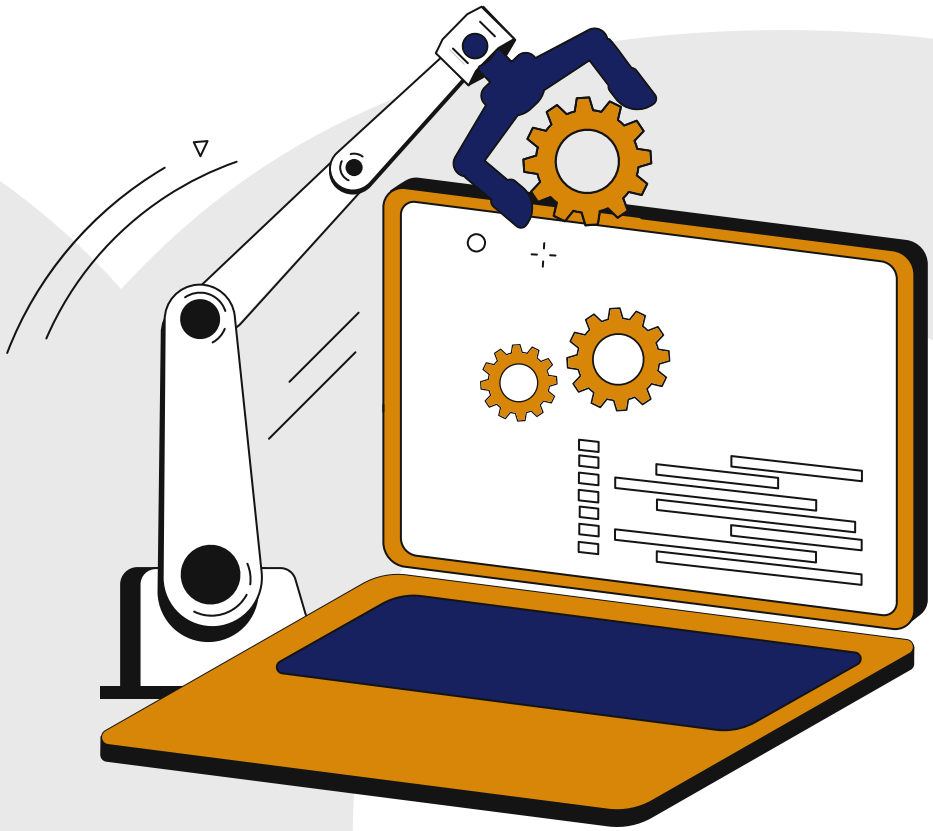
interfaceless design



Automate Your Code

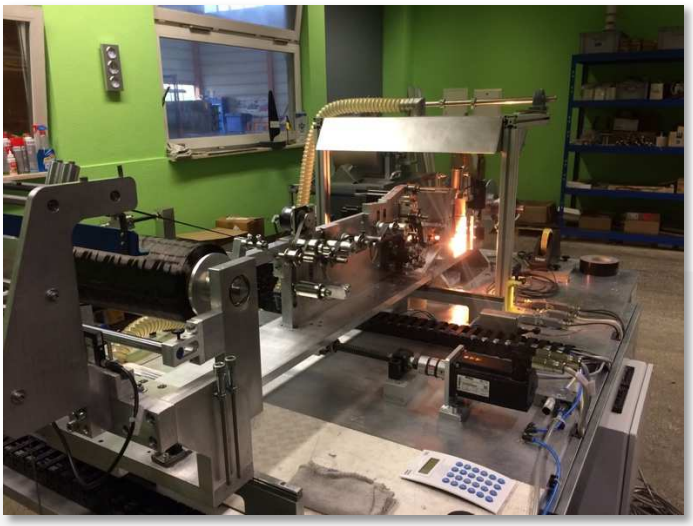
HOOX [SOFTWARE]

HOOX

[SOFTWARE]

Case Study – Introduction

The company **M&A Dieterle** from Ottenbach develops and manufactures systems for the production of carbon fiber tapes. Initially, the entire development of the functions took place on a test system.



