

# Automated creation of KF rules

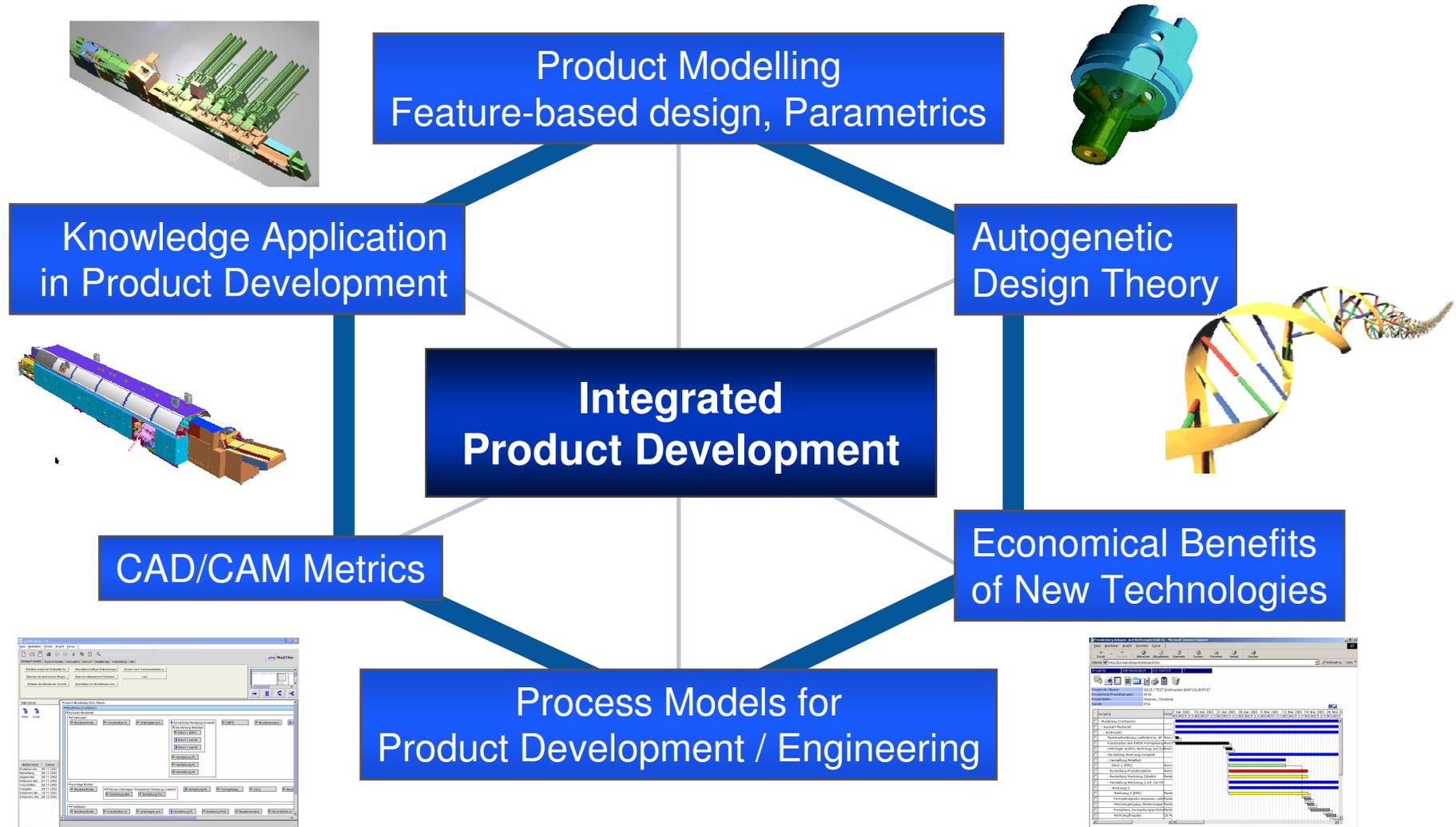
Dipl.-Ing. Guido Klette  
Prof.-Dr. Sándor Vajna  
Chair of Information Technologies in Mechanical Engineering  
Institute of Engineering Design  
Otto-von-Guericke University  
Magdeburg  
GERMANY



