# A Daily Dose of DSL
## *MDE Micro Injections in Practice*

Gerald Stieglbauer[1], Christian Burghard[1,2], Stefan Sobernig[3] and Robert Korošec[1]
[1]*AVL List GmbH, Hans-List-Platz 1, Graz, Austria*
[2]*Graz University of Technology, Rechbauerstraße 12, Graz, Austria*
[3]*Vienna University of Economics and Business, Welthandelsplatz 1, Vienna, Austria*

Keywords: Model-Driven Engineering, MDE in Industry, MDE Micro Injections, Domain-Specific Language, DSL vs. UML, Model-Based Testing, Usability, User Experience, Separation of Concerns, Data Re-Use, Abstraction, Agile Development.

Abstract: Although Model-Driven Engineering has proven to be an adequate solution for increasingly complex engineering problems, its industrial adoption still remains limited. We argue in this paper that an important factor in regard to a failed introduction of MDE methodologies is still a blurry conception and insufficient distinction between *applying* MDE conceptually and *introducing* it into legacy-dominated industrial environment. We further argue that the MDE introduction process can be significantly facilitated by the application of so-called *MDE micro injections*, especially in agile development environments. Finally, we substantiate our arguments by presenting a case study of three industrial research projects, which illustrates the effectivity of MDE micro injections.



# 1 INTRODUCTION

More than ever, industry faces the challenge of rising complexity in many application fields. While this complexity rise is even accelerating, the provided time span for the introduction of new technologies to master the current complexity levels shortens drastically. Consequently, managing complexity by introducing sophisticated but lightweight development methods and technologies is more essential than ever before. Since more than two decades, model-driven engineering (MDE) aims at providing concrete solutions, which promise finding the right level of abstraction, a separation of concerns, as well as improving reuse in model-based software. In contrast to these intentions, however, there is an ongoing debate in the modelling community: Why is industrial adoption of MDE – despite the obvious needs – still limited? Why are other mitigation strategies such as agile development methods (which are by no means contradictory to MDE) favoured to tame the perceived development complexity?

We claim in this paper that one root cause for this situation lies in a blurry conception and missing distinction between *applying* MDE (e.g. the act of modelling) and *introducing* MDE (e.g. learning a modelling language such as UML, choosing right levels of abstractions, etc.). Within an ideal environment for *applying* MDE, modelling principles have been established from the very beginning by a team of enthusiastic and well-educated modelling advocates and corresponding solutions have always been model-based. In industrial practice, however, such an ideal environment is rather exceptional. For instance, legacy practices often cause important *barriers to introducing MDE*. Socio-cultural issues associated with a methodology shift from traditional engineering to MDE further contribute to these difficulties. In addition, while agile development environments are not contradictory to the *application* of MDE, we claim however that they add to the challenges of *introducing* MDE, which are often underestimated or neglected in modelling theory. In this paper, we thus promote an MDE *introduction strategy* called *MDE micro injections*, which has been initially published in (Stieglbauer & Rončević, 2017). The intention of the method is to reduce the gap between an ideal *application* and a practical *introduction* of MDE within an agile industrial environment related to legacy solutions. Since we focused in (Stieglbauer & Rončević, 2017) more on theoretical and methodological aspects of MDE



Stieglbauer, G., Burghard, C., Sobernig, S. and Korošec, R.
A Daily Dose of DSL - MDE Micro Injections in Practice.
DOI: 10.5220/0006754406420651
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 642-651
ISBN: 978-989-758-283-7

micro injections, we are reflecting in this paper on three concrete research projects (covering a period of more than six years), which contribute significantly to the idea of MDE micro injections.