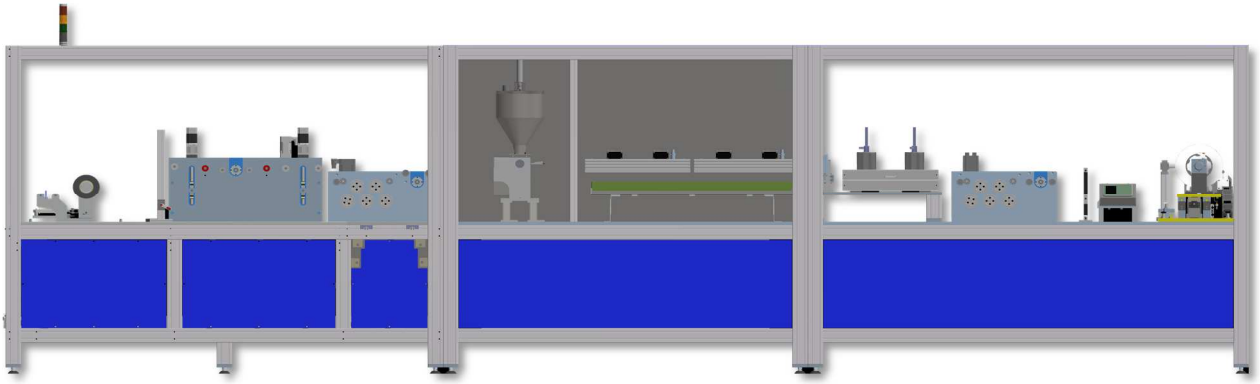
Essentially, this involved an unwinder, a transport system, and a drive for the rewinder. A process takes place in between. The top priority was to create a modular software structure to match the modular hardware/machine setup.

Version 1.0 was created using traditional programming methods.

Once the basic functions were complete, a production system was planned and built based on the knowledge gained from the test system. Due to the increased length, this system already included two transport units, and a regulation mechanism was needed between them to maintain tape tension.

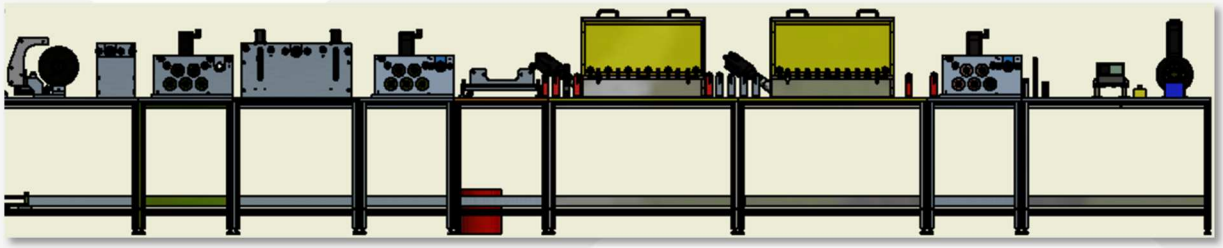
As a result, the first customer orders for tape systems started coming in. However, due to different process requirements, these systems had to be built differently than the “standard” production system.

Production System:



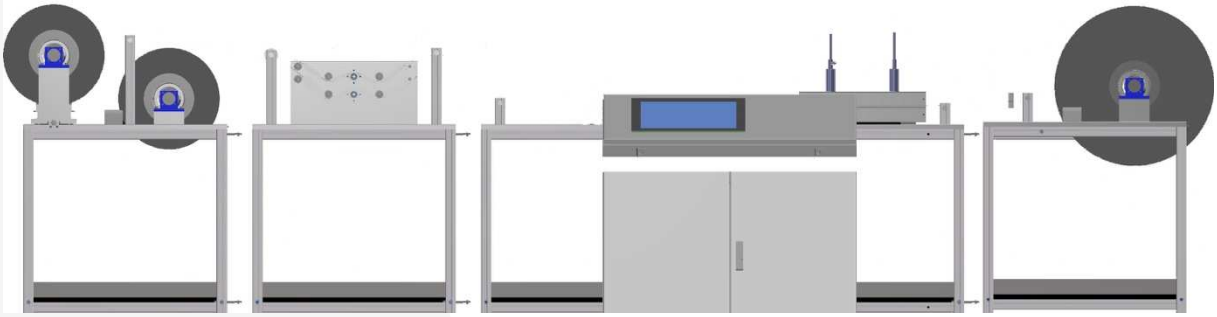
Unwinder, Process 1, Transport, Process 2, Transport, Rewinder

Example 1:



Unwinder, Transport, Process 1, Transport, Process 2, Transport, Rewinder

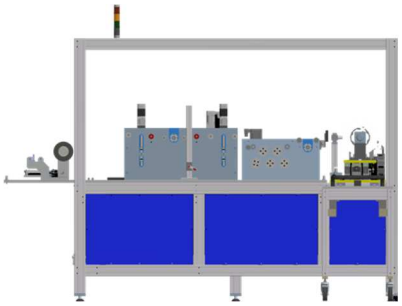
Example 2:



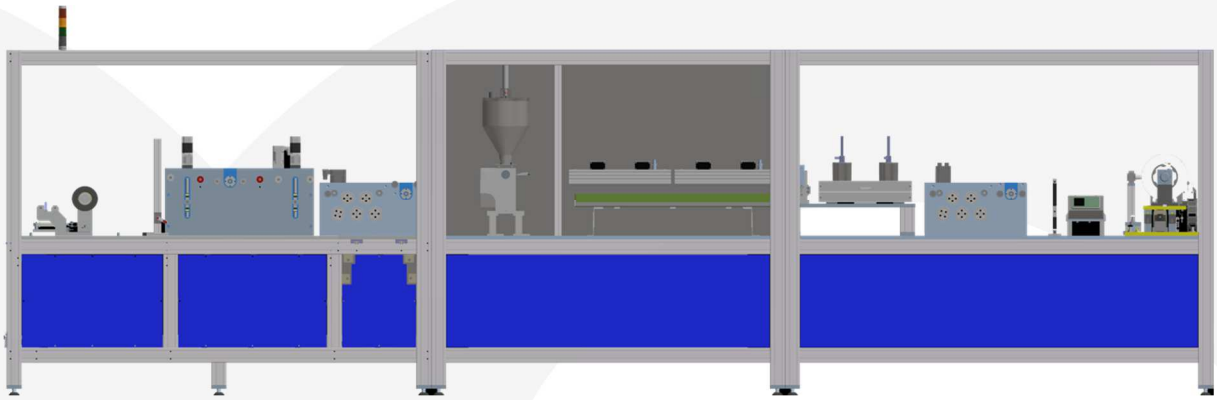
2x Unwinder, Transport, Process, Rewinder

Example 3:

This system can operate in both a “Short” and “Long” version.
The main difference from the standard lies in the electrical setup. While the standard system contains everything in a single control cabinet, here the components are distributed across three separate cabinets.



