Industrial DevOps

Fast Development Cycles and High Software Quality in Automation are No Longer Contradictions

vorausrobotik

Carl-Buderus-Str. 7 | 30455 Hanover | Germany | **vorausrobotik**.com

1 Summary

Industrial automation is subject to changing requirements in ever shorter cycles. This includes of course the automation task itself (e. g. product variety and number of variants, individualisation, troubleshooting). However, normative regulations, e. g. due to the Cyber Resilience Act (CRA), or digital sovereignty as a consequence of increasing geopolitical uncertainties also contribute to this situation. The implementation of these requirements, some of which are even unknown when the automation system is designed, poses major challenges: The complexity and size of automation projects are increasing, this holds true especially for the software part. At the same time, the availability of the necessary experts is decreasing due to the demographic change. The situation is exacerbated by the heterogeneous and largely incompatible software landscape: In industrial automation, a wide variety of software modules have to be created, which often are programmed in proprietary, outdated, and different languages. As a result, (subsequent) adaptations are very time-consuming and software tests can only be carried out on the real system towards the end of the project – this leads to high risks, for example in terms of system downtime or damage. Ultimately, the costs inevitably increase, and the duration of the automation projects is extended, which jeopardises their profitability. These challenges cannot be met efficiently with the software tools currently available for industrial automation.

On the other hand, the DevOps philosophy known from the IT (information technology) world proves that fast development cycles, flexible yet compatible tool landscapes, and high software quality are not a contradiction in terms. This white paper shows requirements to be met in the OT (operational technology) world to transfer the successful DevOps approach to the industrial automation landscape. As explained in more detail below, the solution lies in a software platform that is both programmable in high-level language and real-time-capable to orchestrate all components of an automation solution in a standardised manner. In addition, virtualisation of these components is required. This unlocks efficient development even without real hardware. The latter enables high test coverage prior to deployment – errors can be found and fixed at an early stage at low cost. As a result, software development is decoupled from hardware availability, resulting in fast development cycles with high quality. Our case study shows that the costs for installing updates can be reduced by more than 90 %.

We call it **Industrial DevOps**.

2 Introduction

Industrial automation technology is crucial for commercial success – not only does it enable the economically competitive production of a wide range of goods, but it also helps to cushion the effects of demographic change. Nevertheless, both manufacturers of automation solutions (i. e. integrators and technology providers) as well as manufacturing companies themselves are confronted with numerous requirements that drive up expenditure and costs and, therefore, increasingly jeopardise profitability.

The first reason for this is the inflexibility of older systems, which have grown over the years. These can be upgraded with great effort only – employees and integrators with the relevant expertise are no longer available, software versions are unknown, software interfaces are incompatible with the latest IT infrastructure, etc.

However, this also applies to new systems – various proprietary components providing limited compatibility only are part of the automation solution. In particular, manufacturer-specific programming languages and tools force different experts to work simultaneously and make holistic system simulation, programming, and maintenance almost impossible. Ultimately, essential functions can only be implemented and tested on the real system – errors found at this late stage of the development process or necessary changes lead to an explosion in costs and effort. Figure 1 illustrates this impressively – it shows when errors are made (85 % during programming) and when they are discovered. The later discovery happens, the higher are the costs for fixing. In today's development of automation solutions this is particularly the case towards the end of the project (currently approx. 75 % during system testing). Therefore, the aim must be to move error detection and avoidance to the early part of the software development process.

Regular software updates, that will become mandatory in the future as a result of the CRA, integration into the existing IT infrastructure, which is subject to constant changes, or modifications that arise during regular operation (e. g. due to new products or product adaptations) require a high amount of skilled employees and comprise major risks of errors and downtime. A small example: A medium-sized company delivers 100 systems per year, each of which runs for 10 years. This results in 1,000 systems in the field that need to be maintained, updated, etc. If we assume one necessary update (e. g. due to the CRA or product changes) per system and quarter, this conservatively results in 4,000 service assignments per year – considering the availability of experts and the risk of production downtime, this is hardly economically feasible.

