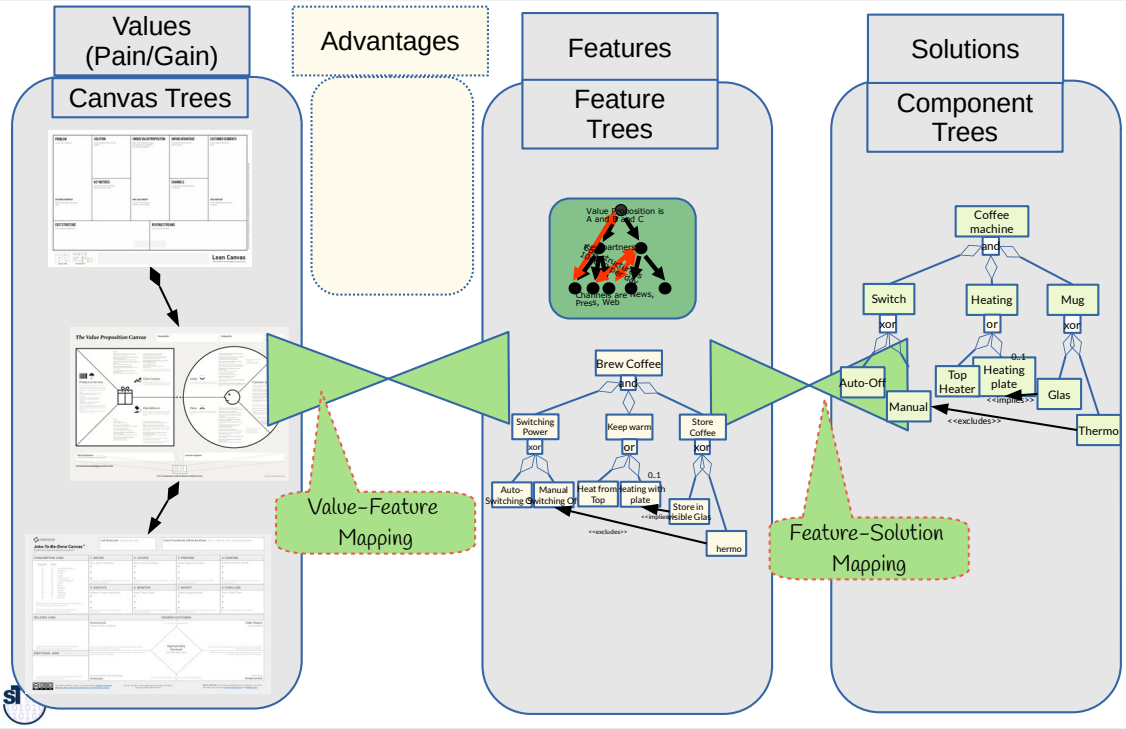

The Business Model Canvas
Designed for:
Designed by:
Key Partners
Key Activities
Value Propositions
Customer Relationships
Customer Segments
Key Resources
Channels
Cost Structure
Revenue Streams
www.businessmodelgeneration.com


PROBLEM
SOLUTION
UNIQUE VALUE PROPOSITION
UNFAIR ADVANTAGE
CUSTOMER SEGMENTS
KEY METRICS
CHANNELS
EXISTING ALTERNATIVES
HIGH-LEVEL CONCEPT
EARLY ADOPTERS
COST STRUCTURE
REVENUE STREAMS
Lean Canvas


STRATEGYN
Jobs-To-Be-Done Canvas™
Job Executor e.g. the end user
Core Functional Job-to-be-Done verb + object of verb + contextual clarifier
CONSUMPTION JOBS
1. DEFINE
2. LOCATE
3. PREPARE
4. CONFIRM
5. EXECUTE
6. MONITOR
7. MODIFY
8. CONCLUDE
RELATED JOBS
DESIRED OUTCOMES
Overserved
Table Stakes
Appropriately Serviced
EMOTIONAL JOBS
Irrelevant
Underserved


The Value Proposition Canvas
Designed for:
Designed by:
Products & Services
Gain Creators
Pain Relievers
Gains
Pains
Customer Job(s)
Value Proposition
Customer Segment
www.businessmodelgeneration.com
Use in Conjunction with the Business Model Canvas

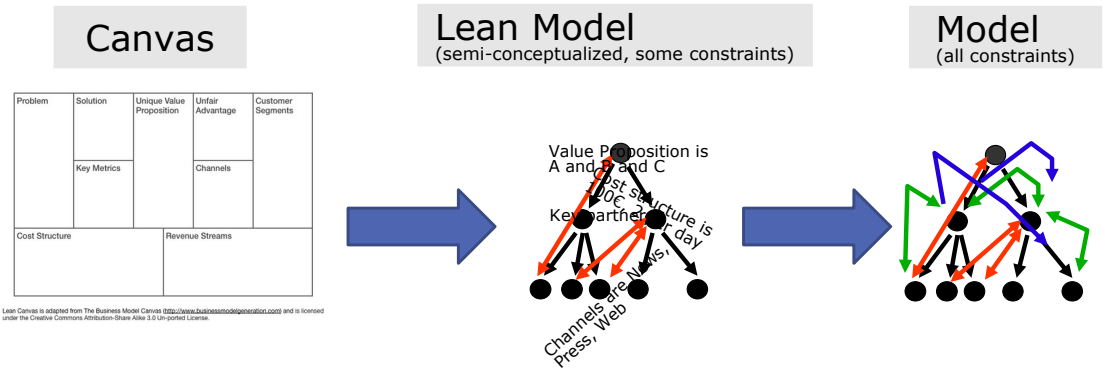
3 Worlds to Bridge



31.2 World 1: Value Engineering with Canvas Trees as Lightweight Collaboration Tools

Canvases as Lean Models

- A **canvas** is a collaborative frontend for a model, in which sticky notes demarcate the formal content from the informal text.
- A **lean (formal) model** is a *partial*, *semi-conceptualized model*, an active document with informal and conceptualized content, fulfilling *some constraints* of a set of constraints C.
  - A **model** fulfills all of the constraints in C and has a full set of StickyNotes.
- **Lean modeling is an agile conceptualization process:**
  - Canvas -> Lean Model -> fully conceptualized Model

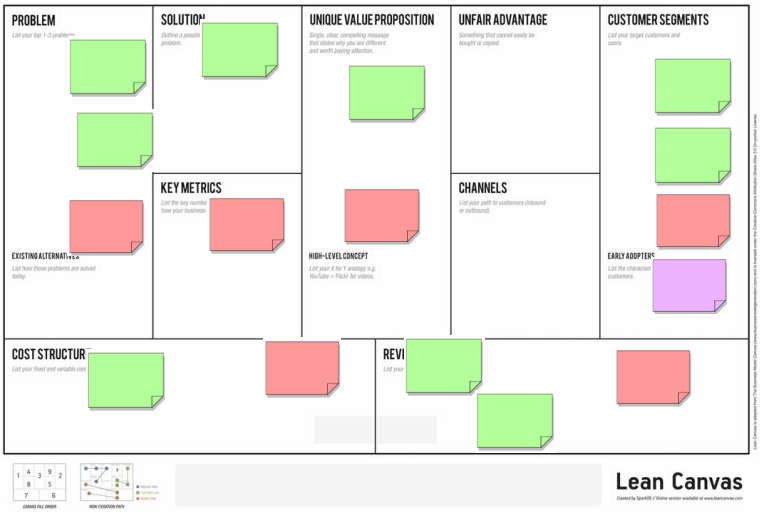


A Canvas Trees (Nested Canvas)

- Def.: A ***canvas tree (nested canvas, deep canvas)*** is a whole-part tree with a tree of subcanvases
  - Canvases and fields recursively alternate (intertwine)
  - Every canvas forms a tuple of fields
  - Sticky notes attach text to the fields, and are related to by AND, IOR, XOR
  - A subcanvas can be attached to a field
  - Constraints constrain the content of the canvas fields
- **Subcanvases** form children
  - Grammars of nested canvases are united (grammar composition)
- The **fill order** of the canvas defines a phase structure on the link tree
  - Metrics on advancement (hierarchical wavefront progress)

What are the Relations between Sticky Notes?

- Often different colors express XOR business cases: What is the relationship between green, red and pink sticky notes?

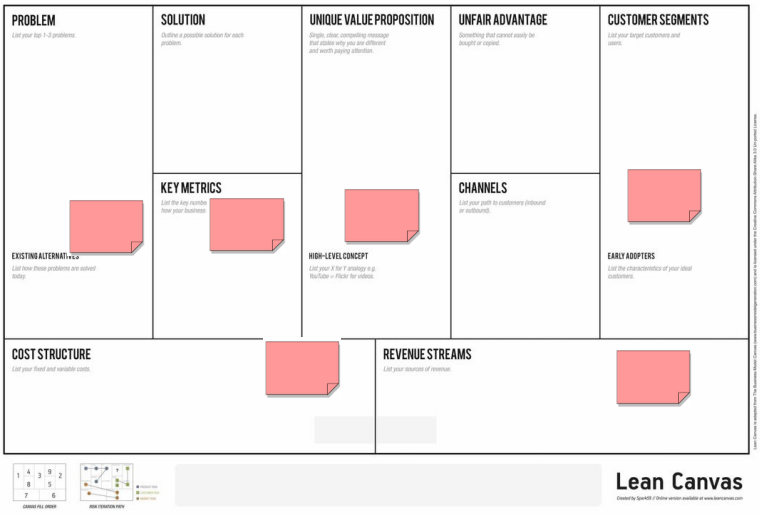


Exp.: XOR Can Be Used to Split a Canvas

- The green business case is the most complete one

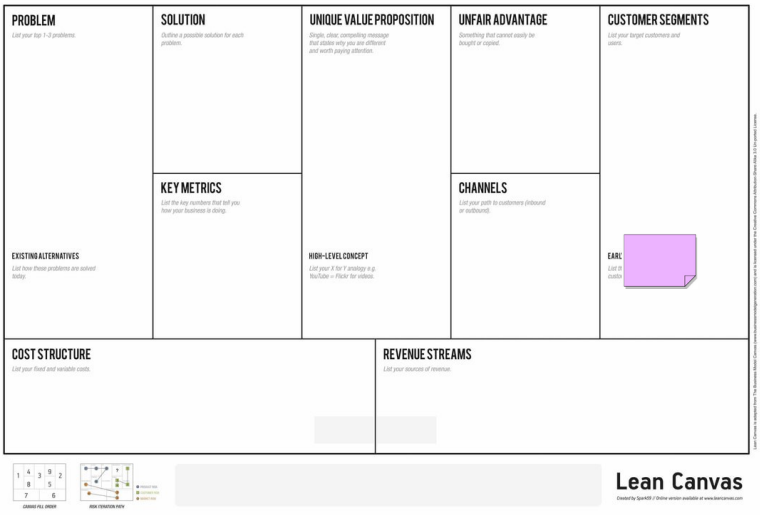
Exp.: XOR Can Be Used to Split a Canvas

- The red business case is less complete



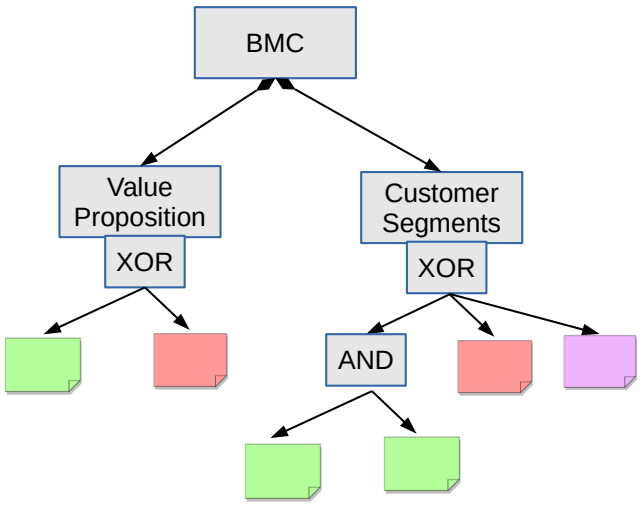
Exp.: XOR Can Be Used to Split a Canvas

- The pink customer segment must be equipped with much more hypotheses



BMC as Lean Canvas with XOR/IOR/AND Constructors, with Sticky Notes as Leafs

It is important to classify Sticky Notes relations into the constructors XOR (different color), IOR (striped) or AND (same color).