Short Version:
Unwinder, Process, Transport, Rewinder



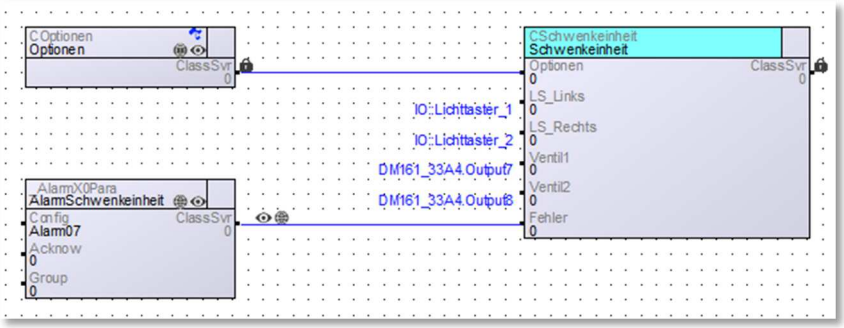
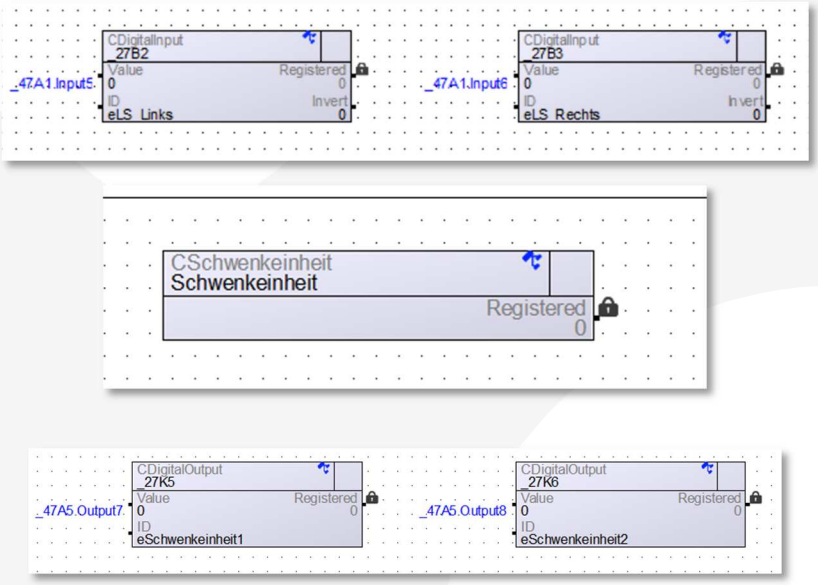
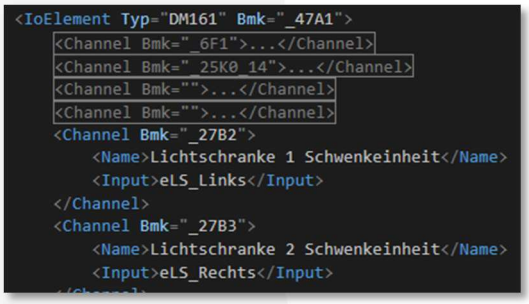
Long Version:
Unwinder, Process, Transport, Process 1, Process 2, Transport, Rewinder

The first two systems were still implemented as classically adjusted programs with optional features. During this phase, I worked in parallel on a **machine framework**. The goal was to be extremely modular and to generate the actual machine software via script.

After discussing with the customer whether we should switch to a framework-based solution, this approach was implemented immediately.

All functions were consolidated into a single library, and individual machines were defined in XML and generated using a (working!) Python script.

Program Excerpt
Implementation with Lasal Class 2

Old Approach	
	The class SwivelUnit was created in the traditional way, the object instantiated, and clients connected.
New Approach	
	The class SwivelUnit was created using the **Interfaceless Design**. Objects for hardware and machine functions were generated from an XML description using the generator. The XML file contains the BMK (circuit reference) from the schematic and is therefore uniquely assignable.
XML	
	The file contains the BMK from the circuit diagram and can thus be clearly assigned.

Old Variant:
If no swivel unit was present, this had to be configured manually in the object's options.
New Variant:
If no swivel unit is needed, you simply omit the object.
Missing instances do not cause additional errors elsewhere in the program.
If hardware signals are forgotten during instantiation, accesses to them are caught, preventing cascading errors.

Summary

The task:

High variance in functionality and hardware setup, yet a standardized program.