Table 1: Different aspects of MDE applied in the corresponding research projects.

| | *Application of MDE* | *Introduction of MDE* | *UX Aspects* |
|---|---|---|---|
| *TRUFAL* | ✓ | × | × |
| *TRUCONF* | ✓ | ✓ | initially |
| *DLUX* | ✓ | ✓ | ✓ |

The associated research projects all refer to the same use case of establishing an *automated test case generation process for measurement devices in the domain of automotive testbeds*. All three projects are related to MDE approaches to generate test cases but differ concerning their focus on MDE *application* vs. *introduction* (see Table 1). The central idea of the TRUFAL project[1] (2011-2014) was the *application* of UML modelling techniques on a mutation-based testing methodology. While the value of the applied methodology was successfully proven (Aichernig, et al., 2014) the industrial adoption of the modelling techniques remained limited. Reasons for this limited adoption were analysed in the follow-up research project TRUCONF[2] (started 2014). Many of these reasons pointed towards difficulties in introducing a generic modelling language for a very particular application field. Consequently, a DSL-based approach was favoured in the TRUCONF project to overcome the observed difficulties. However, it soon turned out that a good overall user experience (UX) is essential when considering the introduction of the DSL. Due to a lack of well-established methodologies for analysing the user experience for DSLs, the DLUX[3] project (started 2017) was initiated to put more emphasis on UX aspects in terms of empirical evaluations related to the MDE introduction process and the applied modelling tools.

## 2 MDE MICRO INJECTIONS IN A NUTSHELL

Agile development and continuous integration methods have shortened development cycles, so-called sprints (associated with concrete submission deadlines), from months to weeks, sometimes even days. These methods were designed to handle complexity by relying on flexible adaption of intermediate goals on a daily basis. If a concrete outcome and its practical application fails to meet the expectations (e.g. due to a lack of knowledge of the overall system), this is instantaneously recognized (e.g. by test automation) and corrected during the next iteration. This method has been accepted as common practice in today's industry, regardless of the discussion whether such an approach favours short-term aspects over long-term goals, if not carefully applied and supervised by dedicated system architects. MDE approaches should be of special interest for such architects, since corresponding abstraction layers structure the long-term goals on the one hand and simplify the implementation phase on the other hand. Many existing MDE tools, however, originate from a past golden tool era when agility was by far less dominant and waterfall approaches were state-of-the-art. A rapid tool-introduction was not at the top of an MDE tool vendor's requirement list and the acquisition of sufficient theoretical and methodical knowledge was less of a strain on the users' weekly schedules than it is today. Consequently, many MDE tools were not intentionally designed to adhere to agile tool introduction sprints. Moreover, adapting them to the changed conditions has turned out to be challenging due to legacy issues of an aged code base and a significant drop of investments on the tool market (Bordeleau & Edgard, 2014). As one consequence, the affected developers primarily perceive the introduction of an MDE tool as a threat against their upcoming deadlines rather than a relief from their actual troubles caused by raising complexity.

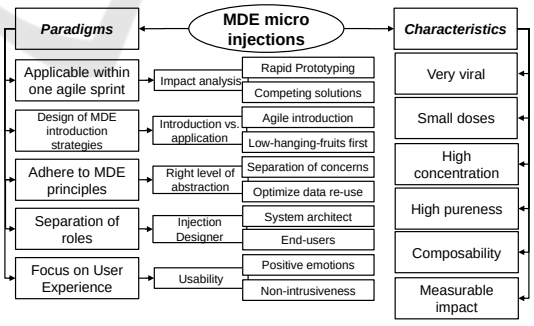


Figure 1: Paradigms and characteristics of MDE micro injections.

The intention of *MDE micro injections* is to circumvent these issues by several *paradigms* and *characteristics* which are summarized in Figure 1.

[1] https://trufal.wordpress.com/
[2] http://truconf.ist.tugraz.at/
[3] https://dlux.wu.ac.at/

One *paradigm* of MDE micro injections is to enhance classical MDE *application* scenarios by agile MDE *introduction* strategies. Another one is a clear separation of roles: potential model-driven developers (i.e. MDE end-users) versus the role of modelling advocates or - as we call them in this paper – the *micro injection designers*. Essential capabilities of these micro injection designers do not only comprise a good knowledge about modelling theory and its application fields but also includes a strategy about the agile introduction of MDE. While classical MDE theory favours *abstraction*, *separation of concerns* and *re-usability* as primary concepts, a micro injection designer has a deep understanding in these issues but focuses on other topics concerning the introduction of MDE: First, she or he takes care about splitting the MDE introduction process into tranches and ensures that each tranche can be realistically applied and completed within one development sprint of the corresponding target group of end-users. Second, the injection designer aims to maximize the overall user experience for the end-users - especially during an MDE introduction process.