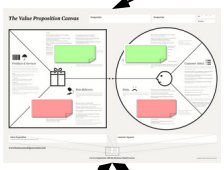


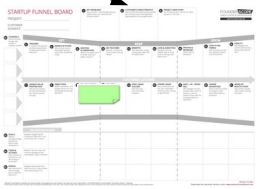
Canvas Trees, with Sticky Notes as Leaves

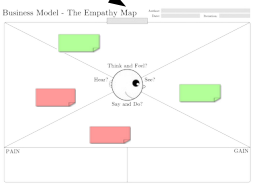
https://dschool.stanford.edu/wp-content/themes/dschool/method-cards/empathy-map.pdf

https://jobs-to-be-done.com/the-jobs-to-be-done-canvas-f3f784ad6270

A lean canvas tree can extend over several canvases.

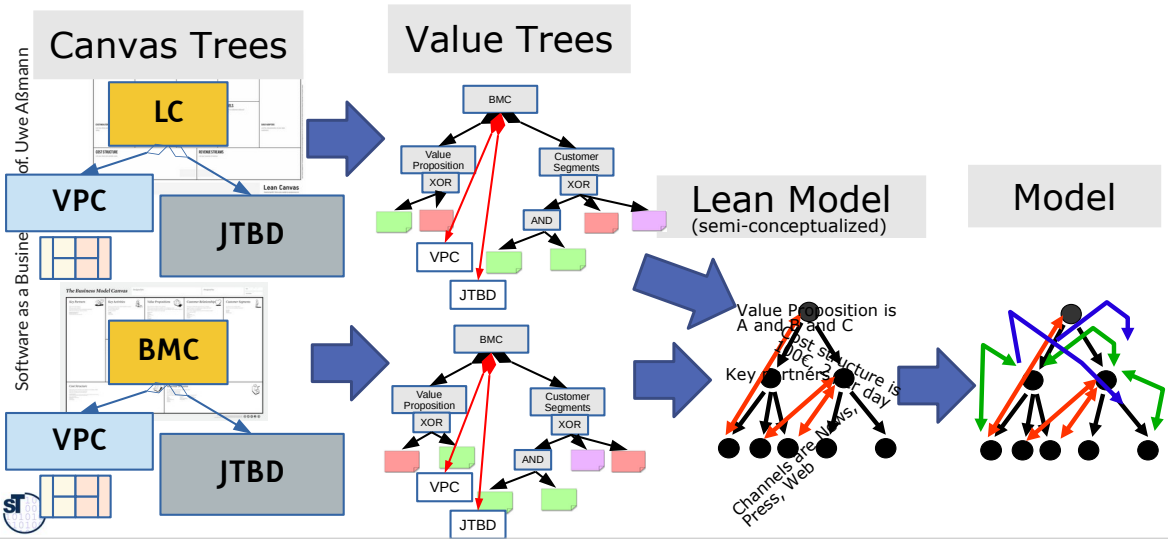






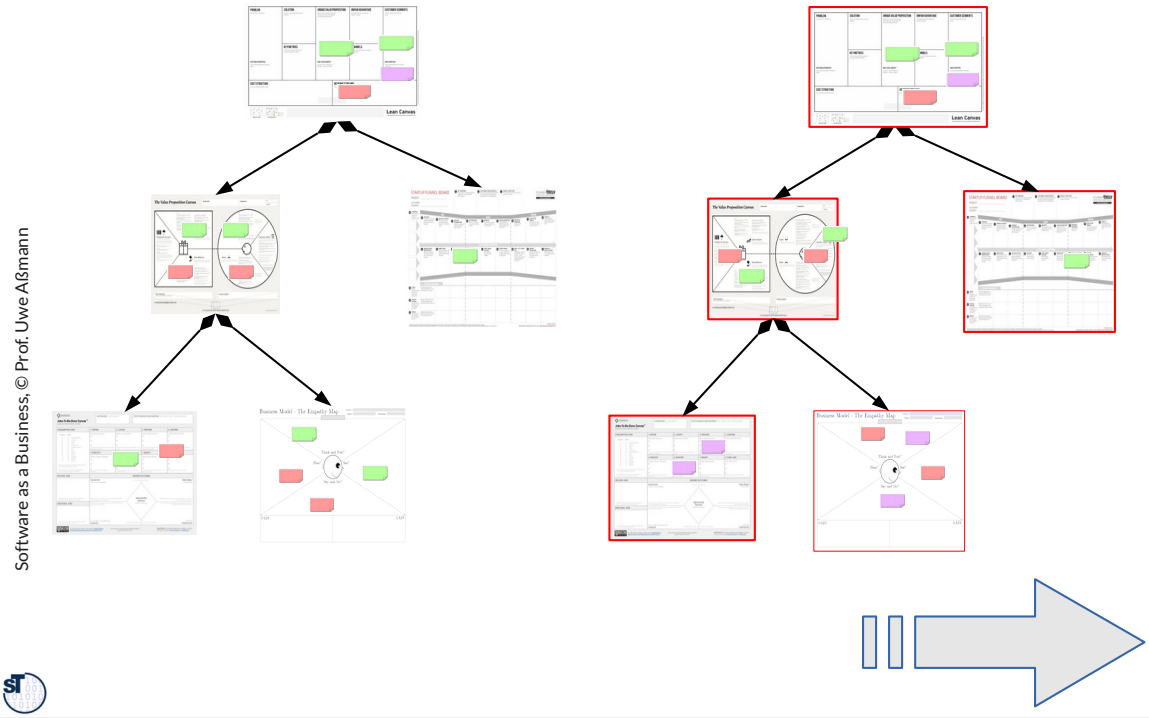
Parallelly Edited Lean Models can be Merged to Get a More Mature Lean Model

- A **canvas tree family** is a set of parallelly edited nested canvas, which can be merged into a lean model by unifying the fields
- **Conceptualization Process:**
  - CanvasTree -> Value Tree -> Lean Model -> fully conceptualized Model

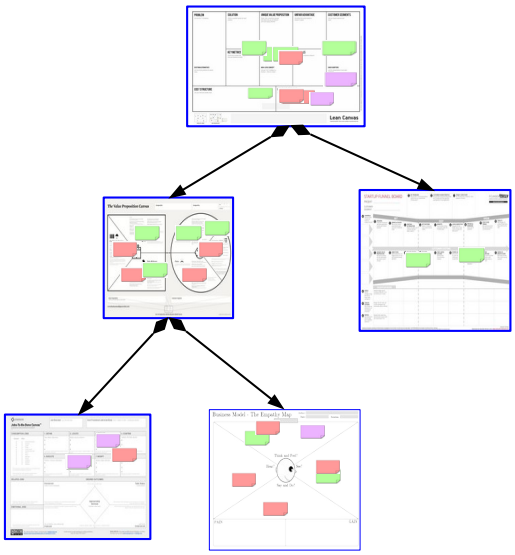


Merging of Canvas Trees, with Sticky Notes as Leaves

https://dschool.stanford.edu/wp-content/themes/dschool/method-cards/empathy-map.pdf



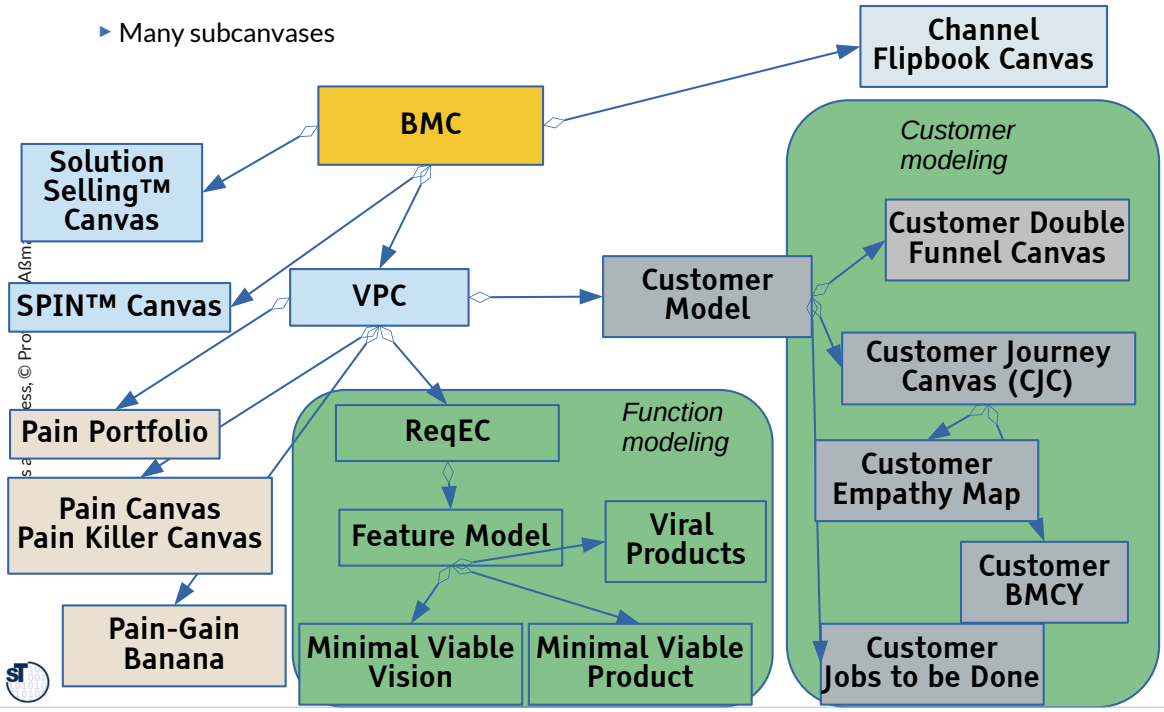
Merge Result of Two Canvas Trees (More Complete)



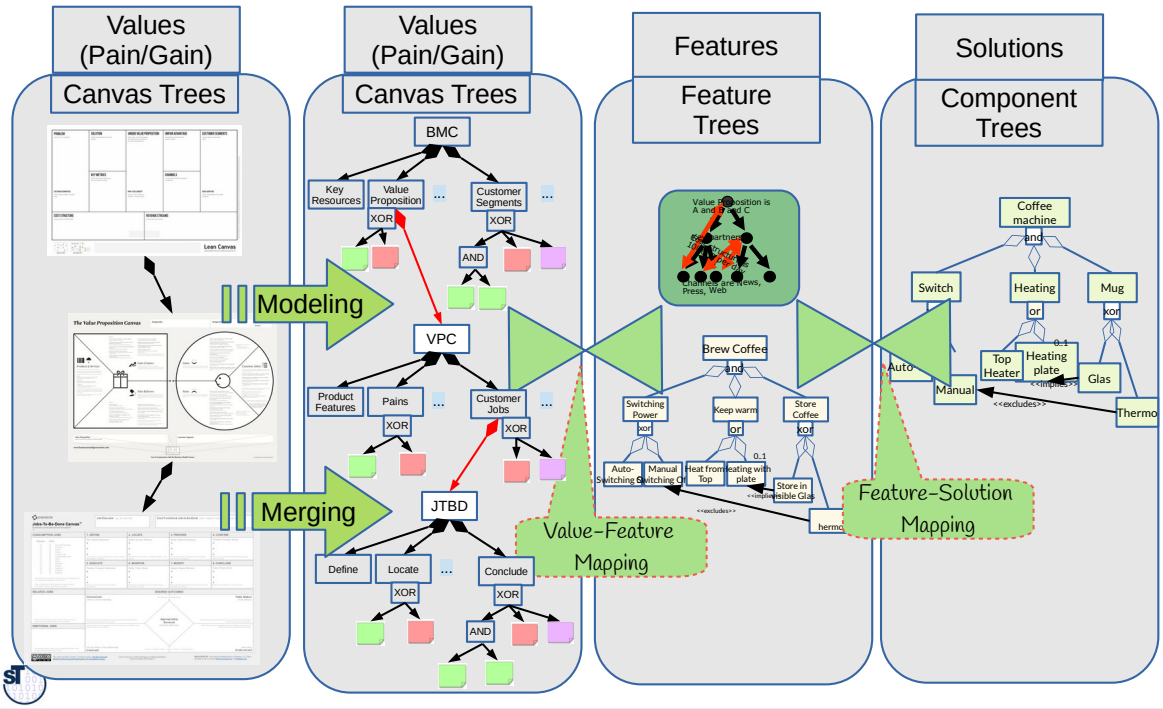
The Nested BMC adds ReqEC and Feature Models as Subcanvas

Q: How Do We Arrive at a Feature Model that is NON-Lean (Complete)?

- Many subcanvases



3 Worlds to Bridge (In Model Tree Syntax)



31.2.1 Lean (Formal) Modeling with Lean Canvas Trees and Canvas Cactus

Canvas trees are *nested hierarchies*.
How do we specify canvas trees?

Canvases are often *incomplete*.
How do we specify incomplete canvas trees?

Canvas „paths“ in a canvas tree:

- BMC—ValueProposition->VPC—Customer Jobs->JobsToBeDone
- BMC—ChannelI->ChannelFlipbookCanvas
- LeanCanvas--UnfairAdvantage->StraticPositioningCanvas

How Do We Describe Lean Canvas Trees (Canvas Models)? (Why Grammars for Typing Canvases?)

Metamodels: Type graphs typing graphs

- Neighbors of nodes can be collections
- Partition constructor (over ALL neighbors)
- Simple recursion (by circular references)
- No complex tree structures
- Constraints

Grammars: Recursive typing rules typing deep trees and graphs

- Attributes and attribution functions
- XOR constructor (over 2 or all neighbors)
- Complex recursion (left, right, alternating, recursion schemes)
- Arbitrary number of Sticky Notes can be modeled flexibly
- Name analysis looks up the meaning of names by attribution functions
- Constraints

Schemas for Flat Canvases as Grammars for Lean Models

- A **canvas** is a structured questionnaire for collaborative development. It can be represented as a **tree-shaped model with constraints:**
- Canvas structure:
  - Canvas left side vs. right side
  - Left part, right part, upper, lower part
  - Canvas fields with sticky text notes, Canvas questions or answers
- Constraints:
  - Inter-field references with inter-field constraints
  - Intra-field constraints
  - Canvas fill order (partial order) on the tree nodes
  - Subcanvases

- Problem: Canvases are *incomplete*; grammars describe *complete* sentences of a language

- Def.: A ***lean model*** of a metamodel or grammar is an incomplete model that violates
  - *structural constraints*, but can be completed to a valid structure (structural wellformedness)
  - *wellformedness constraints*, but can be completed to a valid wellformed model (global wellformedness)
- A ***lean sentence*** of a grammar is a lean model of the grammar.
- A ***lean tree*** of a tree grammar is a lean model of the tree grammar.
- A ***part grammar*** is a tree grammar.
- A ***canvas tree*** is a lean tree of a part grammar.

Overview of Tree Grammars

Sentence	*Part types*	Grammar type
Texts	Ordering of things	String Grammar
Trees	Whole and ordered parts	Tree Grammar, Regular Tree Grammar (RTG)
Link Trees (XML, JSON, EMF)	Whole and ordered parts with references but not sharing	Reference Tree Grammar
Attributed Link Trees	With additional attribution functions	**Reference Attribute Grammars (RAG)**
Shared Trees (Dags) with attributions	Whole-part trees with shared parts	Multiset Grammars (MSG)
Shared Trees (Dags) with attributions and constraints	Whole-part trees with links and constraints	**Constraint Multiset (Attributed) Grammars (CMAG)**
Shared Trees (Dags) with complex collection-constructors and groupings	Whole-part trees, with complex constructors with links and constraints	**Constraint Part (Attributed) Grammar (CPAG)**

Constraint Multiset (Attribution) Grammars

- A schema for a canvas can be described by a *or Constraint Multiset Attributed Grammar, CMAG)* [Marriott] describing Whole-and-Part relationships without or with sharing (whole-part structural constraints) in a simple part rule sublanguage
  - Rules are written by <Nonterminal> ::= <RightPart> .
  - And **global constraints (invariants)** describing wellformedness conditions for the hierarchic structure.

```
%% Example Part Grammar for BMC with several whole-part rules
%%%%%%%%%%%%%%%%%%%%%%%%
Grammar Canvas = {
    PartRules {
      Root Canvas ::= Field+.                    %% Repetition 1..* times
    }
}
%% Grammar rules may use XOR, REPT, and other constructors
Grammar Fields = {
    PartRules {
      Root Field ::= Notes
      Field ::= Canvas.          %% Field can be Notes XOR Canvas
      Notes ::= StickyNote:Note*.  %% Repetition 0..* times (REPT)
      Note ::= Question.
      Note ::= Answer.
      Note ::= Comment.          %% Note is Question XOR Answer note
    }
    Invariants {
      forall q:Question in Canvas: q has a:Answer   %% all questions in  elds must be answered
    }
}
```


Schemas for Canvases: Constraint Part Grammars

- We generalize CMAG to *Constraint Part Attribution Grammar (CPAG)* by generalizing the part rules:
  - Default constructor of a rule body is conjunction (AND), expressed by tuple notation and juxtaposition (.., ..)
  - Exclusive disjunction (XOR) is expressed by (..|..|..|..) alternative brackets
  - Inclusive disjunction (OR) is expressed by set notation { .., .. }
  - Several alternative rule bodies for one nonterminal are connected by inclusive disjunction (OR)


```
%% Example Part Grammar for BMC with several whole-part rules (no constraints yet)
%%%%%%%%%%%%%%%%%%%%%%
Grammar Canvas = {
    PartRules {
      Root Canvas ::= Field+.                    %% Repetition 1..* times
    }
}
Grammar Fields = {
    PartRules {
      Root Field ::= { Notes Canvas }.           %% Field can be Notes IOR Canvas
      Notes ::= StickyNote:Note*.          %% Repetition 0..* times
      Note ::= (Question | Answer | Comment). %% Note is Question XOR Answer note
    }
}
```

Constraints for Canvases

- A schema for a canvas can be described by a *Constraint Part Attribution Grammar (CPAG)* has **global constraints (invariants)** describing wellformedness conditions for the hierarchic structure. Examples in the constraint sublanguage:
  - `forall stickynote in CustomerRelations: exists stickynote2 in Channels;`
  - `exist a revenue;`
  - The partial fill order a set of inter-field constraints

Alternative language for grammars and constraints: EBNF, OCL, ZINC

```
%% Example Constraints in Part Grammar for BMC
%%%%%%%%%%%%%%%%%%%%%%%%
Grammar Fields = {
    PartRules {
      Root Field ::= (Name:ID, { Notes Canvas } ). %% Named Field can be Name AND (Notes OR Canvas)
      Notes ::= StickyNote:Note*.                  %% Repetition 0..* times
      Note ::= (Question | Answer | Comment).      %% Note is Question XOR Answer XOR Comment
    }
    Invariants {
      forall f:Field exists y:StickyNote, y in f.Notes.
      MUST forall n:Name.ID == (Customer Relationships | ValueProposition | … | Costs ).
        %%   elds must be given 1 out of 9 standard names
    }
}
```

VPC as Grammar with Constraints


```
Grammar ValuePropositionCanvas =  {
 import Fields
 PartRules {
  Root VPC ::= ( LeftPart, RightPart ).
  LeftPart          ::= ( GainCreator:Field, PainKiller:Field, ProductsAndServices:Field ).
  RightPart         ::= ( Gain:Field, Pain:Field, CustomerSituation:Field ).
 }
 Invariants {
  forall s:Gain.StickyNote*: exists y:StickyNote, y in GainCreator.StickyNote*.
  forall s:Pain.StickyNote*: exists y:StickyNote, y in PainKiller.StickyNote*.
  forall s:PainKiller.StickyNote*: exists y:StickyNote, y in ProductsAndServices.StickyNote*.
  forall s:GainCreator.StickyNote*: exists y:StickyNote, y in ProductsAndServices.StickyNote*.
 }
}
```

- Invariants:
  - Forall gains there must be a gain creator
  - Forall pains there must be a pain killer
  - Forall pain killers there should be a service or product
  - Forall gain creators there should be a service or product

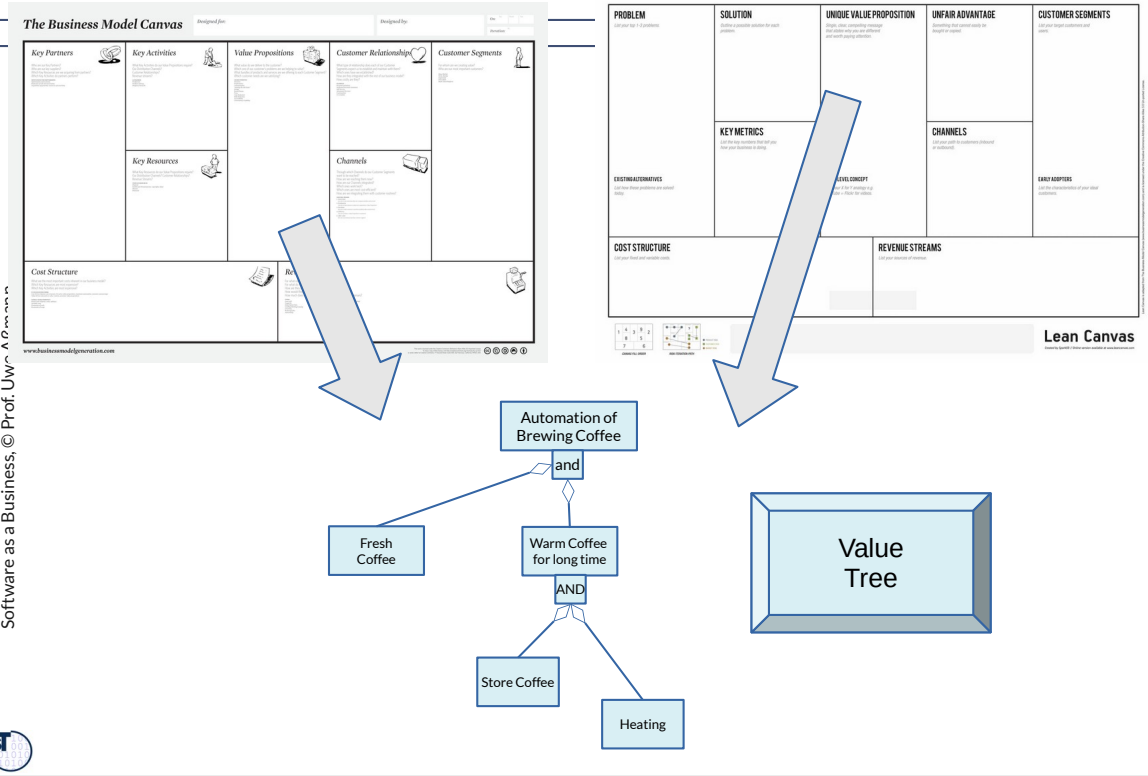
Schemas for Flat Canvases as Grammars

- Part Grammars (Constraint Multiset Grammar, CMG) can be
  - Imported (IMPORT G)
  - Unioned: all rule bodies with common nonterminals are OR-composed; invariants are OR-composed
  - Specialized: supergrammars S can be unioned with specialization extensions E to subclasses SC:
    - SC = S + E
    - S += E

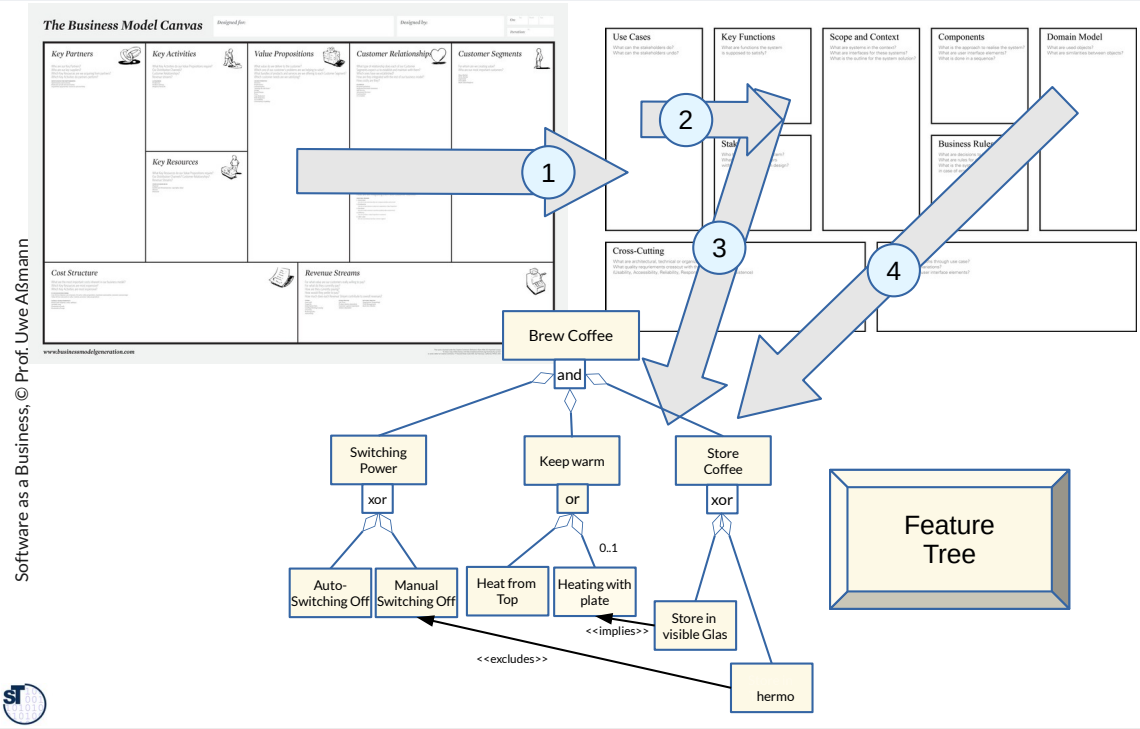

```
%% Example Grammar for BMC

Grammar BusinessModelCanvas = {
 import Canvas, Fields
 PartRules {
  Root BMC        ::= ( LeftPart, ValueProposition:Field, RightPart ).
  LeftPart        ::= ( KeyPartners:Field, KeyActivities:Field, KeyResources:Field, Costs:Field ).
  RightPart       ::= ( CustomerRelations:Field, Channels:Field, CustomerSegments:Field, Revenues:Field ).
 }
 Invariants {
  forall s:CustomerRelations.StickyNote*: exists y:StickyNote, y in Channels.StickyNote*.
  MUST exists r:StickyNote in Revenues.StickyNote*.
 }
}
```

31.2.2 From Canvas Trees to Value Trees

How to Derive Value Trees (from BMC and LC)

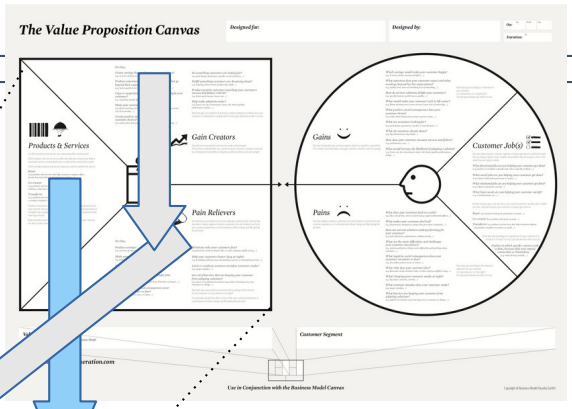


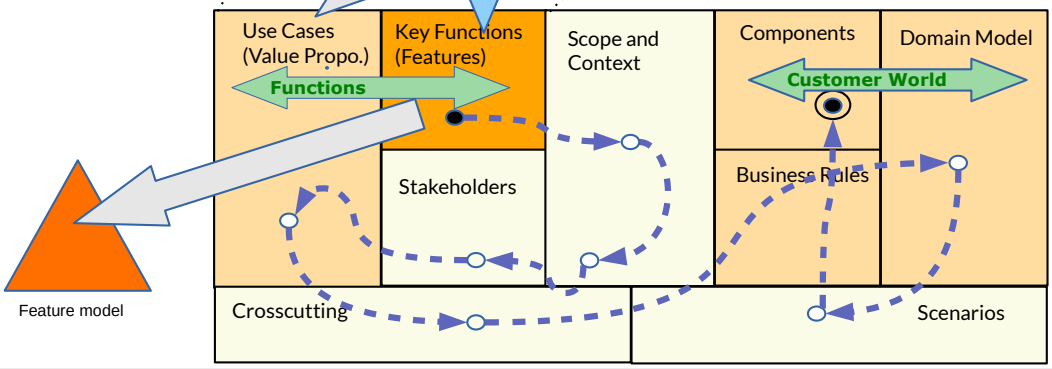
How to Find Features from Values (via ReqEC)



From VPC to ReqEC

- Remember Value Proposition Canvas
  - >ProductAndServiceFeatures
  - >Cain Creators
  - >Pain Killers
- Transfer left side fields to Requirements Engineering Canvas (ReqEC)
- Transfer field KeyFunctions to Feature Model





31.2.2 Validation of Formal Canvas Trees

Validating a Canvas

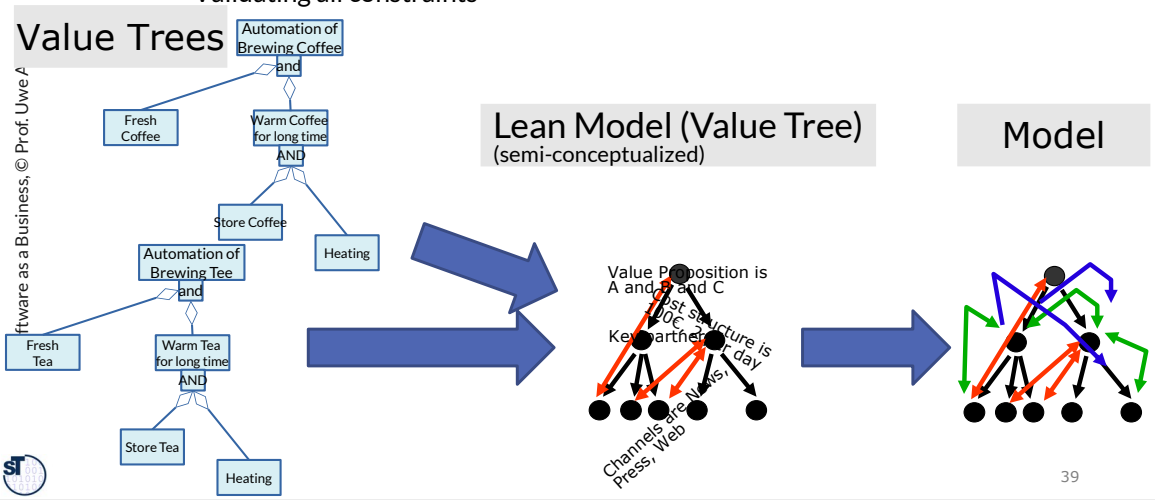
- Def.: A canvas is **structurally complete**, if all part rules are fulfilled.
- Def.: A canvas is called **well-formed**, if
  - All fields are being computed (filled)
  - All fields fulfill all constraints.
- Validation of wellformedness:
  - Parse the canvas with its sticky notes
  - Evaluate constraints in a constraint language
    - Constraint Part AG
    - OCL
    - Reference Attributed Grammar tool (www.jastadd.org)
    - or with an Multiset Constraint Grammar tool (Cider http://users.monash.edu/~berndm/cider/
- Wellformedness is usually computed by using standard and user-defined attribution functions

Standard Attributions in CPG

- If a tree is typed by a CPG, it has standard functions (standard attributions) which can be used by constraints:
- Node listing functions:
  - fun Root.downNodes(): All nodes reachable DOWN
  - fun Tree.nodes() == fun Root.downNodes(): All nodes in the tree
  - fun Tree.leafs(): all leaf nodes in the tree
  - For every nonterminal NT:
    - fun NT.nodes(): All nodes DOWN-reachable from NT
    - fun NT.children(): All nodes DOWN with distance 1
    - fun NT.sharedChildren(): All children shared by other parents
    - fun NT.nonsharedChildren(): All children not shared by other parents
- Edge listing functions:
  - fun Tree.links(): all links in the tree
  - fun NT.incomingLinks: all links pointing to nodes in NT.nodes()

Parallelly Edited Lean Models can be Merged

- A **lean model** can be merged with another lean model
- **Conceptualization Process:**
  - Canvas Tree * -> Value Tree -> Lean Model -> fully conceptualized Model
  - Assembling all constraints
  - Validating all constraints



31.2.3 Grading and Metrics on Formal Canvases Trees

Assessment in Canvases and Nodetypes in Canvas Trees

- **StickyNote dimension:** every field (tree node) can have a sticky note (Answer to a canvas question)
- **Commenting** is done by spanning up a ***comment dimension*** in a canvas tree
  - Every tree node can get a comment
- **Corresponding dimension:** Every node (e.g., sticky note or comment) can invoke a corresponding node in another field that has to be filled
  - When a sticky note invokes another sticky note
  - INVARIANT Exists s:StickyNote: corresponding(self, s)
- **Grading** is done by spanning up a *grading dimension* in a canvas tree
  - Every node can get a grade (green-yellow-red, 1-5, 1-10, 1-15)
  - The grading dimension defines grading functions for sticky notes in the fields
- **SWOT dimension:** every node can get a SWOT grading node: "how strong/weak/opportunity-like/trend-like is node?"
  - BMC-SWOT grading matrix canvas uses the SWOT grading dimension
  - LeanCanvas-SWOT uses SWOT grading dimension for LeanCanvas
- **Grading on nested canvases:** Grading is like commenting, but attributing a grade to a node. It defines the grading functions for all tree nodes of the nested canvas.