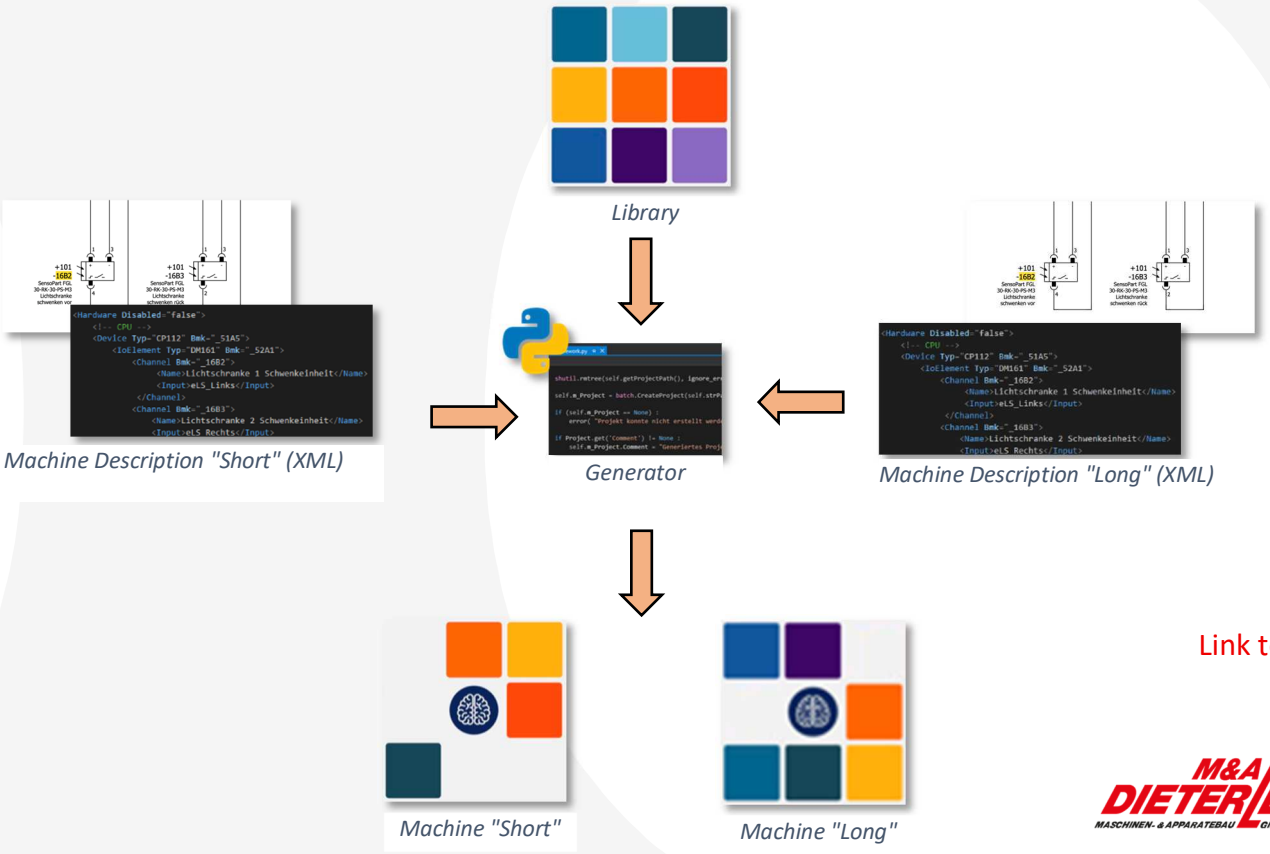
The solution:

Interfaceless Architecture: As few different interfaces as possible
(in this project, exactly one interface) between the functions and one to the hardware signals.
Depending on the requirements, this can look different and is not set in stone.
The description of the specific system is now just a lightly readable XML file.

The result:

- Systems/machines can be fully customized for the customer.
- Flexible electrical planning (each system can be completely planned freely).
- Transfer of BMK + comments into the generated program.
- Master/slave operation between drives can be selected.
- Blocks can be tested independently of each other; possible dependencies can be easily represented with stubs or mocks.

The generator:



Link to project: