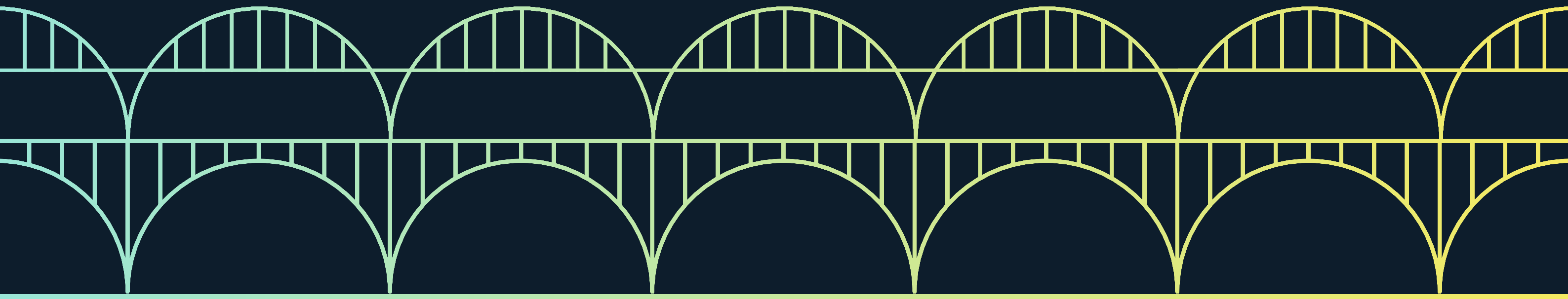
MDENET RESOURCES

The home of model-driven engineering

MDE for DevOps



Jörn Guy Süß

This document introduces you to some challenges in DevOps with pointers to how Model-Driven Engineering can help address these challenges. We do this through some case studies to give a concrete feel for what MDE might bring to the DevOps table.

Before we begin to give an overview of the studies, we would like to introduce devops or development operations by example. If you are a DevOps practitioner and use DevOps daily, you can safely skip this introduction.

1. KEY CHALLENGES

DevOps: Programming in the Very Large

As an example let us consider a logistics information system that allows parcel delivery track and trace for contracted drivers. Drivers register with the company, install an app and start delivering parcels. Customers see parcels being delivered on a website.

This system has a speed-optimized storage backend that enhances the functionality of a database product written in Rust. It has a layer of microservices written in Go, some transactional message queuing components written in Java, a web portal written in Typescript with Help and Documentation written in Markdown. A kubernetes helm program describes how the system components are placed on servers and provided with mutual references to each other so they can collaborate. Onboard clients for delivery drivers written for iOS and Android, and maybe even versions for Android Auto or Automotive. The apps are published to the respective platform marketplaces. All of these components need to be versioned, built, tested, released, deployed and any feedback on their performance integrated back into the process.

This is not an unusual example of a typical business system. Its construction requires operations that in turn require installations of different development languages and development tools. Potentially, these different languages compilers and tools may even require different operating systems to work.

Let us look at how this is done for just the website with its help files. We have five different computer installations to perform the build steps: The typescript computer compiles and tests the typescript installation. If the build is successful it releases the result through an NPM repository. The markdown computer in parallel spell-checks and semantic checks the documentation. If the build is successful it releases the result through a general artifact repository. The Kubernetes helm computer collects these two repository entries and deploys them within a test installation within an off-the shelf nginx web server. That is it instructs a service to install a new server or servers, install components onto that server and then connect that system of servers to the required network, possibly the internet.

DevOps is the programming of coordination between build steps across programming-language-system and tool boundaries in development, including deployment to operations. In addition, it encompasses the feedback loop from operations. This means that if a program fails in production or in testing that failure information will feed the development directly. Generally, the failure is recorded, and provided as a work item to investigate for the developer. Failure information is input to the devops program. Devops is the programming expression of software engineering. It builds the engine that builds and runs the software process. It is programming in the very large because it encompasses not just the object program (the product) , but the subject program (the process).

Devops achieves this goal generally by creating a chain of macroscopic operations, each located on a computer that is specifically installed with the programming language or operations tools required for the step. Devops languages are process languages that allow the coordination or training of such installations and the provision of textual instructions to the various computer installations. They also allow the passing of artifacts between the installations.