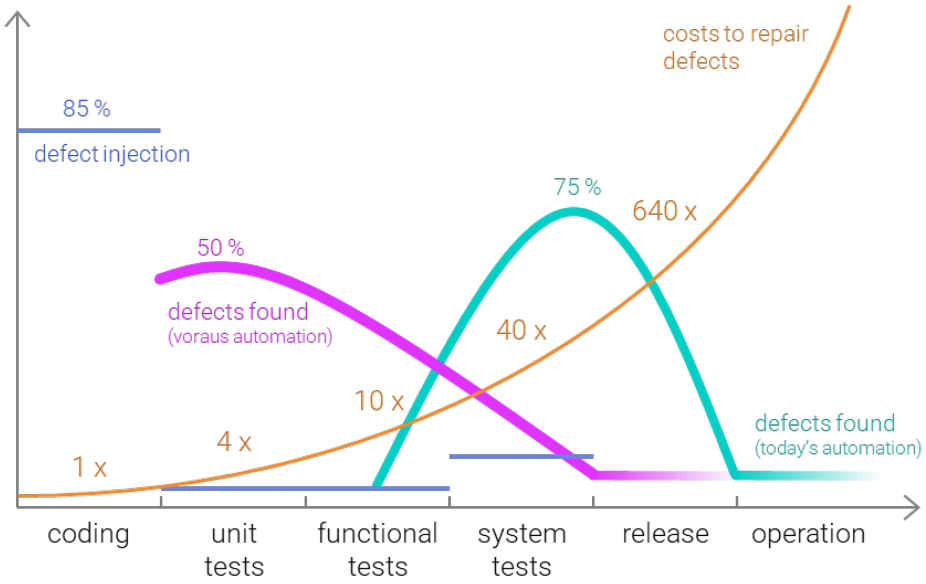

Figure 1: Error occurrence and detection in software development
(Graphic based on Jones, Caspers. Applied Software Measurement: Global Analysis of Productivity and Quality)

What needs to be done? We urgently need a solution that is comparable to the DevOps approach in IT including a holistic view of the entire software life cycle of an automation cell across all components. This is the only way to drastically reduce development times, limit the number of experts required, detect errors at an early stage, and update systems in the field with minimal effort. At the same time, the digital sovereignty of system operators must not be jeopardised, and new dependencies must be avoided.

3 Getting to the Root Cause

The problem mentioned above is not new and is addressed by many providers. However, they almost always focus on partial solutions and their work arounds are based on the existing software and tool landscape. Although this may solve a problem in certain areas, it remains piecemeal in principle. In our opinion, we need to get to the root cause.

3.1 What is DevOps?

DevOps is an IT philosophy that stands for development and operations, i. e. a holistic view of both. This applies not only to the processes, but also to the underlying toolchain. The aim is to develop and deliver software quickly while maintaining high quality. In this context, reference is often made to the "infinity loop" (Figure 2).

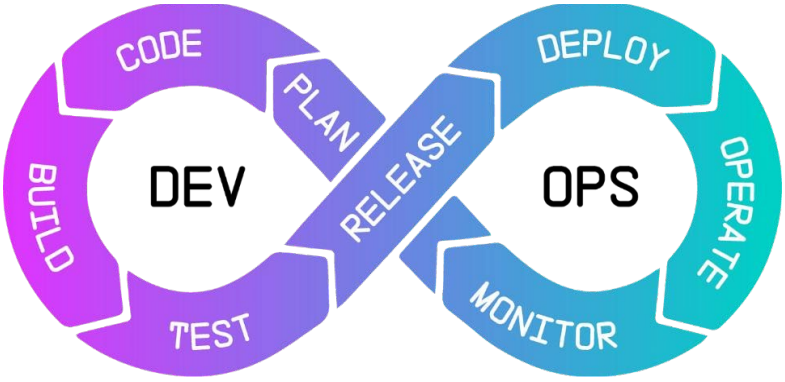

Figure 2: DevOps – process steps of modern software development

In addition to planning, programming, building of the software and testing it (typical development phases) – left part of the figure – the term DevOps also includes the creation of releases, their deployment and the operation of the software as well as monitoring. The respective process steps are supported by powerful, compatible tools. Early (and automated) testing of the software is crucial for high software quality. DevOps is now firmly established in IT and has become the standard – for example, the websites of online stores are updated several hundred times a day without the end user being aware of.

3.2 What are the Challenges in Automation Technology?

Why is it so difficult to transfer the established IT approach, which solves many of the problems mentioned in section 2, in the automation (OT) landscape. Or, in other words, why did we not achieve the so often requested IT-OT convergence?