Examples of Attributes (Variables) of a Canvas Field (Node)

- `Node.Questions: → List(Question)` *// all questions of a field or note*
- `Node.SWOT: → List(SWOT)`
- Node.Comments: → List(Comment) *// all nodes in a canvas can be commented*
  - NumberOf *// all lists in nodes of a canvas can be counted*
- `Field.AllStickyNotes: → List(StickyNotes)`
- `Field.MissingStickyNodes: → List(empty Fields)`
- `Field.Grade:` */* The average of all sticky note grades */*
- `StickyNote.Grade:` */* the grading: e.g., red, yellow, green */*
- `StickyNode.SWOT.Strength.Grade:` */* Grade of SWOT */*
- `StickyNode.SWOT.Weakness.Grade:` */* Grade of SWOT */*
- `StickyNode.SWOT.Opportunity.Grade:` */* Grade of SWOT */*
- `StickyNode.SWOT.Trend.Grade:` */* Grade of SWOT */*
- `StickyNote.CorrespondingStickyNote: → List(Ref StickyNote)` */* corresponding sticky nodes or holes */*

Examples of Attributes of a Canvas Tree

- Canvas.subCanvases() /* Get all subcanvases */
- Canvas.subCanvases.count()
- Canvas.StickyNotes() /* get sticky note list */
- Canvas.CountStickyNotes() /* how many sticy notes */
- Canvas.Grade: /* *The average of all sticky note grades of all nodes* */

Every matrix analysis (SWOT, 7W, SCAMPER, ...)
creates a metric on the Canvas Tree

Thresholds for Canvas Tree Metrics

- Status of invariants is important for the ***maturity* of the canvas**

A **green** canvas tree fills all its variables
and fulfills all its invariants.

Only green canvas trees are models.

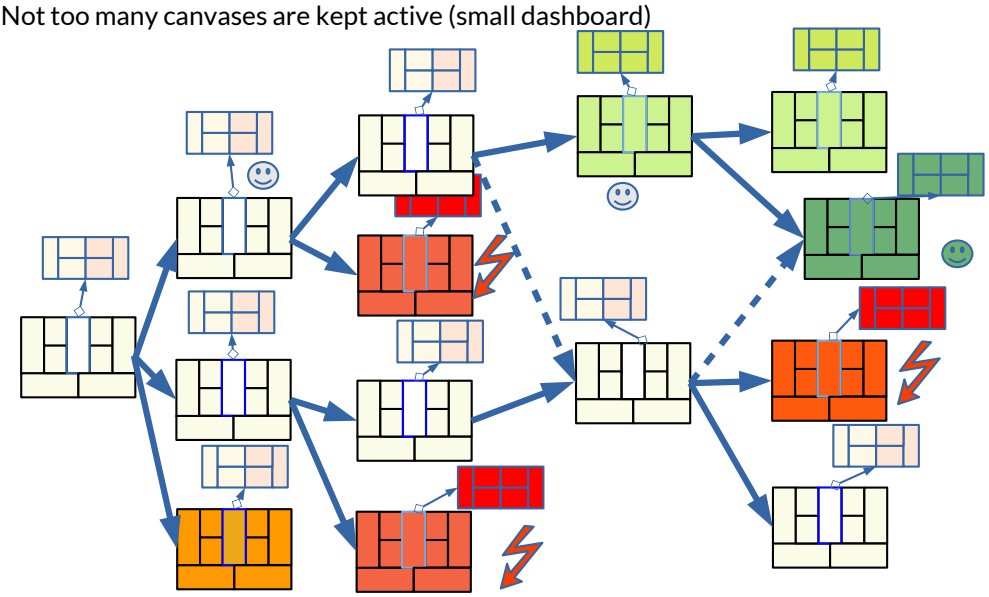
If a set of metric function on a nested canvas does not
fulfil their thresholds, or if not all invariants are fulfilled,
we call the canvas tree ***orange.***

A **red** canvas tree does not fulfill all its **MUST invariants.**

31.2.4 The Formal Canvas Tree Cactus as Multimodel and its Metrics

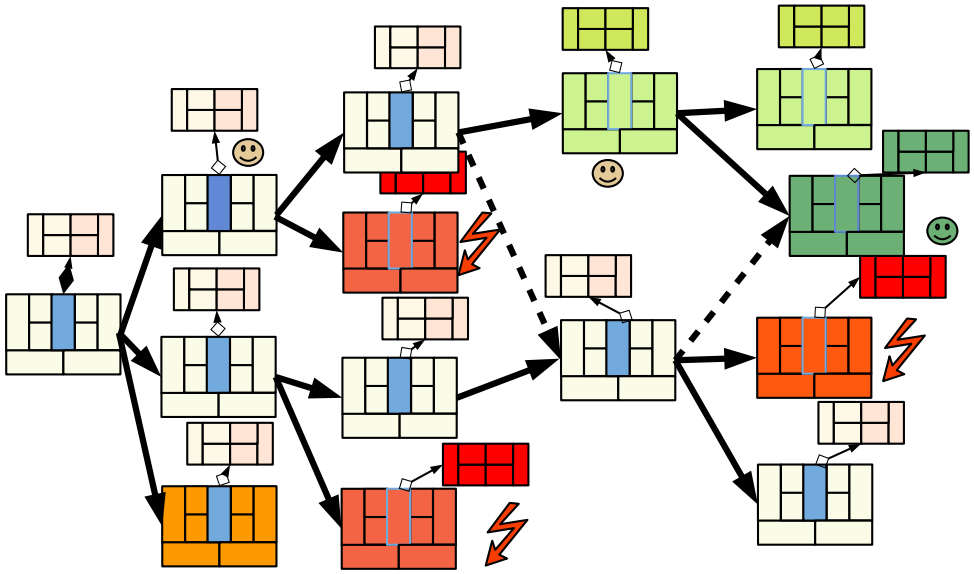
The Evolving BMC-VPC Canvas Cactus (extended)

- Growing a tree with side edges (link tree - cactus) out of a first version
  - Assess with red-yellow-green; choose a current “greenest” “champion”
- Every step tests **hypotheses** about the customer
- Not too many canvases are kept active (small dashboard)



The Evolving deep-BMC-VPC Canvas Tree Cactus (extended)

- Growing a tree with side edges (link tree - cactus) out of a first version
  - Assess with metrics and red-yellow-green; choose a current “greenest” “champion”
- Every step tests **hypotheses** about the customer and **changes metrics**
- Not too many canvases are kept active (small dashboard)



The Multimodel of Evolving Canvases

- A **multimodel** is a set of interrelated models, a **megamodel** describes a multimodel

- A **canvas tree cactus** is a multimodel of canvases, i.e., a link-tree-shaped multimodel of canvases
- **Canvas cactus evolution** evolves the multimodel with agile modeling
- The multimodel of canvases in a cactus is set of link trees
  - and can be analysed by constrained multiset grammar (CMG)
  - Wellformedness
  - Metrics
  - Constraints

31.3 From Values to Features - How to Derive the MVFS

(World 2 – Feature Engineering)

31.3.1 Variability Modeling in Feature Trees Variability Modeling with And-Or Trees

- Describing Feature Models as Constraint And-Or Trees

And-Or Trees as CPG


```
Grammar AndOrTree = {
 import Fields
 PartRules {
  Root Tree       ::= (Conjunction | Disjunction | ExclDisjunction) Child +
  Conjunction   ::= ‚AND‘
  Disjunction   ::= ‚OR‘ | ‚IOR‘
  ExclDisjunction     ::= ‚XOR‘
  Child ::= Tree Lower:Cardinality Upper:Cardinality
  Cardinality: ‚0‘ | ‚1‘ | ‚*‘
 }
}
```

For specification of Variation Points

- Every OR and XOR describes a *variation point*

And-Or Trees as CTG with Constraints


```
Grammar ConstraintAndOrTree =  {
  import AndOrTree
  PartRules {
    Source:Tree      ::= ( Constraint Target:^Tree )   %% from nodes, constraints may go out
        %% they are modeled as references to other tree nodes can go out
    Constraint ::= ( ‚implies‘ | ‚excludes‘ | ‚bidirectionally implies‘ | ‚bidirectionally excludes‘ )
  }
  Invariants {
    Invariant forall t:Target: t is_contained_in(Root.nodes) %% Target node must be IN tree
  }
}
```

- Invariants:
  - All references must point to target nodes within the tree

Feature Models as ConstraintAndOrTrees


```
Grammar FeatureModel =  {
  import ConstraintAndOrTree
  PartRules {
    Root ProductFamily:Tree ::=  %% Root of tree models a product family
    Feature:Tree ::=             %% Tree nodes model features of a product
    }
}
```

- Invariants:
  - All references must point to target nodes within the tree

Multi-Product Feature Models as ConstraintAndOrTrees

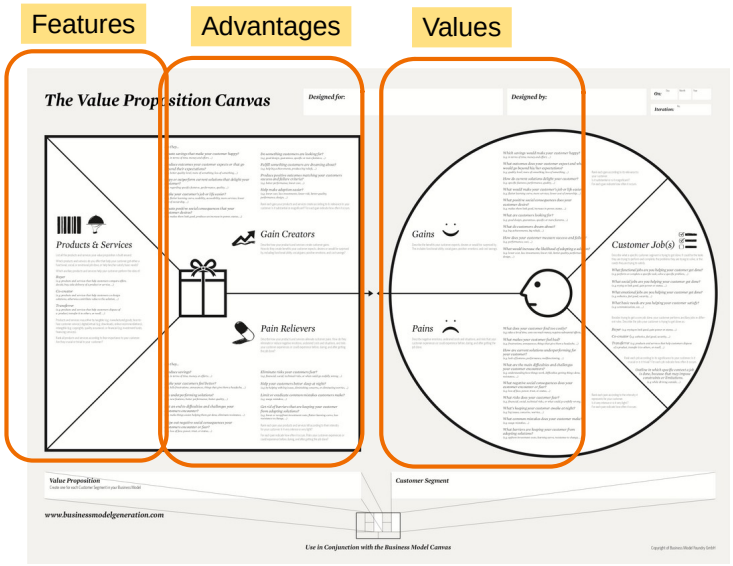

```
Grammar MultiProductFeatureModel =  {
  import ConstraintAndOrTree
  PartRules {
    Root MultiProductFeatureModel:MultiTree   ::= . %% Root of tree models a product family
    MultiTree ::= Tree +.
    Feature:Tree ::= .              %% Tree nodes model features of a product
    }
}
```