Devops programs are written for and executed on proprietary server products that offer a runtime for Continuous Integration / Continuous Deployment (CI/CD), such as Github Actions, or Gitlab CI, Circle CI, Bamboo and many more.

The example above speaks of specifically installed computers to perform the various steps. However, these are almost always replaced by containers. A container is the definition of an installed computer.

The example shows how DevOps is different from running just Maven, Gradle, CMake, Helm or any other programming or test tool. DevOps is programming above all these tools including the aspect of deployment and feedback.

This enormous scale causes DevOps to accrue enormous software engineering complexity. Optimisation of DevOps processes is as challenging, if not more challenging, then the design of the actual product. For example, the ability of an organization to respond to the publication of a crucial software security exploit can make the difference between ongoing business and bankruptcy. This directly correlates with how long it takes to deploy a new version of software.

At the same time, the human process participants cannot be burdened with understanding the technicalities and details of the complex DevOps program. They need to interface with useful, adequate abstractions. For example, an author of website content will think in document versions and terms of reviewing and publishing, not in terms of pull requests and commits. Thus, not only is DevOps a complex problem on its own, it also suffers from additional, accidental complexity of being captured in highly technical terms rather than terms that are accessible to the human process participants.

What MDE has to offer

Model-driven engineering (MDE) is about developing systems from domain-appropriate abstractions and encoding the translation into lower-level, technical concepts in automated transformations. Two key ideas from MDE can be helpful in the DevOps context: domain-specific languages (DSLs) are languages that provide problem-specific concepts that can be understood by human experts – this can help human process participants to engage productively in the DevOps process. Model transformation languages allow the executable specification of translation between different models, expressed in DSLs, and code, for example instructions for DevOps tools and infrastructures.

Designing DevOps - Programming the Process is Complex

The Challenges

As a programming practice for processes in large businesses, DevOps has two major challenges: The heterogeneity of DevOps languages and the speed and potentially irreversible nature of executions.

Challenge 1: Heterogeneity and Lock-in

Much like the domains of programming languages, the domain of Devops is characterized by competing implementations of the machinery that carries out the build process. Products like Gitlab CI, Github Actions, CircleCI just to name a few, fill the marketplace. During the recent economic downturn companies have come to realize by the example of cloud providers and services, how costly being locked into a specific vendor API can be.

In the case that a DevOps vendor fails, falls behind in development, or raises usage prices in an unacceptable manner, transfer of a DevOps process to a competitor's DevOps product requires analysis of the high-detail vendor-specific DevOps code. Given this code is often expressed in hard-to-analyze scripting languages, due to its relationship with operating system shell scripts, the cost of this analysis can be as high as reimplementation. Further, if multiple DevOps products are deployed at the same time, reporting on execution in a unified way has no support.

Challenge 1: Heterogeneity and Lock-in

Because development operations affect the output of a company, having homogeneous standards with regards to naming, documentation and implementation of a process is important. In the area of architecture frameworks like TOGAF show how this is done for products and systems that are deployed. For DevOps, there is no equivalent at the moment.

At the far other end of the abstraction spectrum are software systems and teams that are not able to perform a fundamental build from their source code any more, and have non-reproducible intermediate patched binaries and libraries checked as binary large objects into version control. In this case the provenance and source code link cannot be recovered any more. Under such circumstances, security and quality certifications are usually impossible to achieve, leading to business disadvantages and losses.

Challenge 2: Simulation and Debugging

Development and deployment processes in DevOps often have an irreversible nature. Once they are triggered they cause resource-intensive program executions that bind storage space and server processing capacity, and in the case of a deployment, immediately affect users.

While some of these costly effects can be mitigated by dividing the DevOps process into stages and checking each, the actual execution of the mechanism is still time-consuming. Devops developers currently have little means to simulate and hence to predict the implications of a change to the development process. This includes the simulation of values passed between higher-level parts of the process such as variables and parameters.

The MDE Approach

Using MDE, we would design a domain-specific language, which enables DevOps engineers and architects to express their processes in a way that is close to their conceptual understanding of their environment, and where general processes could be refined for and rendered to the artifacts of the underlying CI and DevOps products.

Language-based processes could be modeled to follow best practices and concerns could be shared between language- and process-oriented engineering. Concrete implementations would be unambiguously identified using version control system locations and commits. Reference ids (branches and tags) can be used to define requirements that link in with version control planning.