There are two main reasons for this:

- The programming languages used in automation are outdated (some have their origins in the 1970s) and often proprietary. One example is the structured text for PLC programming (standardised in EN 61131-3) or robot-specific programming languages. On the one hand, this implies that modern software development tools are not accessible and, on the other hand, that the development environments supplied by the manufacturers are not compatible with each other and only cover the respective sub-area. Ultimately, this results in a highly fragmented software and tool landscape with numerous breaks.
- The second main reason is the lack of holistic testing of the entire automation solution. Although it is possible to carry out virtual (i. e. hardware-independent) tests with the manufacturer-specific tools, these are very limited (i. e. linked to the respective component). The entire control system of an automation solution (including so-called edge cases) can be tested with the real automation cell only. As shown above, this leads to high costs for troubleshooting and requires the availability of all components, which, depending on delivery times, is only given late in the project. As a result, delays and cost increases are usually unavoidable.

The consequences are well known: Systems are rarely updated and hardly optimised once they have been successfully commissioned, as the risk of errors and, therefore,

(unplanned) system downtime is too high. However, ever-shorter product life cycles, high number of variants and, last but not least, legal requirements including the CRA are fundamentally opposed to this.

4 Solution: Industrial DevOps

The good news is that a solution for industrial automation exists. The DevOps IT toolchain can, in principle, also be used for OT – but is currently inaccessible due to proprietary programming languages and incompatible manufacturer-specific development environments.

An important part of the solution is the possibility of programming all components of an automation cell in high-level language, without restricting real-time capability. This breaks down the incompatibilities and makes existing IT tools accessible. For voraus robotik, the Industrial DevOps toolchain is shown in Figure 3. Here, common (sometimes even free of charge) IT tools are assigned to the individual processes, which cover the entire automation software stack and are compatible with each other. Furthermore, they often outperform the proprietary partial solutions in terms of functionality, UI design etc. There is no vendor lock – in principle, users are free in their tool choice. Additionally, there is also no cloud enforcement – again, the customer has full control. Therefore, Industrial DevOps provides a crucial contribution to digital sovereignty.

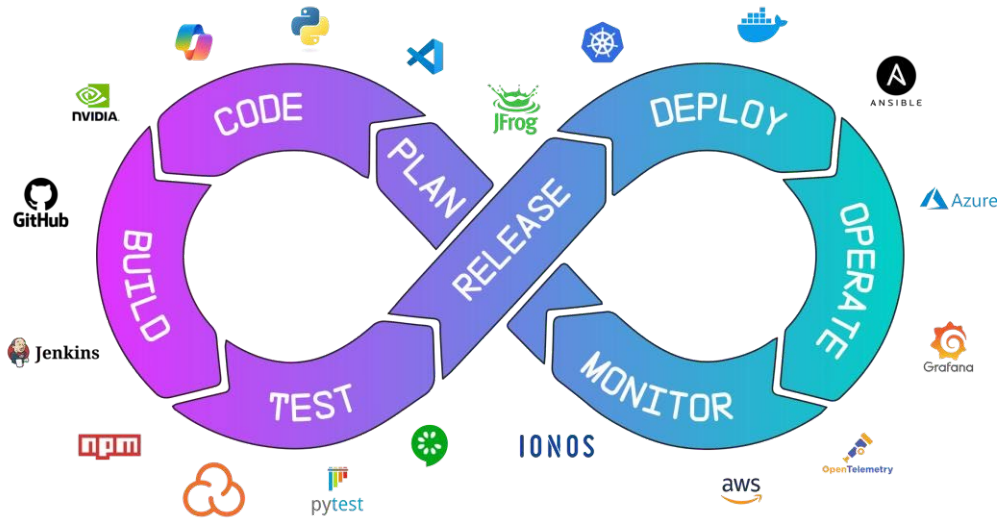

Figure 3: **Industrial DevOps** from voraus robotik incl. toolchain (source logos: manufacturer)

The second part of the solution is the virtualisation of the automation components. This means that the software can be developed and tested against the virtual counterparts and optimisation as well as bug fixes can be carried out at an early stage and, therefore, cost-efficiently. As the interfaces / APIs of the virtual components are identical to the real ones, deployment does not require any adjustments. A standardised and consistent code base is established.

Where does the OT-IT convergence take place?

Figure 4 illustrates this using the well-known automation pyramid (according to ISA-95). The decisive levels are Level 1 and Level 2: If incompatibilities are dissolved here and APIs are standardised, seamless access from the IT level to the fieldbus level becomes possible. This is an absolute prerequisite, especially for big data and artificial intelligence applications.

