Role of Flanders Make

1. Look forward, follow trends and define research roadmaps

2. Anticipate, help formulate and tackle challenges and opportunities, to inspire companies

3. Strengthen companies’ competences through a dynamic, collaborative innovation ecosystem

4. Create & transfer knowledge

5. Connect companies, research institutions and knowledge centres
Innovation through collaboration

Universities lay the foundation, develop new ideas & technologies

Flanders Make involves partners and offers knowledge for further development

The product reaches technology and market readiness

From lab to industrial environment

From technology leaders to early adopters

Competence/technology development

Involvement of additional companies

Universities

Industry

Valorisation Partner

Flanders Make

Innovation through collaboration
FLANDERS MAKE MEMBER COMPANY ECOSYSTEM

2014: #56 — Number of Members — 2019: #134

[Logos of various companies]
3 co-creation centres and labs at the 5 Flemish Universities

€ 60 mln turnover
600 highly specialised researchers

Highly specialised researchers
3 co-creation centres

Co-creation Centre
Vehicle development
Lommel
4200 m²

Co-creation Centre
Machine development
Leuven
3500 m²

Co-creation Centre
Customised production - Industry 4.0
Kortrijk
6000 m²
Fields of application

1. Production
2. Machines
3. Vehicles
Research supporting the digitalisation of the manufacturing industry clustered around 4 competences

- Decision & Control
- Design & optimisation
- Motion products
- Flexible assembly
Design & optimisation

How and which technologies?

1. Simultaneous model-based design of hardware & software for smart systems
2. Model-based design of assembly systems enabling high product variability
3. Design that takes into account the (financial) impact of product lifecycle phases
4. Efficient design of product variants & product families

Methods and tools to manage design complexity & efficiency
Modelling Variability in Mechatronics Systems Design
Smart, interconnected products and production systems

Mechanical system

-Mechatronic system

Increase in components, functions, technologies & disciplines => higher COMPLEXITY

Daimler (press release 2015)
Mercedes-Benz E-class
100 mio lines of code
Customized products

One product

Product family

Mass customisation

Increase in variants => higher COMPLEXITY
Managing Variability

https://www.youtube.com/watch?v=nd4tPauouQk
Modeling Variability

Andrzej Wąsowski

IT UNIVERSITY OF COPENHAGEN

PROCESS AND SYSTEM MODELS GROUP
Drowning in Clone-And-Own
Opportunistic Reuse Does Not Work

► Common scenario:
   ■ version the code, reuse when *opportunity* appears
► If the file to be reused needs change, *copy* it
   ■ You *clone-and-own* it
► **Benefit** from **quickly** available functionality
► But have to test, debug, change and evolve the file *yourself*
► **Product specific** code grows
► Platform code **diminishes and degrades**
SUCCESSFUL REUSE IS

PROACTIVE

PLANNED

MANAGED
AGENDA

- SPL Method, Architecture
- Variability Implementation Spectrum
- Variability Abstraction: Feature Modeling
- Variability Modeling in Practice
- Variability Realization
Domain vs Application Processes

Pohl et. al. Software Product Line Engineering
A Simple Product Line Architecture

- Less product specific = more reuse: development/tests/debugging/build
- Model of *commonality* and *variability*.
- Scope under control. Explicit feature life cycle
Exploit commonality
Manage variability
AGENDA

- SPL Method, Architecture
- Variability Implementation Spectrum
- Variability Abstraction: Feature Modeling
- Variability Modeling in Practice
- Variability Realization
Spectrum of Variability Architectures
Stay as close to the left as possible

property & configuration files + build system
feature models + build system
feature models + product specific code
domain specific languages + code generation
frameworks + framework completion code
only product specific code (no reuse)

feature models + product specific code
feature models + build system
property & configuration files + build system
Implementation Technologies