The Benefits

Explicable and Tractable Process

At the core of modeling DevOps is the advantage that the process can be viewed and queried as a model, and details of the process are hard-referenced from the model. This provides tractable information and documentation generation on processes that can be convoluted and obscure without governance.

For a DevOps developer, all instances of a certain DevOps step could immediately be queried because they would correspond to the modeled step artifact. A DevOps developer could ‘list’ all the ‘builds’ that were instances of a certain design and visually inspect them, or even enforce a condition on those builds. For example, if a file always held content corresponding to an invariant, that file may not need to be written, instead, the invariant can be documented.

More Agility

The availability of architecture information and explicitly stated responsibility and purpose prevents process-based technical debt, where a project team paints itself into a corner and cannot move, because nobody understands and can hence change its process automation any more. In effect, it defines a type of interface between the elements of the process at the level of the team’s choosing.

This does not mean that all details of the DevOps implementation are captured, but that the elements that are essential to create an effective model of the DevOps process are captured. New DevOps operations can then be developed based on existing elements of the model.

Hence by extension, a team that understands that the effect of a change at the interface level will be able to make changes more quickly and confidently. This is generally what is termed agility in software development: The ability to make significant changes more quickly, because flanking measures protect the team from unexpected side-effects of those changes.

The Benefits

Reduced Cost

The ability to compare costs and move a project to a different CI and DevOps product more easily provides leverage towards vendors in the market. As with all negotiation the ability to act is often sufficient to further one's position.

The ability to port, reproduce and parameterize DevOps processes at scale reduces individual setup cost. If such a setup model and system is provided as open-source, the providing company can propose its development style of DevOps and shape terminology in the public sphere. As a result, training needs for staff are reduced when hired from OSS projects using the approach. Staff training is a highly significant factor in time and money.

Reduced Vendor Risk

Using such a model, a company may adopt new entrants into the DevOps product market more readily and easily. This allows the use of smaller novel vendors over established ones. Becoming an early adopter to a successful novel solution can produce significant partnership advantages as the new company expands. Effectively, a model helps replace expert gut feeling, sales powerpoints and small non-representative consultant tests with experimentation on real company projects.

Reduced Operational Risk

While implementation detail is always a reality and every staff change entails the risk of such knowledge being lost, architecture and a model connected to artifacts allows to control the risk of changes and to make predictions of impacts.
If it is possible to know how many projects use a certain DevOps vendor, and within that, a certain feature offered, the impact of dropping the vendor and moving can be predicted and calculated. If a certain DevOps product is found to be implicated in security-relevant issues, e.g. supply-chain attack, the model can be consulted and mitigations can be approached in a targeted manner.

Given the high cost of IT insurance, such best practice can be audited and can contribute to an argument for a significant reduction in ongoing insurance costs.

Team reporting - Plotting a course based on position

The Challenges

Agile practices have taken hold and accelerated with the implementation of tools that connect planning in a traceable way to implementation and deployment through the structures of databases. CI and DevOps projects express these links in their user interfaces and usually through underlying APIs. This is true of the long-term leaders like Atlassian, side-entrants like Github, new-comers like Gitlab and all other players in the CI and DevOps market. For teams, access to this information empowers them to understand where the well-managed points and shortcomings of their process are. For any software process to achieve any more than a repeatable level, execution information needs to be captured in such a way that the team can reflect on it.

Challenge 1: Understanding where you are

As projects are ultimately specific to a domain, the off-the-shelf reporting and data structures ultimately do not align with a team's needs, as it aims for quality. This usually leads to painful shoe-horning of the team's specific needs into the rigid framework of the product's data structure.

For example, tests may be running too long or too frequently but catching few issues, while the real issues lie deeper. A typical case are functional tests encoded as user interface tests. Duration of UI tests is not usually a property of general CI systems, because it is specific to UI-based projects.

Without capturing the salient properties, the team does not know where it stands with regard to quality.

Challenge 2: Plotting a course

Often the data captured in the inadequate general structures of the vendor's framework is base data, but what is required is the ongoing production of derivative data and usually the conversion into control inputs.

A team may have a weight matrix for their components and would apply priorities and test intensities based on these. Without additional tooling, they would not be able to automate these specific project needs.

Likewise, if the team would capture test quality and user feedback, these would be challenging to aggregate meaningfully with the vendors general facilities.

A team left without the ability to make predictions and plan with derived specific models is left to discover outcomes by testing. However, unless the system's architecture is very well defined, this step is often slow and expensive.

Tests are partial and live on the 'right' side of the development process, while software engineering aims to make processes and tools that shift verification and validation to the left, to fail early.

Without engineering the process and process data and just generic data capture tools, this is very challenging to accomplish.

Challenge 3: Get a bigger map

For company-level management, as the setter of directions, the needs and trends at the actual team level are an important concern. If there is a sound trust relationship between company level and teams, examining objective trends can help to decide deployment of complex tools that need substantial infrastructure and training. For example, fuzzing is known to be a highly effective testing practice, but has low adoption in most projects because its barrier to entry is high.

The bigger map will often require the management to either become technically more proficient or to place more trust in the teams to seek support with resources provided. Reporting of actual useful numbers can drive the bigger map.

Without truthful, rich and adequate reporting, management is unable to assess the situation and assist.

The MDE Approach

Using MDE, we would design a domain-specific language, which enables teams to express typical queries against their processes in a way that attaches directly to the underlying artifacts of the CI/DevOps product, such as stories, tickets, epics, incidents, deployments, branches, tags, commits, releases and the feedback that the underlying processes produce, like test outcomes, coverages, timings and other metrics. That information is captured in a versioned database system like the Eclipse CDO™ (Connected Data Objects) Model Repository.

The Benefits

Powerful reporting and analysis with every commit

Model-driven approaches provide teams a way to succinctly express their own model of quality and attach it to that of the off-the-shelf DevOps product where there is a semantic match. Model-driven toolchains use these models for queries to render reports. At this stage, all model-driven practices such as validation, advice generation, transformation and reporting become available.

Validation ensures that business rules that pertain to a client's model of a state of business or work item can be checked. Advice generation uses rules to suggest improvements to a model to the modeler maintaining it. They allow lowering the level of entry for modelers, Transformation allows the use of rules to turn one model into another, mix multiple models together and upgrade versions. Transformations allow one to convert a model of one aspect to produce data for another. They allow the team's own model to be exported into models in other modeling languages. For example, if the team has a way to express its data in UML or TOGAF, transformations allow the export into models based on these languages. System and Enterprise architects can then use them unambiguously. Reporting produces textual, graphical or tabular summary information about the content of one or more models, supporting discovery and the making of decisions. All of these practices are usually required in model-driven projects on a larger scale.

Single model representation covering all performance needs

The advantage of the model-driven approach is that all items of analysis are presented in a conceptually homogenous way and can be linked easily. They can be prototyped with little effort in interpretation and, if required, optimized for higher performance by compilation.

This does not apply to approaches that use databases or other data stores. Only model-driven technology combines the strength of schema-based languages with linking of instances, ad-hoc storage and the ability to optimize performance.

Incremental adoption: Start small, take the first hurdle, grow

While the solution section above talked about all the sources reporting may ultimately draw data from and the potential for growth and acceleration, teams can start with a simple daily report that benefits them. Tools like Epsilon allow easy interfacing with the standard artifacts of business, such as CI system reports in XML, JSON and YAML, and spreadsheets in CSV and Excel format. Teams can lean on this ad hoc data support to implement initial steps within the small time budget usually available for process changes to use the results to argue for more internal time investment.

Adapting the process based on data

Using the succinct and project-specific input in the model, new metrics can be defined that reveal underlying obstacles. This allows the process to be changed to address these issues early (shift left), so they do not cause the wedge effect of great cost later.

For teams that know their position based on data, there is the opportunity to take advantage to revise the queries that produce it until the underlying challenges are clear and the process improves. Plotting the course can also involve discovering "black spots" where the team has process challenges that it cannot cover with its own expertise.

Quality and compliance mapping

The Challenges

For team management and sales, as the external interface of a team, and for a company as a whole, casting process information in such a way that it shows compliance with industry standards is a great way to communicate the quality of outcomes. Domain performance standards like GDPR (EU protection of privacy on the internet), PCI DSS (credit card payment), SOX (banking accountability legislation), HIPAA (health insurance data legislation), FIPS (U.S. federal information processing standard) and FISMA (U.S. federal information security management) and software process and quality standards like TOGAF, ISO 27001, 29119, 12207, 27034, 62443 and 9001 are all either required or desirable properties of software or software services for purchases in the software market.

Challenge 1: Not knowing where to start

Showing compliance is often tedious because there is no internal structure to map the external elements of compliance to, leading to paper tigers of procedures stuck in a TBD state for most projects. This situation makes it hard for companies to get started with compliance mapping, leading to a concentration of large vendors that achieve certification and limited competition from small and medium-size companies (SMEs) in bidding where compliance is required.

Challenge 2: Non-functional as an after-thought

Compliance tends to involve a number of non-functional aspects of software: for example the aspect of what data is being logged, how it is being transported to log output and who the logs will be available to. These aspects are often omitted in the solution and feature-oriented design of systems and are hard to retrofit once implementation is notionally complete. As a result software solutions can fail compliance and accreditation despite superior quality of the underlying software in terms of features and usablity.

Challenge 3: Continuous updates

Accreditation projects are often undertaken at considerable cost involving external consultancy. The accreditation of the software or service is often linked to the requirement that the service or product remains unaltered from the time of accreditation. This is diametrically opposed to the idea that agile software should be updated aggressively to serve the customers needs.

The MDE Approach

Software teams are usually able to consider and explain non-functional aspects of their software if this is required and requested. Likewise company management usually has a specific idea as to what accreditations they would like to pursue for the products and services to be sellable. An MDE approach would create a model that captures the aspects that the various accreditation schemes have in common and connect that model to data input in every project. The approach would further encompass appropriate model transformations that connect the software being constructed.

Reduced cost for multiple certification targets

The advantage of this approach over paper is that models are traceable and can be projected in various ways. The quality model of a company can be cast to produce documentation for various and not just a single accreditation aspect.

Live link to the software in question

By creating a company specific model that expresses the desirable accreditation needs the software that a team produces can actually be linked to the model of the accreditation that the company is pursuing.

The model would be incrementally filled by the participants in the various project teams. Audits can then follow a structure pattern that leads to actual parts of the software pertinent for the audit. for example designs for logging or authentication would be highlighted during the process of the software design and I hence easy to find into inspect.

To enable software developers to interact with the model, inline annotation in the code will probably emerge as a means to provide information to the model. For example, C# would use custom attributes while Java what use custom annotations and C++ would use compiler attributes or custom documentation tags.

Low Update Effort - Dynamic Software

The process of linking the software to the quality model and producing up-to-date projections from this model can be made an ongoing process based on the DevOps practice. This allows the company to have audit-ready quality processes at any time without sacrificing an Agile approach to software.

The live link will also surface effective means to maintain quality, and emphasize standards that have current value, while highlighting standards that are hard to implement or are behind the current practice. In this sense, this live link will encourage a company to engage with the creation and maintenance of sensible industry standards.

In the extreme, it will change a company from being a consumer of standards to becoming an influencer of standards, adding to the company's reputation through evidence of its professional practice.

3. NEXT STEPS

If you have found this interesting and would like to explore MDE further, there is help to hand.

Join us in MDENet (www.mde-network.org) to engage with an international network of experts in model-driven engineering, access learning resources, events, and funding.

About the author

Jörn Guy Süß is the founder of Committed Consulting, a software engineering consulting company focussed on DevOps, Model-Driven Engineering and Eclipse. With twenty years of experience in applying MDE to areas ranging from connected car IoT to mining simulation, traffic management and full-stack software generation, he is one of the old hands at applying MDE in business.

ABOUT MDE NET

The EPSRC network MDENet brings together research and practice in Model-Driven Engineering (MDE).
We will do this by:

Driving Future Research

We will establishing a clear understanding, shared by the community, of the current state of the art in research and the challenges at the forefront of academic research and industrial use. We also aim to identify and create opportunities for cross-disciplinary MDE research.

Training the IT Industry

We will curate and produce training material, lowering the barriers to entry for potential new users of MDE technology. These, in turn, will bring new demand and challenges for future research and development in MDE, creating demand and support for new and improved training materials.

Building the MDE Brand

Focusing on activities to increase the visibility of MDE research, and on community building. MDENet will be the home of the MDE community and the authority on MDE, opening this space to individuals and teams not previously involved in MDE

This network is a broad church. If you are interested in software development, automation, or (computational) modelling in other domains (biology, AI, robotics, finance, ...), this network will likely have something for you.

Join our community platform