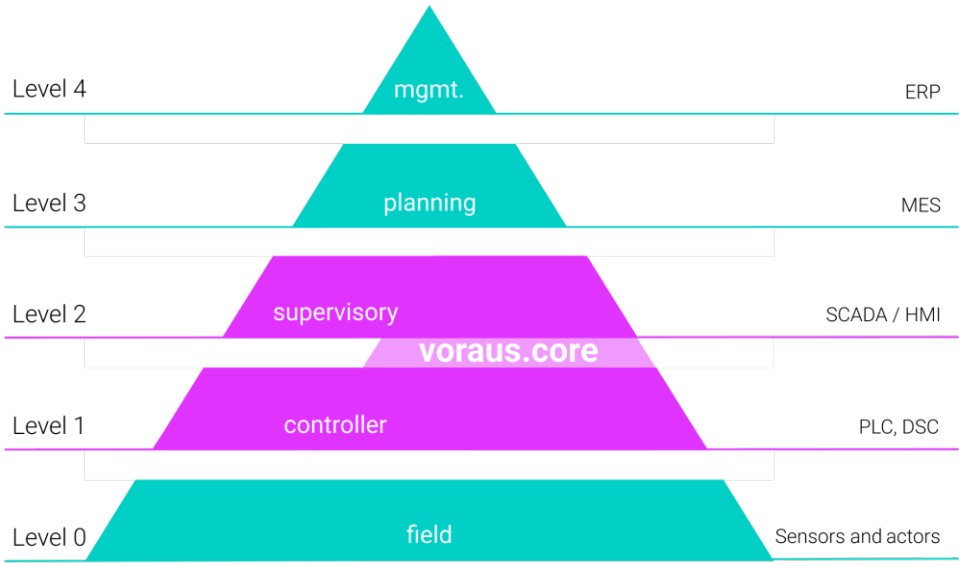

Figure 4: The voraus.core (see below) makes Level 1 and Level 2 IT-capable

4.1 Real-Time Capable SW Platform – voraus.core

Even if all components are programmed in high-level language, the orchestration of the system must still be robust and in real-time. This requires appropriate drivers that are based as far as possible on industrial standards (e. g. EtherCAT, PROFINET, OPC UA) or integrate proprietary systems (e. g. robots from various manufacturers). These are the two lower software layers in Figure 5. To ensure expandability, the software stack of the control platform is modular and containerised; the interfaces are based on IT standards and are open.

Based on the API, the actual application is programmed in high-level language (e. g. Python), whereby provided software modules (e. g. for robotics) can be used on the one hand and customised functionality can be added on the other. Thanks to standardised and modern interfaces, the connection to existing IT (e. g. HMI, SCADA, ERP systems, fleet control for AMR) is possible without any restrictions and in the shortest possible time. Furthermore, the integration of the latest AI methods is no longer a problem thanks to high-level APIs.

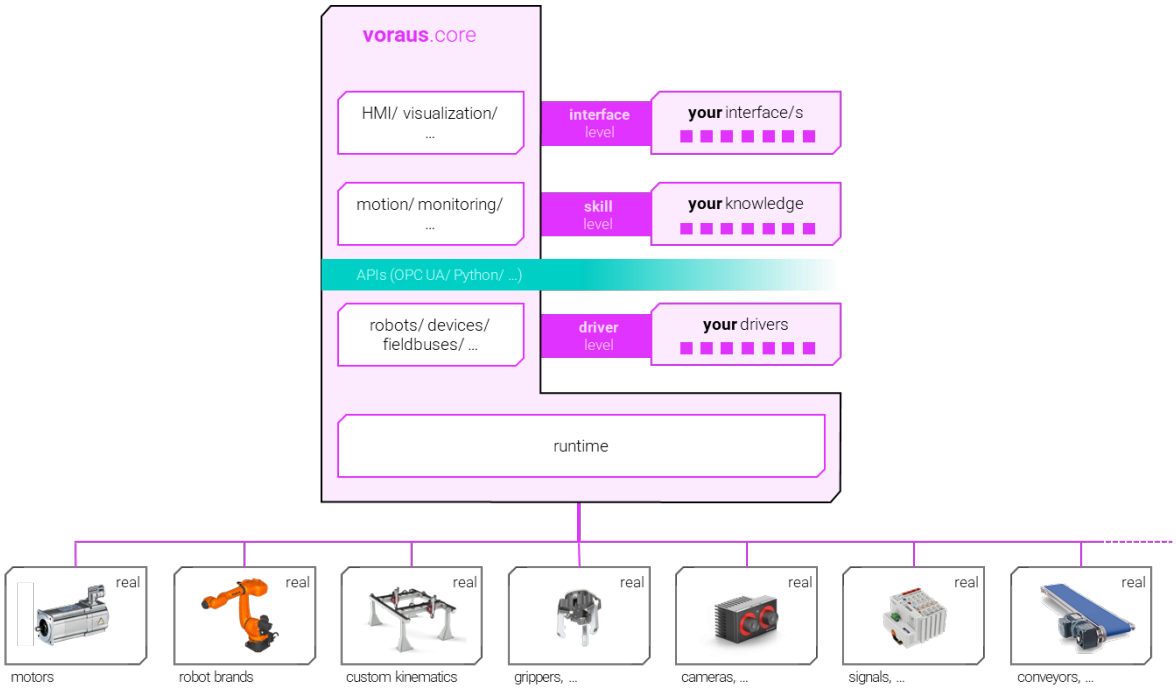

Figure 5: voraus.core – from the fieldbus level to the application

The architecture of voraus.core described above ensures the right-hand side of the DevOps infinity loop.

4.2 Development Environment – voraus.pioneer

The left-hand part of the DevOps infinity loop is realised by the left-hand part of Figure 6. Numerous modern IT tools interlock to ensure efficient programming. The selection of software tools is not predetermined and can be customised to individual preferences. The virtualisation of the components described above helps to ensure that the entire application can be developed independently of the availability of the hardware. In particular, systems can be quickly visualised, simulated and optimised taking their physical properties into account (thanks to the numerous (open source) physics engines available). Test cases are created in parallel with the definition of SW specifications or programming in accordance with the Industrial DevOps philosophy. Automated tests ensure (after each change) amongst others that the required KPIs (availability, cycle time, etc.) are met. Edge cases and error injection guarantee robust behaviour of the system in the field – test coverage of over 95 % can be achieved before deploying the software, whereas it is likely to be less than 30 % with classic PLC programming. However, this is only possible if development and operation use the same code base and the virtual and real components have the same APIs.

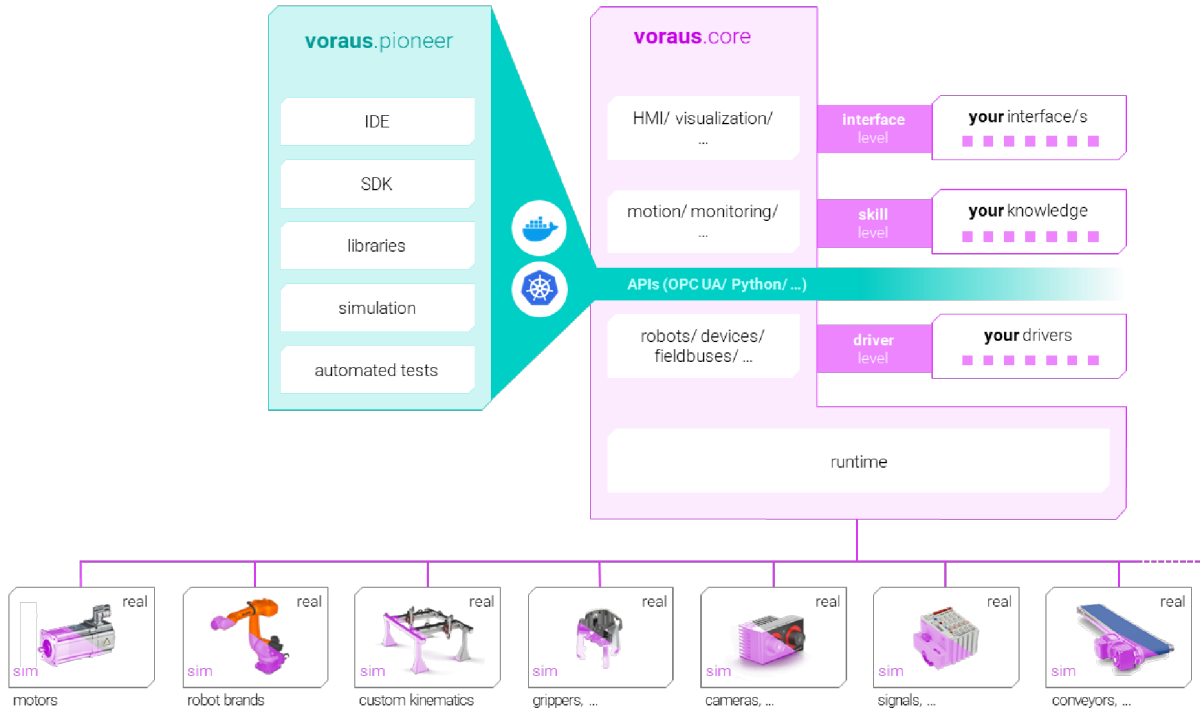

Figure 6: Holistic implementation of **Industrial DevOps** by voraus.pioneer and voraus.core

5 Case Studies

In this section, we will use two examples to illustrate the advantages of the Industrial DevOps approach compared to the classic approach.

5.1 Update and Rollback

In order to distribute updates to automation cells with low risk (or to initiate a rollback in the event of problems), it is essential to create software with high quality (i. e. with high test coverage). At voraus robotik, we have established the following test pipeline for internal and external purposes: The Jenkins-based test pipeline follows a structured Industrial DevOps workflow with several stages to ensure software quality and release deployment. The test cycle starts with pipeline preparation, including initialisation of workspace variables and formatting checks. Artefact processing includes retrieving, processing, and completing build information, followed by the actual build process, generating documentation and running unit tests as well as static code analyses. Further steps cover compliance checks (e. g. OSS licence compliance), the creation of container images and packaging for releases. Deployment as well as system, integration and end-to-end tests are then carried out on the defined target hardware. Final steps in the test pipeline, such as SonarQube analyses and saving of metadata, ensure traceability.

The pipeline described enables fully automated testing and the creation of releases in the shortest possible time. The manual effort that used to be common, at least in part, is completely avoided and the manpower required has been reduced from several days to a few hours. As the creation of test cases is part of software development anyway,

the one-off additional effort is limited to setting up and filling the test pipeline described above.

The process of installing updates has also changed drastically: The containerised software can either be offered by download (on a customer-specific basis) and is installed with just a few clicks or can be distributed via a Kubernetes cluster. Whereas it used to take approx. 60 minutes for an update, it now requires less than 5 minutes. If we consider again the example of the machine manufacturer with 1,000 systems in the field and 4 updates per year and system, the figures are as follows.

	Classic approach	**Industrial DevOps**
Duration per update	60 minutes	5 minutes
Costs per update (200 €/h)	200 €	17 €
Costs per system (4 updates per quarter)	800 €	68 €
Costs for 1,000 systems	800,000 €	68,000 €

In this example, the savings with Industrial DevOps add up to more than 90 %.

5.2 Optimal Selection and Programming of Robots

autonox Robotics GmbH offers a variety of robot kinematics for a wide range of applications. But which robot is the right one for my application? Answering this question takes several days. To make it easier for customers to choose, we have linked the autonox Finder, which contains several hundred different kinematics, with our voraus.pioneer – thanks to open interfaces, this was possible without any problems. In this way, various robots can be automatically loaded into the application simulation and tested (reachability, interference contours, cycle time, etc.), see Figure 7. The resulting application code can be used directly for the real system after selection of the right robot. Reprogramming is not necessary. The time required has been reduced to a few hours instead of a couple of days.

Figure 7: autonox Finder and voraus.pioneer: optimised selection of kinematics made easy

6 Conclusion

Fast development cycles, as increasingly required in industrial automation, are currently a major challenge. In addition to the declining availability of experts, the main reason for this lies in the proprietary components of the automation cell from a wide range of suppliers whose programming languages and development tools are not compatible. Modern software development, on the other hand, demonstrates that fast update cycles and high quality do not have to be a contradiction in terms. To enable Industrial DevOps, two prerequisites must be met: programming of all components in high-level language and virtualisation of these components for development and testing while retaining the API.

If these conditions are fulfilled, a consistent, modern, and powerful toolchain (without vendor lock) is available and fast development cycles with guaranteed quality become reality in the OT sector. Users of automation solutions regain their digital and technological sovereignty – an important feature, especially in times of geopolitical uncertainty and fragile supply chains. The consistency of data across all layers of the automation pyramid is also a prerequisite for efficiently integrating automation solutions into existing IT infrastructure and for profitably implementing artificial intelligence solutions.

7 Company Information and Contact Details

You can find more information about our Industrial DevOps solution and how we can support you on our homepage http://www.vorausrobotik.com/en/. Our documentation with many examples is available here: https://docs.vorausrobotik.com/ – now with AI-powered search.

Contact details

voraus robotik GmbH

Carl-Buderus-Str. 7 | 30455 Hanover | Germany | **voraus**robotik.com

Contact person

Tobias Ortmaier | tobias.ortmaier@vorausrobotik.com