# Agenda

- research activities
- introduction
- process
- logging
- action
- rule creation
- example
- summary

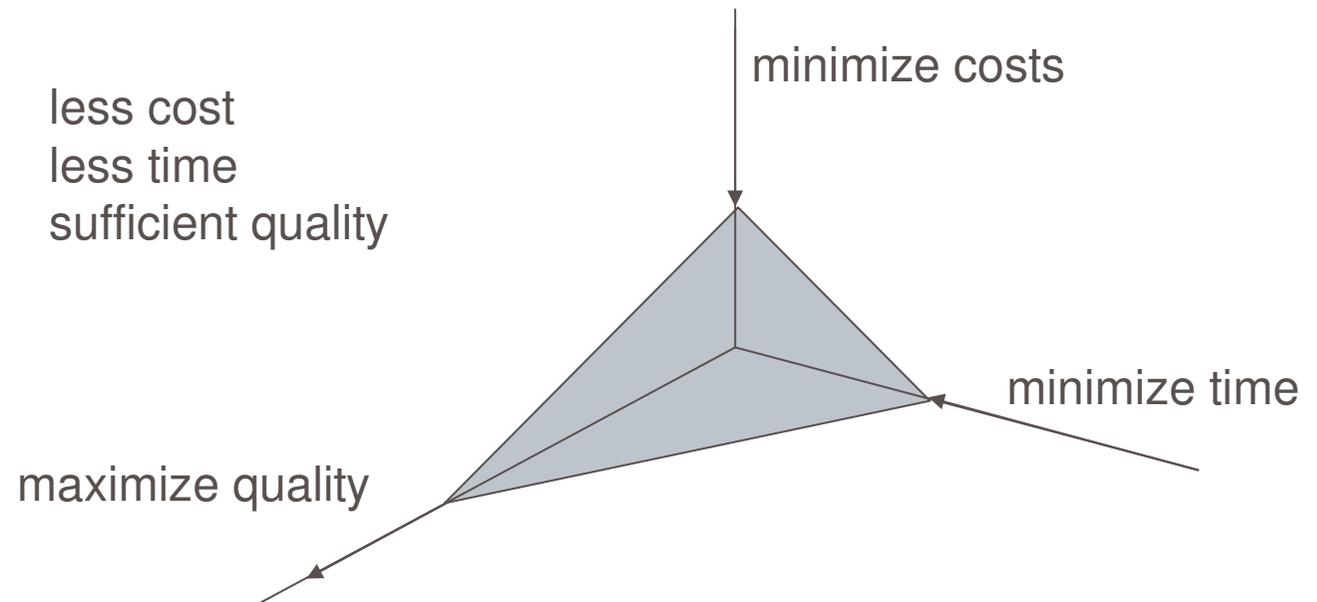
# Research activities



# Introduction

## the need

- Classical approach in product development
- Parametric CAD/CAM systems help to do
- Transfer this to product data definitions
- Create CAD data with:



# Introduction

## Today's factors

- Parametric CAD/CAM systems for easy changes
- Rule based design (design logic etc.)
- KF applications to fulfill quite specific design tasks with high engineering design knowledge effort
- EDM/PDM systems to manage CAD/CAM data and other related documents
- EDM/PDM systems to manage design work flows, product structures, users and rights, etc.
- Interfaces to several CAD/CAM systems for data exchange
- CAD data quality checker (VDA-Checker, Check-Mate, etc.)