- Variability abstraction: FMs, DSLs, or none
- Variability resolution:
  - XML property file
  - FM configuration
  - Domain specific model (DSM)
- Variability realization:
  - general purpose code
  - w/ variability techniques
  - code generators
  - model transformers
  - parts may use DSLs
  - etc.

```
# CPUFreq processor drivers
CONFIG_X86_PCC_CPUFREQ=m
CONFIG_X86_ACPI_CPUFREQ=y
CONFIG_X86_POWERNOW_K6=y
CONFIG_X86_POWERNOW_K7=y
CONFIG_X86_POWERNOW_K7_ACPI=y
CONFIG_X86_POWERNOW_K8=y
CONFIG_X86_GX_SUSPMOD=y
CONFIG_X86_SPEEDSTEP_CENTRINO=y
CONFIG_X86_SPEEDSTEP_CENTRINO_TABLE=y
CONFIG_X86_SPEEDSTEP_ICH=y
CONFIG_X86_SPEEDSTEP_SMI=y
CONFIG_X86_P4_CLOCKMOD=m
CONFIG_X86_CPUFREQ_NFORCE2=y
CONFIG_X86_LONGRUN=y
CONFIG_X86_LONGHAUL=y
CONFIG_X86_E_POWERSAVER=m
```
AGENDA

- SPL Method, Architecture
- Variability Implementation Spectrum
- Variability Abstraction: Feature Modeling
- Variability Modeling in Practice
- Variability Realization
Problem Space

Domain specific abstractions

Solution Space

Implementation-oriented abstractions

features

SMS notification on transaction

paid services

variation points

SMSLoggerAspect

Phone No. in data model

c

© Andrzej Wąsowski, IT University of Copenhagen

13
CVL Architecture for Dummies

The degree of coupling can be controlled by moving the mapping

Variability Abstraction

Variability Realization

Base (model)

Feature/Decision Models

Feature Mapping

Source Code

Feature Modeling (I)

**feature:**

a single variability increment in the problem domain (decision)

**variation point:**

a single variability increment in the solution space
Feature Modeling (II)

Example from Czarnecki’02

- **Hierarchy** constraints, for example:
  - manual requires transmission (each child node requires its parent node)
- **Groups** constraints: engine is electric or gas driven or both
- Not all constraints in hierarchy & groups, cross-tree constraints in text:
  - electric requires automatic
- Attributes are added like to classes (eg. engine volume)
Feature Modeling (III)

Configuration

- car
  - body
  - engine
  - transmission
  - pulls trailer
    - electric
    - gasoline
    - automatic
    - manual

- electric requires automatic

© Andrzej Wąsowski, IT University of Copenhagen
Feature Models (IV)
An example meta-model from Janota’08

meta-model (abstract syntax)

- Note a single generic kind of relations: **subfeature**
- No distinction between **kind-of** (inheritance) and **part-of** (containment), like class modeling does
- A characteristic feature of **configuration and constraint languages** (as opposed to structural modeling languages)
- Clafer (as a structural modeling language) supports the distinction, but so do other feature modeling languages
Feature Modeling and FODA

Feature Oriented Design and Analysis by Kang et al. 1990

- FODA succeeds for its simplicity
- Probably best intro in Czarnecki’s *Generative Programming* (Chpt. 4)
- 3950+ citations, never formally published
Feature Modeling vs Class Modeling

A feature model in Product Variant Master Notation (Hvam)

More on this: Bąk, Czarnecki. Wąsowski. Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled. SLE 2010

Above models from: Haug, Degn. Poulsen. Hvam. Creating a documentation system to support the development and maintenance of product configuration systems
Applications of Feature Models

Design & Management

- Domain modeling
- Product line scoping
- Product line management
- Code generation
- Driving build system
- Driving
- Testing

Development & Test
How To Build Feature Models?

Two strategies, but only one good :)  

**top-down**

- **Big-bang** adoption  
- Perform **careful domain analysis**  
- Document **concepts, abstractions and relations** between them in a FM

**bottom-up**

- Identify a **cloned component**  
- Find the **patches** that describe differences  
- Translate **diffs to variation points**  
- Organize **variation points into features**, and a hierarchy  
- Works well with **incremental adoption**  
- See SPLC07 paper by Danfoss

Hans Peter Jepsen, Jan Gaardsted Dall, Danilo Beuche. *Minimally Invasive Migration to Software Product Lines*. SPLC 2007
SPL Method, Architecture
Variability Implementation Spectrum
Variability Abstraction: Feature Modeling
Variability Modeling in Practice
Variability Realization
Variability Modeling is The Success Story of Modeling
A Laboratory Feature Model

keyless_entry → power_locks
A Healthy Wild Feature Model Cub
ToyBox project, 71 features

The Linux Kernel has 6-12K features, depending how you count! But maximum depth is 8, most leaves are at 4!

↓ this is the Linux kernel model fit to the slide width ↓

Is FODA special? Not really!

eCos configurator

- Linux kernel and eCos operating system use similar configurator, controlled by textual variability models
- Trees become unwieldy very fast
- Many tools used linearized trees, like above
- Nice trees are good for PowerPoint, whiteboard and brainstorming
It is easy to design a textual syntax

Kconfig of Linux kernel, designed by non-experts, with no tools
CDL of eCos, designed by non-experts, with no tools

```
#menuconfig MISC_FILESYSTEMS
bool "Miscellaneous filesystems"

if MISC_FILESYSTEMS
  config JFFS2_FS
    tristate "Journaling Flash File System" if MTD
    select CRC32 if MTD

  config JFFS2_FS_DEBUG
    int "JFFS2 Debug level (0=quiet, 2=noisy)"
    depends on JFFS2_FS
    default 0
    range 0 2
    help Debug verbosity of...

  config JFFS2_FS_WRITEBUFFER
    bool
    depends on JFFS2_FS
    default HAS_IOMEM

cdl_component MISC_FILESYSTEMS {
  display "Miscellaneous filesystems"
  flavor none
}

cdl_package CYGPKG_FS_JFFS2 {
  display "Journaling Flash File System"
  requires CYGPKG_CRC
  implements CYGINT_IO_FILEIO
  parent MISC_FILESYSTEMS
  active_if MTD

cdl_option CYGOPT_FS_JFFS2_DEBUG {
  display "Debug level"
  flavor data
  default_value 0
  legal_values 0 to 2
  define CONFIG_JFFS2_FS_DEBUG
  description "Debug verbosity of..."
}

cdl_option CYGOPT_FS_JFFS2_NAND {
  flavor bool
  define CONFIG_JFFS2_FS_WRITEBUFFER
  calculated HAS_IOMEM
}
```
CVL Architecture for Dummies

The degree of coupling can be controlled by moving the mapping

Variability Abstraction

Variability Realization

Base (model)

Feature/Decision Models

Feature Mapping

Source Code

CVL Architecture for Linux Junkies

Variability Abstraction
- KConfig files

Variability Realization
- KBuild files + CPP

Base (model)
- C-code

© Andrzej Wąsowski, IT University of Copenhagen
Commercial tools support **multiple views** of the same model

- Some vendors: Pure Systems (DE), Big Lever (US), most PLM tools
- Clafer also has the grid view
Are all variability models trees?

A Fire Alarm System

- detection zones
- alarm zones
- wiring
- three different structures
- Modeled as constrained class diagrams!
- Sometimes called topological variability

Berger. Stanciulescu. Øgaard. Haugen. Larsen. Wąsowski. To connect or not to connect: experiences from modeling topological variability. SPLC 2014

Fantechi. Topologically configurable systems as product families. SPLC 2013
Modeled using class diagrams in Papyrus

A model view showing two model hierarchies

A home grown configurator used to instantiate models
SPL Method, **Architecture**
Variability **Implementation Spectrum**
Variability Abstraction: **Feature Modeling**
Variability Modeling in **Practice**
Variability **Realization**
Connect Abstraction to Realization

Most of the school is about it :)

variability

abstraction

realization

variability

abstraction

build

system

configured against

conforms

core assets

platform

product

specific

assets

variability

resolution

variability

realization

© Andrzej Wąsowski, IT University of Copenhagen
Feature Models vs DSLs

- Feature models are **ready and simple** (no design effort, deep insight)
- DSL requires **design effort**, but rewards with more **expressiveness**
- Effort also translates to **maintenance**
- FM effort is offset by **existing feature modeling tools**
- DSL development effort is offset by **language workbenches**
Advice on Realization

[Stahl and Völter]

- Choose **functional** domain concepts as features / DSL concepts
- Start a **small** domain model and grow it **iteratively**
- Keep the **build automatic** at all times
- Generate/synthesize **legible code/models**
- We follow these principles in the Clafer tutorial on railway stations
AGENDA

- SPL Method, **Architecture**
- Variability **Implementation Spectrum**
- Variability Abstraction: **Feature Modeling**
- Variability Modeling in **Practice**
- Variability **Realization**
Interested to join our team? Find our open positions and apply at

https://jobs.flandersmake.be