- The MultiProductFeatureModel has several roots (multi-hierarchy), and therefore describes several product families (Multi-SPL)
- Invariants:
  - All references must point to target nodes within the tree

31.3.2 From Canvas and Value Trees to Feature Trees

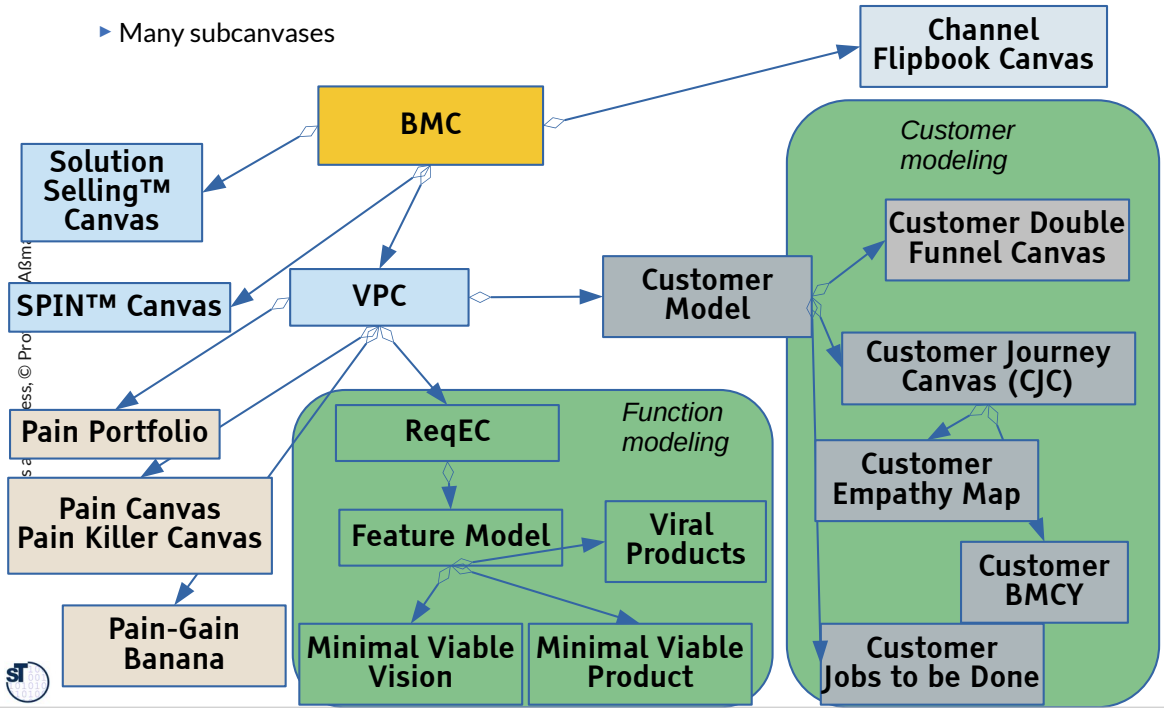
How Do We Derive a Minimal Viable Feature Set Systematically?

- Remember Value Proposition Canvas
- Features of Products and Services
  - are derived from right to left from gains and pains (Values)
- Products and services can be modeled by feature models!
- How to develop a feature model from the VPC?

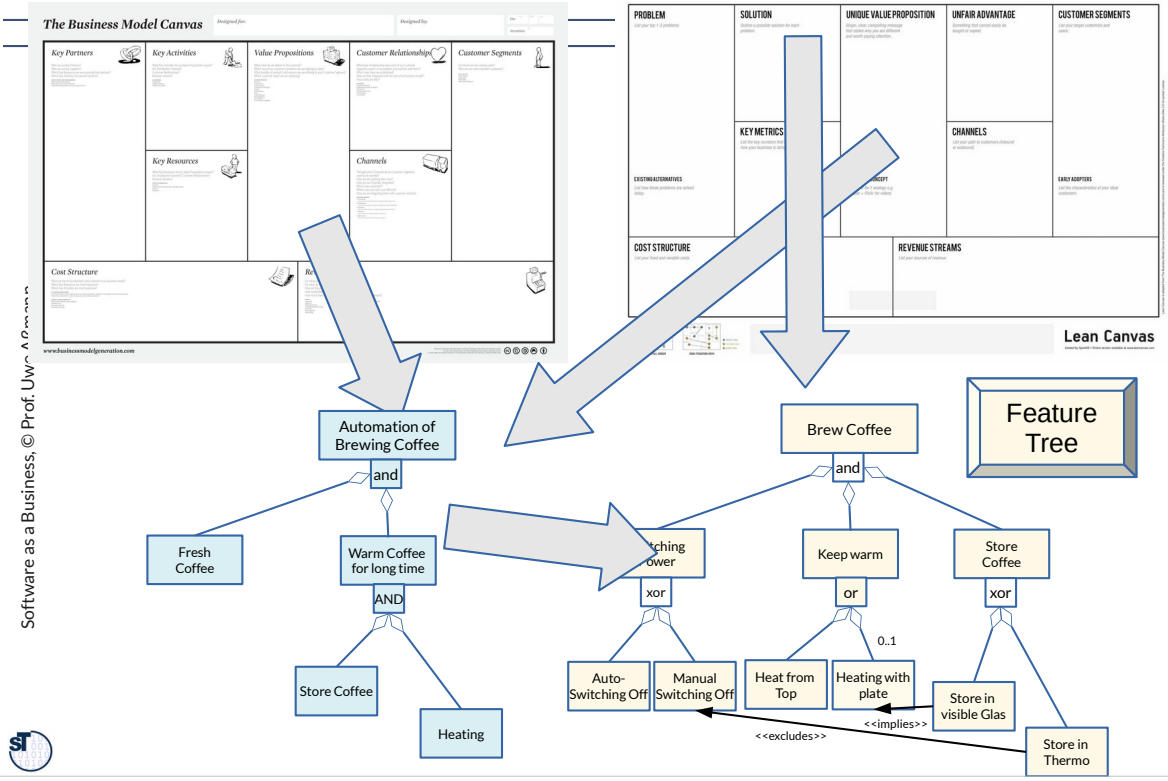


The Nested BMC adds ReqEC and Feature Models as Subcanvas

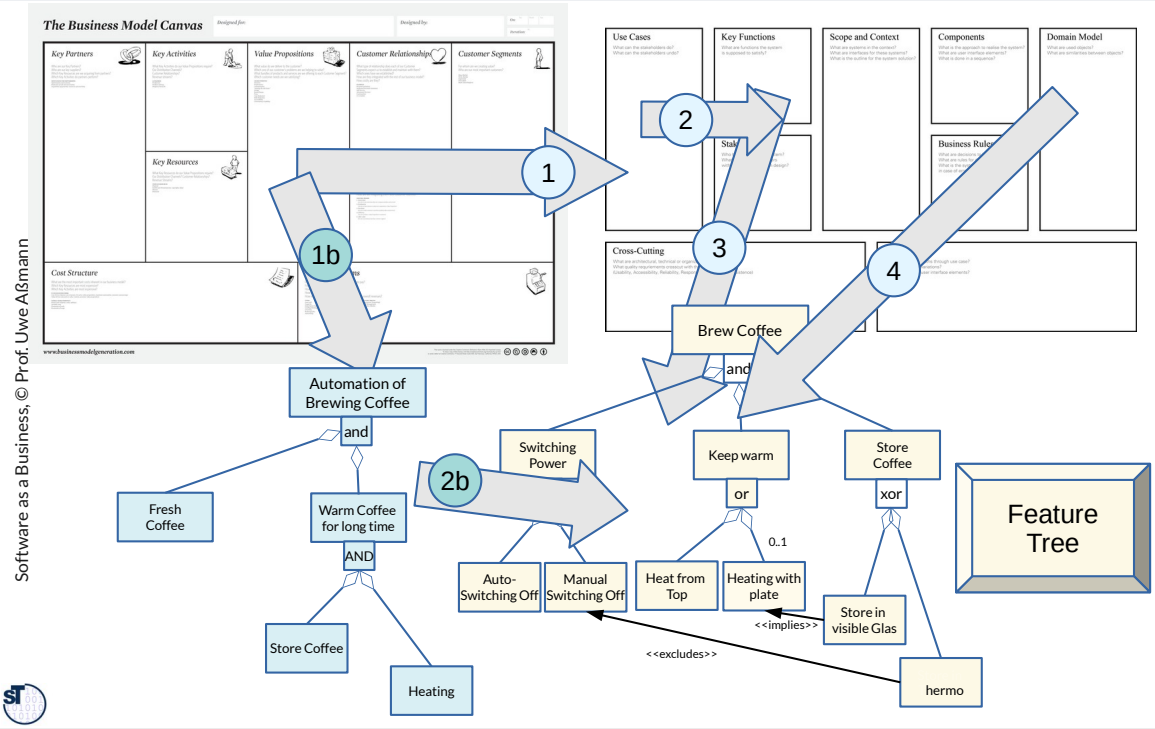
- Many subcanvases



How to Find Features (from BMC and LC)

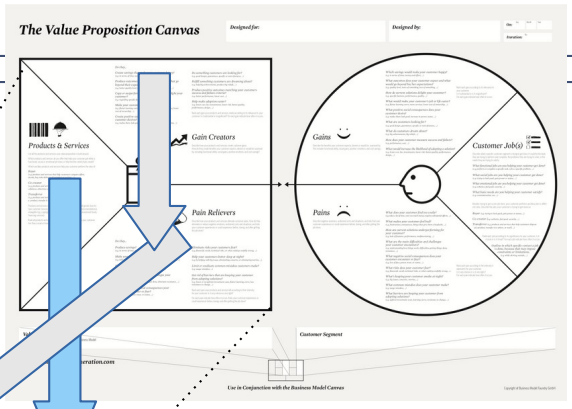


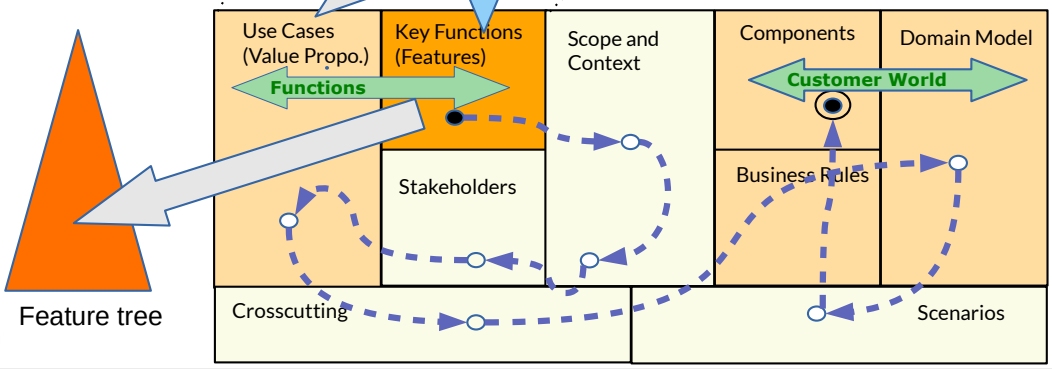
How to Find Features (via ReqEC)



From VPC to ReqEC

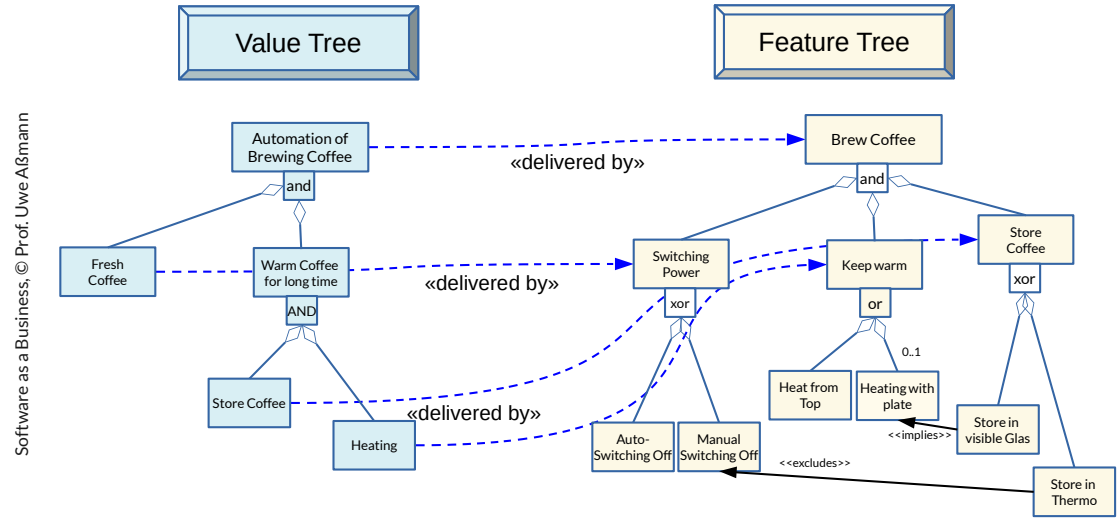
- Remember Value Proposition Canvas
  - >ProductAndServiceFeatures
  - >Cain Creators
  - >Pain Killers
- Transfer left side fields to Requirements Engineering Canvas (ReqEC)
- Transfer field KeyFunctions to Feature Model





Feature Models as well as Value Trees Obey Grammars

- Between two hierarchies, a *value-feature mapping* can be drawn showing *which value is delivered by which feature*

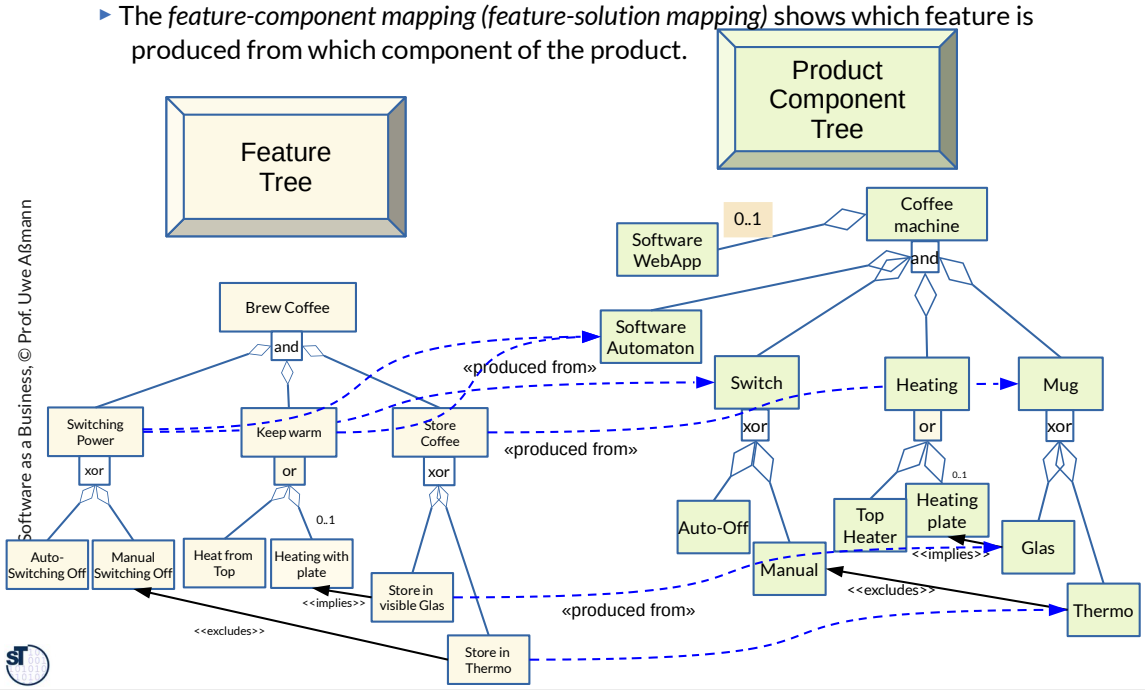


31.4 Bridging Features to Solutions (World 3 – Solution engineering)

- Describing Feature Models as Constraint And-Or Trees

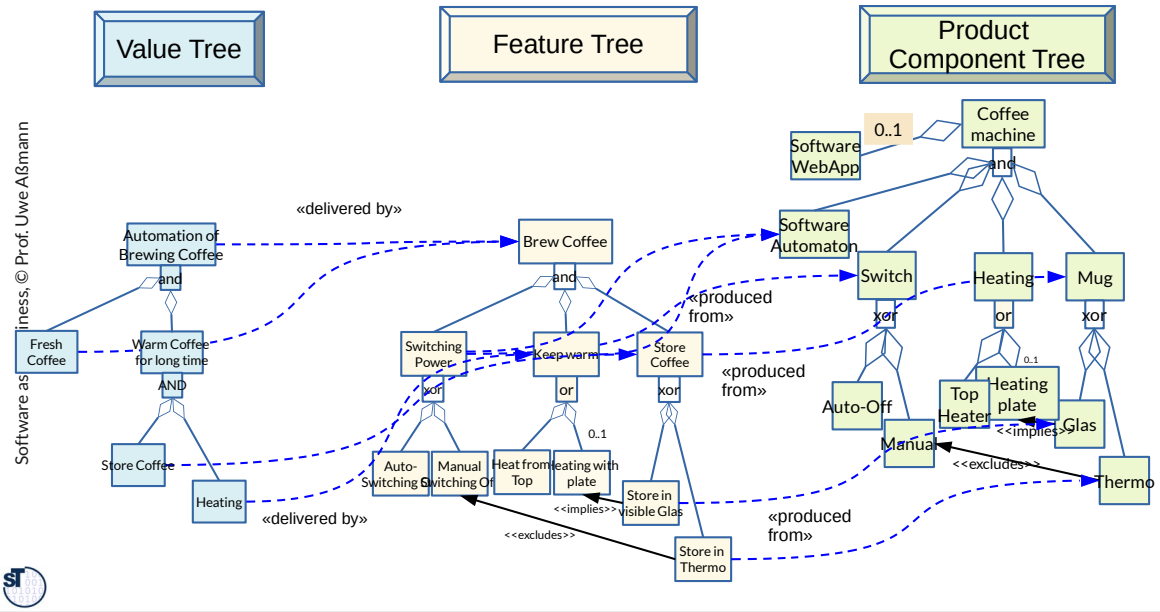
From Feature Trees to Product Component Trees

- The *feature-component mapping (feature-solution mapping)* shows which feature is produced from which component of the product.



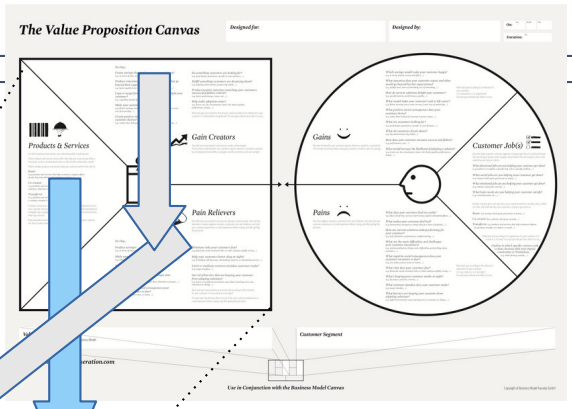
Bridging three Worlds: From Value Trees via Feature Trees to Product Component Trees

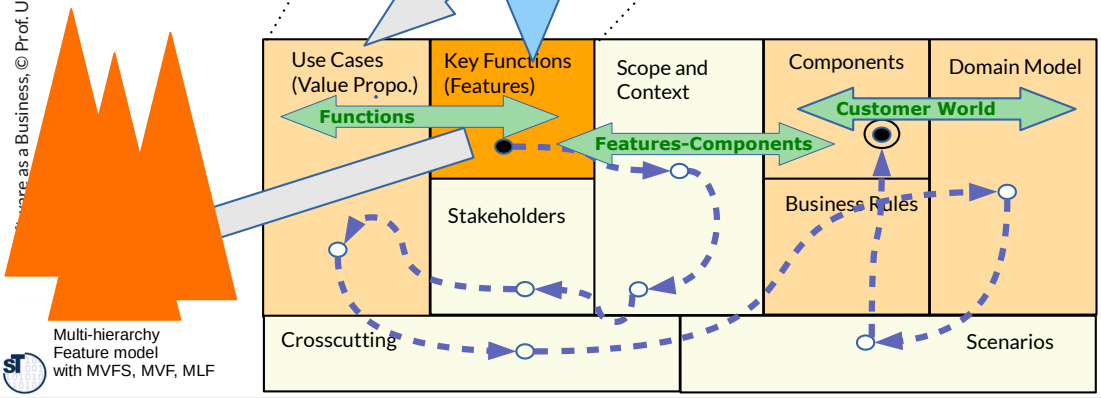
- Values can be traced via features to components of the product



How Do We Derive a Minimal Viable Feature Set Systematically?

- Remember Value Proposition Canvas
  - >ProductAndServiceFeatures
  - >Cain Creators
  - >Pain Killers
- Transfer left side fields to Requirements Engineering Canvas (ReqEC)
- Transfer field KeyFunctions to Feature Model

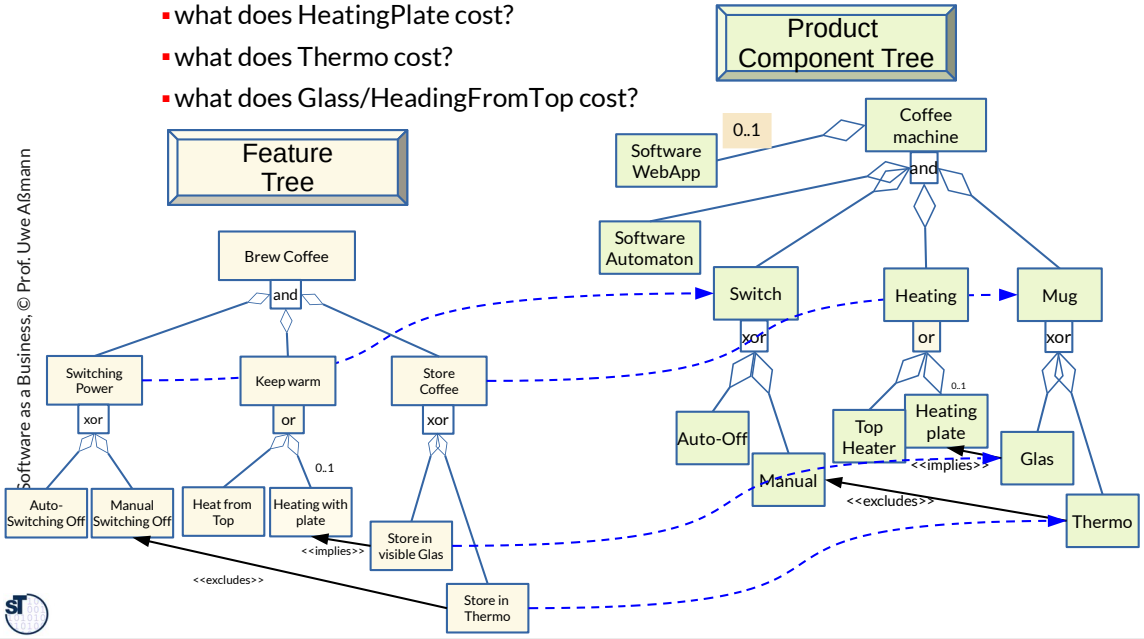




Multi-hierarchy Feature model with MVFS, MVF, MLF

How Can We Select a MVFS and MVP from the Feature Model?

- The MVP is a minimal-development-cost model: *What does a Product Component tree cost to develop, if a feature configuration is selected?*
  - what does HeatingPlate cost?
  - what does Thermo cost?
  - what does Glass/HeadingFromTop cost?



How To Compute the Cost of Features and MVFS in a Lean Model

The cost of a feature is the accumulated cost of all related subproducts.

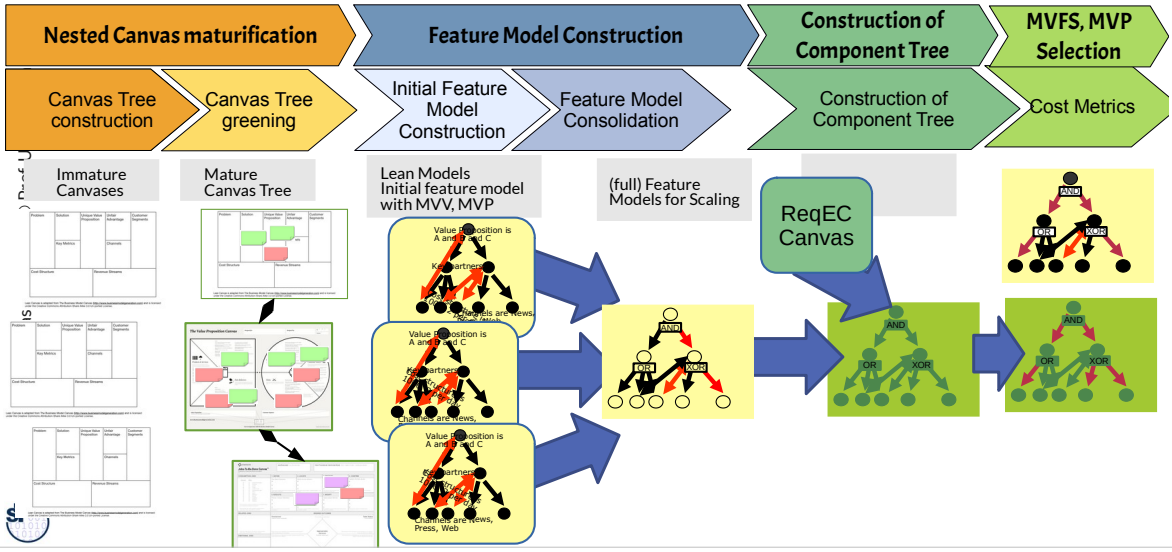
- Feature.Cost() → Integer is a function on the feature tree looking up all selected subfeatures, relates them to their subproducts and accumulates their costs (**the feature cost metrics)**

The MVFS is the feature set with the minimal cost.

The MVP is the realization of the MVFS.

Overview of MVP Construction

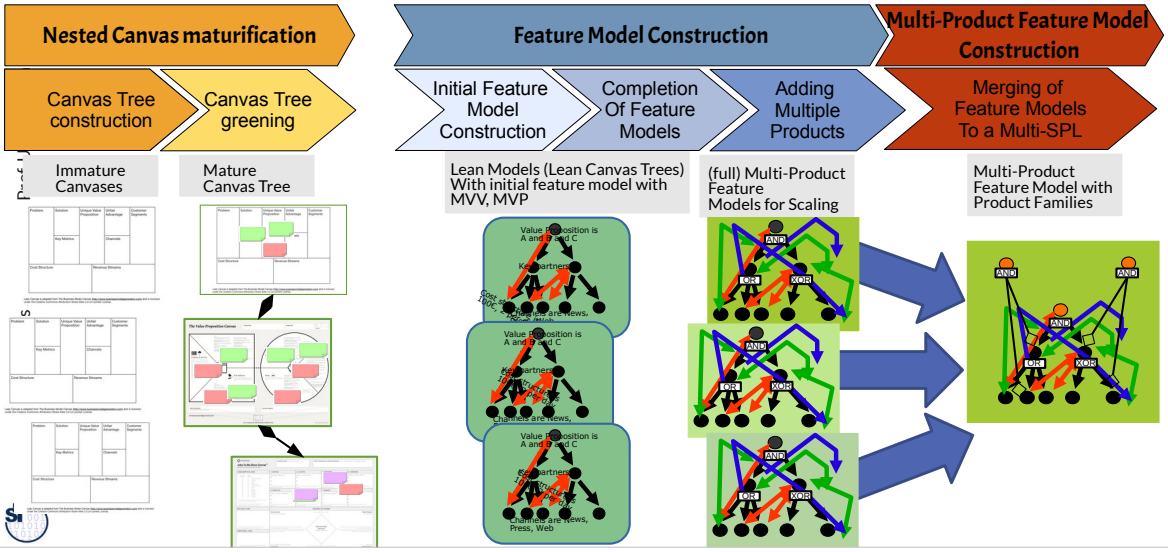
- **Conceptualization Process:**
- Canvas Trees → Value Trees → Feature Trees → Product Component Trees → Multi-Feature Model



31.6. From Feature Models (FM) and Multi-Product Feature Models (MPFM) to Software Product Lines (SPL)

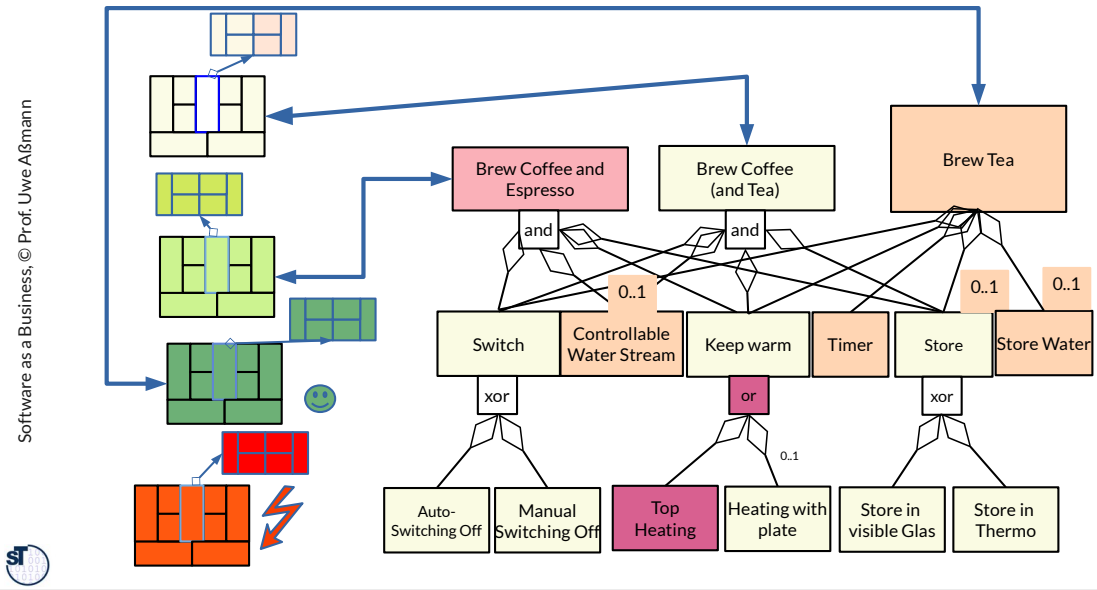
Multiple Canvases to Multi-Product Feature Model of Multi-SPL

- **Conceptualization Process:**
- Canvas Trees → Value Trees → Feature Trees → Multi-Product Feature Model



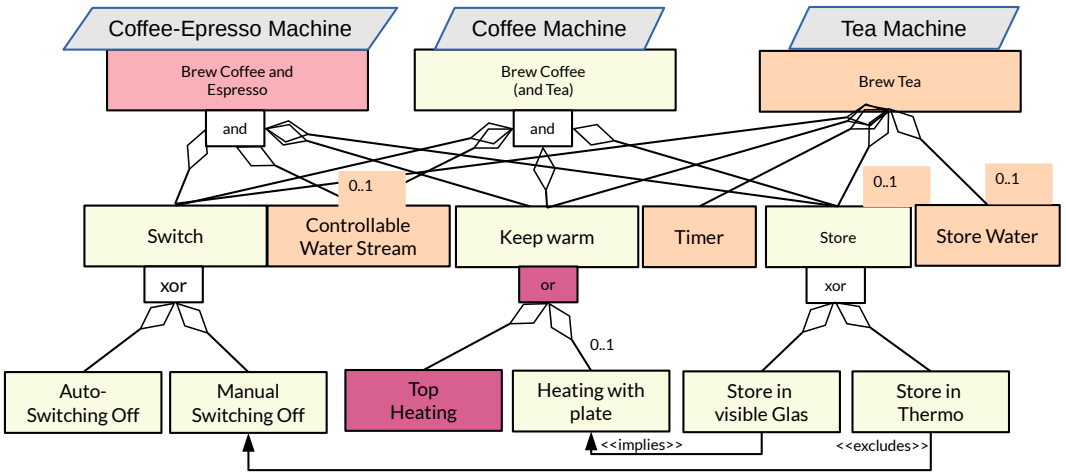
Multi-Feature Product Model of a Product Line

- Combined with a multi-feature model, the green canvases document all products of your product line



Remember: Multi-Feature Model (of Product Line)

- Variation adds 2 new products (Tea machine, coffee+pad-espresso machine)
- CoffeeMachine with enriched feature set
- Feature model may become too complex → refactoring necessary



The End

- More on modeling, lean modeling, and megamodeling in the course
  - "Model-Driven Software Development in Technical Spaces (MOST)" in WS

- Why do we need a grammar to model Canvases?
- Explain the concept of a Constraint Part Grammar (Constraint Tree Grammar (CTG) vs Constraint Multiset Grammar (CMG)).
- Explain why a canvas is an instance of a CPG.
  - Which role do invariants play?
  - Which role do filling functions play?
  - Can the user execute / simulate a filling function?
- What is the difference of a value, a feature, and a component of the product?
- How do you compute the costs of an MVP?

31.6. Context-based Feature Models (CFM) and Software Product Lines (CSPL)