## Missing

- Identification of the needs designers have while designing without exhausting meetings, surveys or long lasting implementation strategies in companies
- CAD data quality checks to follow best practices in companies
- Self extending CAD system rules to automate routine design tasks without heavy application engineering effort
- Design complexity logging algorithms for possible cost estimation while designing

# Introduction

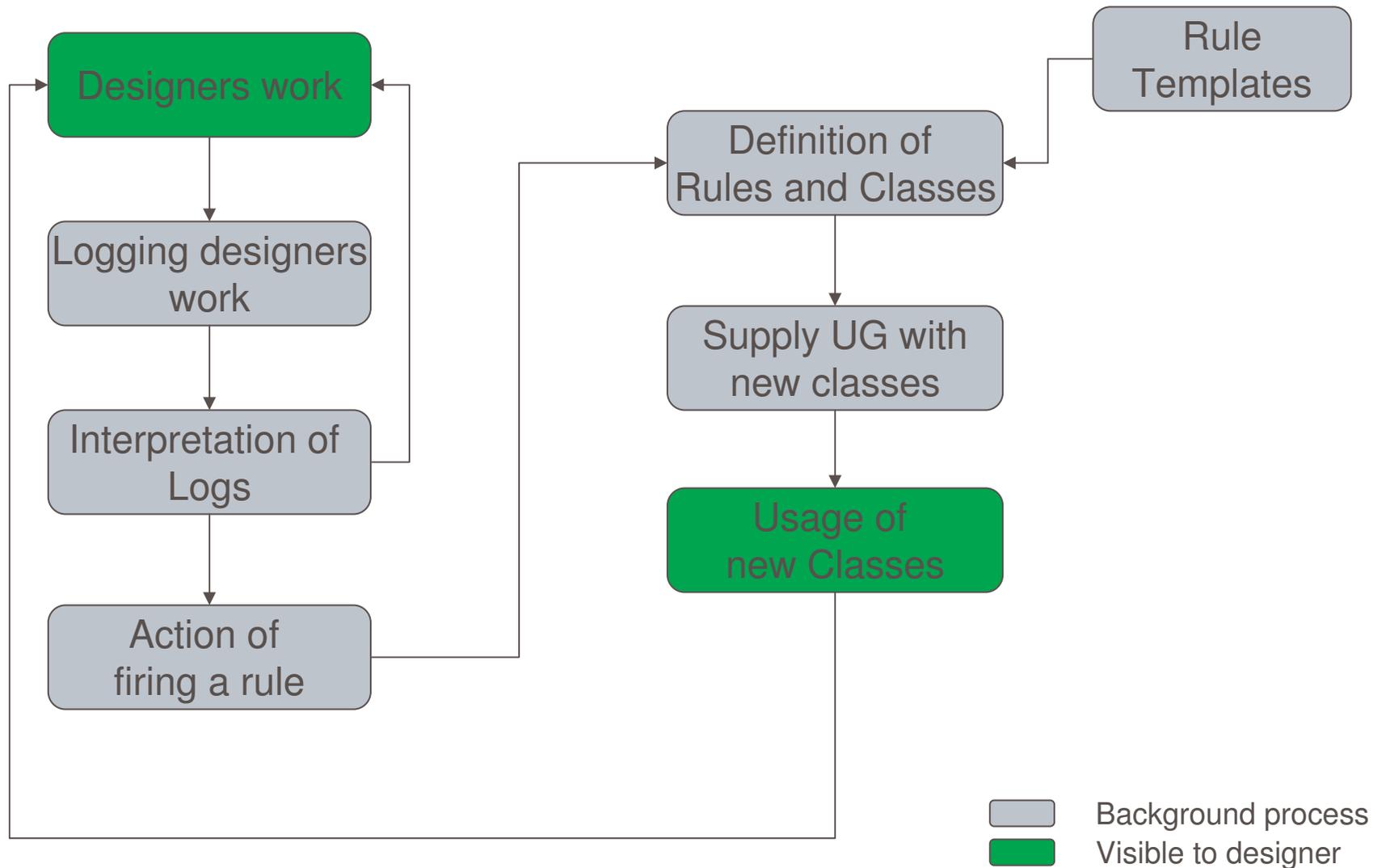
## Characteristics of design

- Designing seems to be chaotic and hard to forecast
- It's hard to find patterns in the usage of CAD functions and features

## Examples of standard functions and features:

- Part new, part open, use of features, use of parameters, save as..., add components, change/delete features
- Timestamp order, can be changed
- Feature Dependency Browsers
- Use of EDM/PDM system information

# Process of self based rule creation



# Logging

## Different ways of logging:

- Event based logging  
When designers use functions (Save, Check, after certain time, after certain amount of features built in, etc. )
  - KF-Log (scan parts for new geometry – ug\_cycleobjectsby...() )
  - KF-Application Log (write log-routines for your KF applications, database entries)
  - Call specific KF design analysis functions (manually or connected)
- Continuous logging  
All the time, when the designer uses the CAD system
  - Unigraphics Log-File (hard to interpret, ask GTAC?)
  - Write separate log file as it is needed

# Action

## Question:

- When a user-process should fire a new rule?
- What will the new rule include?
- How to make sure, that no self-induced rules are fired?

## In terms of continuous logging:

- Looking for patterns in the chaos to create a defined rule

## In terms of event based logging:

- New rules can be fired connected to specific events or manually

# Rule creation

## Basic application for create new rules and Classes

- Event driven - when adding a new component or changing in design takes place
- This action together with some information the user has to provide generate new .dfa files with a specific class that
- A template .txt file contains a standard .dfa class
- With a KF algorithm the new class is build from the template and the user entries
- The class is instantly available to the system

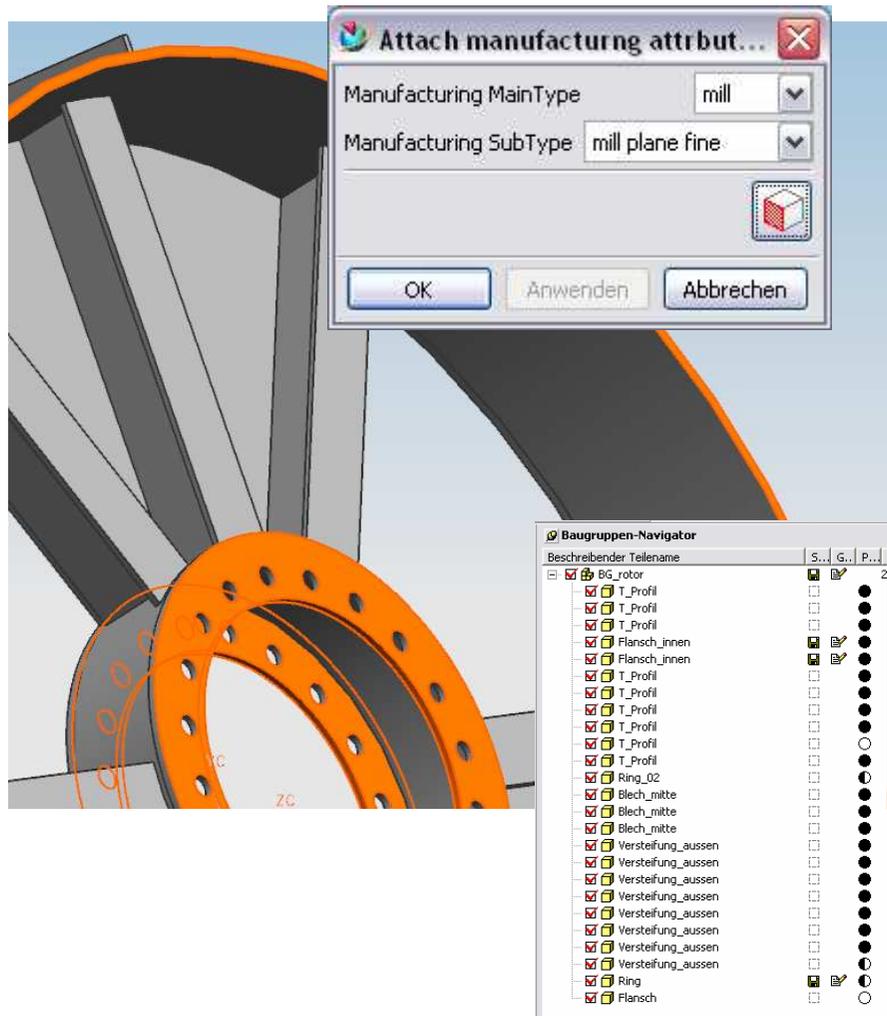
-> The system extends itself

# Example of a self extending KF application - ICE

## Usage for first complexity estimations of machined and assembled parts for possible cost

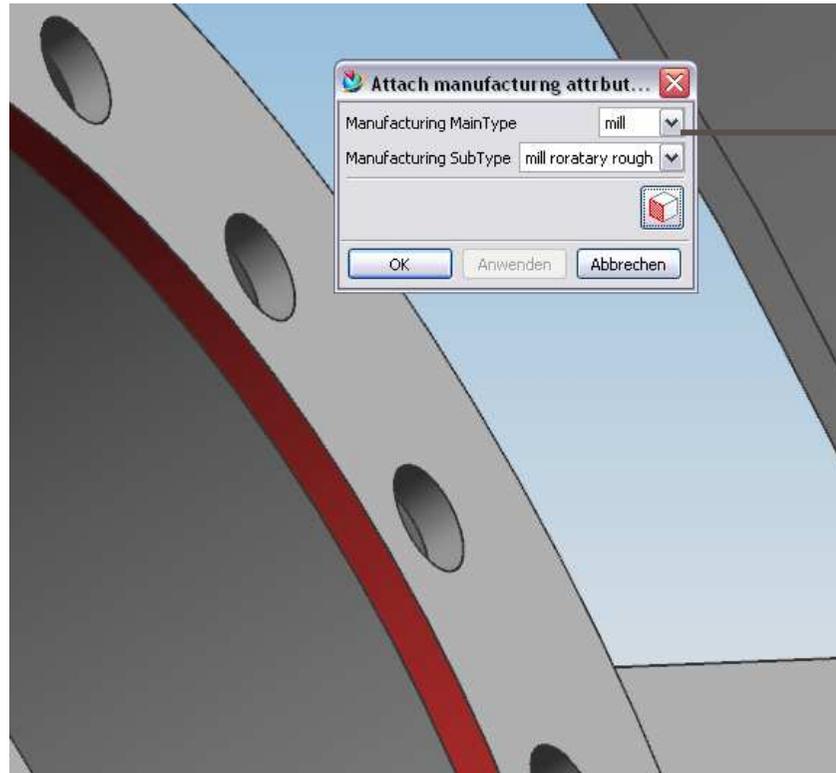
- Start with an empty KF application (integrated complexity estimation – ICE)
- Definition of types of machining (milling, welding etc.) is done, when needed
- Usage of a KF application to define the types of machining on geometry with following information:
  - Mask (which geometry should be selected, edge, face, body, ...)
  - Analysis type (welding, cutting = length of edges, milling = area)
  - Simplified costs per unit (welding, single side = 450€ / m)
- Attributes of are written on geometry
- Machining types are saved in .dfa file for further usage in ICE or other applications

# Examples



- Here: Rotor of a generator
  - cutted
  - welded
  - milled rotary
  - milled plane
- Within an assembly single geometry (edges, faces, etc.) is selected
- Manufacturing type is assigned
- Assigning one component, all same components have the same information (saves clicks)

# Examples



If manufacturing type not available,  
new manufacturing definition  
can be created

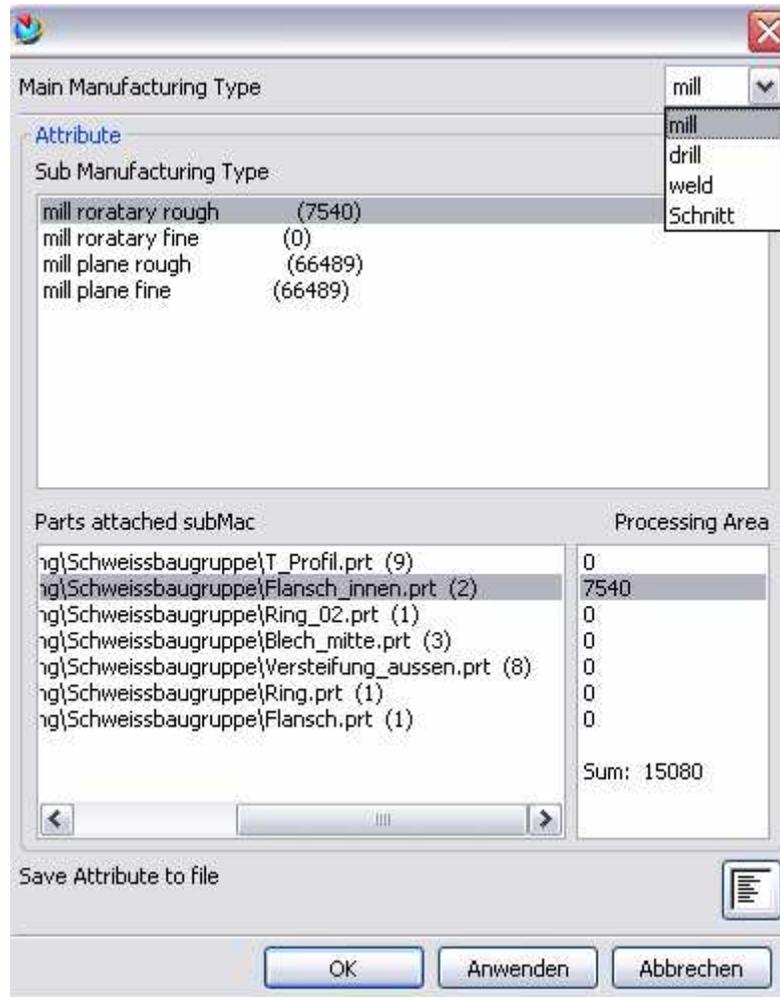


This results in a new .dfa file  
with a new class and new rules,  
that are instantly available to the system

```
#! UGNZ/RF 2.0
DefClass: CF_Requirements1 (ug_base_part);
(Any)      commandValue:          ();
(List)     Requirement_Name_List: if (List_1 != ()) then Loop
                                           {
                                           for $a in List_1;
                                           Collect nth(1,$a);
                                           }
                                           else ();

(String Parameter Modifiable) Requirement_Name: "";
(number)   Requirement_Value:     if (List_1 != ()) then Loop
                                           {
                                           for $b in List_1;
                                           if (Requirement_Name = nth(1,$b)) return nth(2,$b);
                                           return is 0;
                                           }
                                           else 0;
```

# Examples



- ICE collects all available assigned information on the assembly
- Sums up specific analysis results specifically to the manufacturing type
- Result shows a simple manufacturing complexity of parts
- Results can be reported (standard KF)
- If design changes are made, results vary and may show better or worse solutions for manufacturing

# Summary

- ICE can provide a simple way to estimate complexity of design in terms of manufacturing (costs) while designing
- ICE is self extending
- Created rules are simple, but can be used further on from other KF applications
- Might be useful in local and central usage of KF-classes
- To create more complex rules continuous logging should be used
- Hard to build applications, that extend themselves in an intelligent way (might be an AI approach)
- Next step would be a self-learning system
- Further research necessary
- Discussion welcome!

Thank you for your attention

## Live demonstration

**If there are any questions, please ask!**

---

**Dipl.-Ing. Guido Klette**

Information Technologies in Mechanical Engineering

Otto-von-Guericke University, Magdeburg

Universitaetsplatz 2

39106 Magdeburg, Germany

[guido.klette@mb.uni-magdeburg.de](mailto:guido.klette@mb.uni-magdeburg.de)

Tel: ++49 391 67 18094