We want to emphasize that the term user experience includes but goes far beyond the notion of tool usability but comprises as well socio-cultural and psychological aspects. A good user experience should minimize negative emotions (e.g. not endanger an upcoming deadline or cause fear about loss of control) but should maximize positive ones (e.g. solving a concrete problem immediately within the actual sprint).

Figuratively, the skills of a micro injection designer are reflected by the following *characteristics* of MDE micro injections: first, these injections must have a *viral effect*. Corresponding MDE approaches must become *desirable* to potential end-users so that they propagate the approach autonomously, once the injection designer has 'infected' a critical mass. A good user experience supports desirability.

Despite its infective potential, however, particular MDE injections must be given in *small doses* to minimize possible side effects such as negative emotions and other hindering social-cultural effects like the 'not-invented-here-syndrome'. This syndrome is a common reaction to not yet established innovations, which have not been created by the target group but should be realized by them. However, the syndrome can be cured by the injection designer via a careful selection of early adopters from the target group, who are the first in line to gain the laurels of success from the management in case of a successful MDE adoption.

Independent from its small doses, the MDE micro injection is *highly concentrated* to maximize its effectiveness. Instead of a time-consuming design phase for an overall cross-product and company-wide MDE approach, potential islands of success should be detected, analysed and low-hanging fruits should be prioritized. A valuable indicator of a low-hanging-fruit is if a corresponding MDE approach increases the level of automation (e.g. by using test automation). To ensure effectivity, small but coordinated teams should bring up initial solutions by using rapid prototyping (e.g. model editor generation).

However, (separated) islands of success must not contradict another characteristic: *high pureness* of an MDE micro injection should avoid any violation of MDE principles. Besides selecting the *right level of abstraction* and applying *separation of concerns* where feasible, *composability* of the various islands of success is an essential requirement, especially if the islands grow in size and prospectively merge to bigger ones. The best individual solutions will not be sustainable if they are incompatible with other ones.

Furthermore, sustainable growth of the islands can only be assured by a long-term support of the involved management stakeholders. What managers require for their support, however, are concrete numbers. Thus, the effect of each small MDE micro injection must be *measurable*. Corresponding measurements may be done by empirical evaluation (e.g. user experience studies) or by competing parallel solutions, one using a traditional approach, while the other is based on an MDE approach. The measurable comparison criteria for competing solutions may be development speed, code quality (e.g. bug fixing benchmarks, test coverage, etc.) or requirement fulfilment rate. To provide expressive numbers, the MDE-approach-related benchmark must be normalized in terms of initial overhead, which shall be taken into account during the introduction phase.

However, even if it is subtracted from the benchmark, this overhead must not be neglected and has to fulfil the overall MDE micro injection paradigm: a single injection action and analysis of its impact must be *feasible within one development sprint*. Otherwise, the effectiveness of the MDE micro injection will be significantly diminished independent from the chosen concentration factor: the MDE prototype solutions will be significantly inferior to the previously established solution and the expressiveness of the measured impact is degraded and blurred again.

# 3 USE CASE: MODEL-BASED TEST CASE GENERATION FOR MEASUREMENT DEVICES

AVL is the world's leading company in providing instrumentation and test systems to the automotive industry and other domains in form of automotive testbeds. A testbed is a complex system of systems exhibiting a tremendous variety, depending on the customer's requirements. For instance, a testbed for engine testing consists of more than 70 different subsystems, including measurement and conditioning systems for fuel, air, oil, exhaust, coolant and electricity. Consequently, each testbed variant needs to be tested extensively to eliminate any side-effects caused by the integration of its subsystems.

The use case addressed by the mentioned research projects covers a specific class of subsystems, i.e. *measurement devices*. These devices are used to measure specific quantities of the unit under test (e.g. a combustion engine), for instance exhaust gas concentrations or fuel consumption. All subsystems of the testbed are controlled by a testbed automation software called PUMA Open. For each new software release, about 50 different measurement devices undergo various integration tests to ensure compatibility with the testbed automation system. This is traditionally done by applying a series of test suites to the PUMA Open software objects which encapsulate the connection to the individual measurement devices. A more detailed description of the measurement device testing process can be found in (Auer, 2014). For each individual device, the testing process requires a minimum effort of one person-day. This relatively big effort is partially caused by the issue that test suites are mainly written and maintained at coding level (i.e. using Visual Studio and NUnit), which burdens the test engineers with the time-consuming and error-prone tasks of test suite maintenance and ad-hoc test coverage analysis.

We aim to mitigate these shortcomings by introducing a model-driven testing methodology to the measurement device testing process. The testing methodology is based on a mutation-based testing approach, which has been developed[4] during the TRUFAL project (Aichernig, et al., 2014) and was further improved by the TRUCONF project (e.g. by including testing of non-functional requirements). The methodology requires the test engineer to create a behavioural model of the measurement device under test. The model serves as an input to a mutation-based test case generator, which is used to generate the NUnit test suites per a mutation coverage criterion. Due to this methodology shift, we intend to significantly decrease test suite coding and maintenance effort, while increasing test quality and coverage using mutation-based testing.

[4] Development on the model-based testing approach started in an earlier project, in which AVL was not involved. See http://www.mogentes.eu/

# 4 EVALUATING THE TRUFAL PROJECT AGAINST THE IDEA OF MDE MICRO INJECTIONS

The TRUFAL project (2011-2014) was a research project, which aimed at the development of an efficient model-based test case generation methodology for highly complex systems. On AVL side, measurement devices were chosen as case studies. The focus was on efficiently finding implementation faults, which violate the functional specification. At the time when the project was started, functional testing was done either by performing manual tests on corresponding user front-ends or by writing unit tests on the level of the device's communication protocol. Testbed components such as measurement devices are not manufactured in form of mass production but are high-end devices for professional industrial use, available in many variations and combinations. Consequently, device testing is an expensive task in general, especially if combinations of variations of these devices must be tested. To make this kind of integration tests affordable, AVL has already developed device simulators before the TRUFAL project was started. However, it was the intention of the TRUFAL project to elaborate on a next step regarding cost reduction and efficiency improvement after device virtualization and simulation: On the one hand, manual testing should be superseded by an automated method, whereas each test is reproducible at any time, e.g. if a new variant of a device has been introduced or a different combination of device variants must be integrated. On the other hand, a model-based approach was introduced to raise low-level unit test programming to a much higher level of abstraction. Test models were intended to encode the intended functional behaviours from which a test case generator derived the (formerly manually written)

unit (or integration) tests. To improve the efficiency of this model-based testing approach even beyond the classical introduction of abstraction, the principle of mutation-based testing was applied. Here, the test models are slightly modified by a mutation algorithm. Afterwards, the test case generator produces test sequences which uncover observable deviations in the behaviour of these mutants. The intention of this approach is to improve testing efficiency by increasing the test coverage rate for the generated test cases in comparison to a more straightforward mapping of the test model to a test suite. More details can be found in (Aichernig, et al., 2014), where the approach as such has been verified successfully. For instance, the quality and efficiency of several types of auto-generated test suites has been evaluated and a more efficient mutation-based test case generation algorithm with a significantly reduced runtime has been developed (Jöbstl, 2014), (Aichernig et al. 2015).

Despite its illustrated usefulness, however, the industrial adoption of the approach remained limited. An obvious indicator was the low number of modelled devices during the project and the amount of time needed to correctly specify a device model. For instance, the creation of an initial model took roughly 12 hours, followed by an additional 40 hours for fine-tuning and bug-fixing. This estimate does not include the effort of familiarizing oneself with the overall model-based testing approach, as well as the modelling tools, which took 6 additional hours. This was despite the use of a standard modelling language (UML) and a standard compliant tool (Papyrus).

In the following, we are analysing possible root causes of the limited adoption by evaluating the applied methods of the TRUFAL project against some of the MDE micro injection paradigms and characteristics. First, we raise the question if there was a clear separation of the *application* and the *introduction* of the TRUFAL methodologies. One could argue that this separation was indeed present for the following reason: a team of modelling advocates put reasonable effort on the definition of the right abstraction for device modelling to fit the requirements of the mutation-based test case generation approach. Only due to their expertise, an *adequate level of abstraction* was found to fulfil the project's aims of automated, reproducible test-case generation with reasonable test coverage. So, the baseline for a successful application was set. However, is such a baseline enough for a successful introduction?

To answer this question, we examine the *desirability* of the approach. In case of the TRUFAL project, this examination remains inconclusive, since the end-users have neither been interviewed, nor were otherwise involved in the design phase. Instead, they were confronted with model structures, which fulfil the requirements of the test case generator to enable a successful evaluation of mutation-based test case generation methodology but not necessarily reflected the test engineer's way of thinking. This led to scepticism against this approach, which is quite the opposite of *desirability*. One could argue that corresponding modelling courses would bridge this gap, another could argue that it is still advisable to minimize the gap by a more suitable language design. However, since this was considered to be outside the scope of the TRUFAL project, the corresponding MDE injection has turned out not to be very *viral*.

If the MDE injection was not *viral*, did it at least fulfil the characteristic of *small doses* to avoid negative side-effects? In case of the TRUFAL project, the test models have been based on UML state machines and class diagrams, which were implemented in the UML modelling tool Papyrus. However, it was as well beyond the scope of the project to analyse, which subset of these UML diagrams would be sufficient for test case generation (e.g. by applying a UML profile) or how Papyrus needs to be tailored (e.g. limiting menu entries to those needed by the involved diagram types). Consequently, the end-users were confronted with a full-fledged UML tool. Due to the feature-richness of both the tool and the UML language they had to learn the semantics of UML first and map their intuitive view of individual device tests to an appropriate UML representation using the correct tool features. Additionally, the underlying test case generator required some additional model features (specifically, class diagrams) to specify the test interface. The end-user had to be aware of all these issues in order to create a test model suitable for test case generation. Any mistake led to an obscure failure of the test case generation process, the root cause of which was hard to evaluate. Due to limited *user guidance* by the model editor, considerable efforts were needed to train the end-users, which contradicts the MDE micro injection characteristic of a *small dosage*. Incorrect models and misinterpretations of model semantics caused frustration and led to the subjective impressions among the test engineers that MDE *introduces* complexity rather than *diminishing* it.

Due to the lack of a UML profile, it could be argued that defining one would have avoided this situation. However, defining such a profile does not intrinsically imply that the right elements are selected and, even if so, that the semantics of the profile

becomes understandable out-of-the-box to the end-users. Furthermore, even if the profile seems to be appropriate objectively, the end-user's subjective impression may vary significantly. Related to our use case, test engineers have to rely on a particular amount of information about a measurement device (e.g. device documentation, requirement agreements[5]). It may be quite difficult for them to perform the mental process of translating this information to the semantics of a correct model, even if a UML profile has been applied. If this translation process is not considered during languages design, the efficiency of the modelling process decreases significantly. At this point, the importance of a careful domain specific language design becomes apparent. According to our experience, it is not sufficient *that* a UML profile (or any other DSL variety such as a grammar for a textual DSL) is applied but *how* such a profile is designed (in close collaboration with the end-users) to create 'real' domain-specific languages.

Since a UML standard compliant tool such as Papyrus was used, one could argue that the MDE injection must be at least *very pure* and no MDE principles could have been violated. Nevertheless, we observed several MDE principle violations. If considering the principles of a *proper abstraction level* and *separation of concerns* with the intention to reduce complexity, the resulting measurement device models were surprisingly complex - not only in terms of comprehensibility but also in terms of their size. Consequently, even a modelling advocate, who was actively involved in the research project, had to spend considerable time to produce a suitable device model. One reason for this was the involvement of several interdependent state machines. In case of the interdependent state machines, a state switch in one state machine may cause a series of state switches in other state machines. Although these interdependencies were following a certain repetitive scheme, they were not classified as primary language constructs. Instead of separating them from the remaining model, these relations have been modelled rather implicitly and through several model elements spread over the entire model.

Apparently, we failed to meet almost every characteristic of the proposed MDE micro injections in the TRUFAL project, which in our view caused the very limited practical adoption despite the proven usefulness of the model-based testing approach. Since the applied MDE injection was neither *viral* nor small in terms of a *micro* injection nor *highly concentrated* nor *very pure*, the *measurable impact* was very low or even non-evaluable, not only because we tried to replace a complex traditional methodology with a complex modelling approach, but also because the introduction of the chosen approach was by no means feasible within a corresponding development sprint. This not only prevented concrete empirical evaluations of the practical impact of the approach, but made a comparison of the traditional and model-based approaches impossible: Keeping the model consistent with the weekly updates of the traditional approach was hard to achieve. Consequently, the model-based approach always remained inferior to the traditional one, which negated its claimed effects (such as complexity reduction) and promoted scepticism among the potential end-users.

# 5 APPLYING MDE MICRO INJECTIONS DURING THE TRUCONF AND DLUX PROJECTS

## 5.1 Establishing the Role of a Micro Injection Designer

After a successful evaluation of the TRUFAL test case generation methodology itself (independent from its industrial adoption rate), the TRUCONF project (started 2014) is intended to enhance the applied methodology by supporting non-functional requirement testing (e.g. by including performance constraints) on the one hand but also has a strong focus on overcoming the limited industrial adoption of the TRUFAL results by applying some of the paradigms and characteristics of MDE micro injections. A dedicated role of an *MDE micro injection designer* was established in the TRUCONF project.

The injection designer began a process of acquiring the end-users' domain knowledge more deeply through close but non-intrusive interaction. This action was strongly supported by the team leader of the test engineers, which turned out to be essential to establish short meetings on a regular basis with motivated participants. The injection designer soon found out that many test engineers had a similar structure in mind when they were designing test cases. Due to a lack of modelling skills, however,

[5] This information may be created by another department and (due to several company-depended restrictions ) cannot be modified to better comply with the intended model semantics.

they were not able to formalize or communicate this in form of a precise semantics. Due to the obvious gap between the model semantics applied in the TRUFAL project and the input gained from the test engineers, the injection designer investigated, which publicly available model languages had the potential to diminish this semantic gap. It turned out that the language Gherkin[6] was a good candidate for the use as a language baseline. In one particular case, it even turned out that a test engineer invented languages similar to Gherkin to keep track of the tests with pen and paper.

The injection designer continued closing the gap by further interviews through which he evolved an initial version of the so-called Measurement Device Modelling Language (MDML, more details can be found in (Burghard, et al., 2016)). Each sprint period, the injection designer came along with a concrete modelling tool prototype for MDML, which has since been updated for each iteration. This prototype made a rather abstract semantics more obvious to the end-users and initial hands-on sessions gave them the experience of a deep language design involvement, where fears (e.g. of a loss of control) were no longer an issue. Due to the short duration of each iteration, the Eclipse Xtext framework[7] was used to create rapid prototypes of a textual DSL editor. Using the Xtext core features to generate a model editor, a data model and a model parser led to already quite mature results (e.g. syntax highlighting and code completion), which were very attractive to the end-users.

The rapid prototyping facilities of the Xtext framework significantly supported the step-wise introduction of MDML using *tiny doses* of MDE injections at an early state of the collaboration, which turned out to be *very viral*, since the test engineers felt involved but not threatened by the evaluated MDML approach and continued on their own to spread the discussed ideas to other test engineers. Rapidly changing and updating the prototypes according to their input was an essential catalyst and important to establish trust between the injection designer and the test engineers. This trust was strengthened due to the fact that the prototypes gave them a concrete impression of the future implementation of the model-based testing approach, as well as of the straightforward and understandable tooling.

## 5.2 User Experience on Spot: DLUX

The positive feedback of the test engineers during this early design phase emphasizes the importance of *user experience* (UX) aspects during the introduction of MDE approaches. As sketched in the previous section, user experience aspects go far beyond tool *usability*. Although tool *usability* was already kept in mind, simplifying the mental mapping process from existing workflows to a corresponding model representation was the essential breakthrough in terms *virality* and *desirability*. To address user experience systematically, beyond the scope of TRUCONF, led to the DLUX project (User Experience for Domain-specific Languages). DLUX has a two-fold purpose: First, it should be examined whether a generic evaluation-method kit for assessing and improving the *user experience* of DSLs can be developed in support of MDE-related activities at AVL (including TRUCONF). Second, the evaluation-method kit should be applied on selected use cases (MDML). Regarding TRUCONF, the selected use case was the modelling and test case generation process of a particular measurement device[8] using MDML. For this use case, the following sub-aspects of UX have been investigated:

(1) How do the test engineers locate and collect the required information about the involved device? How do they judge the relevance and quality of their data sources?

(2) How do the engineers map the collected information mentally to their (potentially sub-conscious) semantics and workflow structure? What are the perceived difficulties of this mapping process?

(3) How are their semantics and workflow structure represented in the current MDML prototype?

(4) How do they perceive the usability of the concrete MDML tool prototype?

(5) Regarding the generated test cases, is the end-user sufficiently informed about their properties and the related test-case coverage?

(6) What happens if test case generation fails? How likely is a discontinuity of the modelling process and a disruptive fallback to manual testing?

(7) How can the cause of the failure (e.g. due to a tool bug vs. due to an invalid model) be examined? Which kind of support should be established to overcome the issues successfully?

[6] https://cucumber.io/docs/reference

[7] https://www.eclipse.org/Xtext/

[8] Specifically, the AVL Micro Soot Sensor (AVL List GmbH, 2015)

(8) How can sufficient test reports be created, once test case generation and execution has terminated successfully?

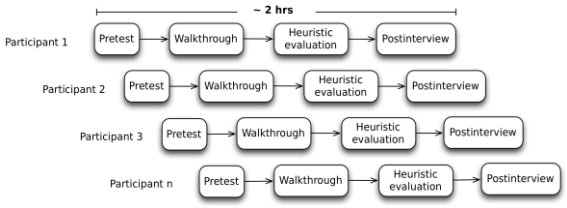


Figure 2: Series of heuristic walkthroughs in DLUX.

At the time of writing this position paper, the issues 1) to 6) have already been partly evaluated in the DLUX project. The evaluation has happened through a series of *heuristic walkthroughs* (Mahatody, et al., 2010) of the MDML prototype, with test engineers taking the role of tool evaluators. The 2-hrs walkthroughs consisted of pre-tests, a task-based tool walkthrough, a tool assessment based on UX heuristics, and a post-interview (see Figure 2). The activities were recorded, transcribed, and analysed. The results were forwarded to the injection designer, on the one hand, but also to the interviewees to strengthen their involvement, on the other hand. Overall, the modelling tool and language was well received regarding user guidance vs. degrees of freedom, minimalistic design, flexibility and efficiency. However, the evaluators expressed the need for improved documentation, authoring support, validation, and error mitigation. Furthermore, means to access materials (e.g., device handbooks) required during the modelling process should be made available from within the tool.

## 5.3 Ensuring Composability, Data Integration and Re-Use

In this section, we elaborate on selected intermediate DLUX results and examine their relations to MDE micro injections. We established *high pureness* as a characteristic of MDE micro injections, as well as *right level of abstraction*, *separation of concerns*, *data re-use* and *composability* as corresponding requirements. When examining UX aspect 1) ("How do test engineers collect the required information?"), it turned out that test engineers often do not have the necessary access to primary data sources on measurement devices and/or prefer other sources, which are easier to access but provide lower quality in terms of completeness and correctness. In case of the measurement devices, this led to inconsistent naming conventions for state labels and state dimensions, incomplete and/or contradictory specifications in edge cases as well as a prevalent sense of confusion about the correctness of particular device models or tests.

The needed device information is mostly created by the device-firmware developers. However, this information source is barely accessible for the following two reasons: First, information is commonly kept deep inside the firmware code and the corresponding documentation as a common source of information is not always kept perfectly up-to-date. Second, integration test engineers and firmware developer work in different departments, which turned out detrimental to communication flows. The MDE efforts in TRUCONF and MDML form part of a broader consolidation process towards inter-departmental data exchange and reuse (Stieglbauer & Rončević, 2017). A so-called Device Knowledge Base (DKB) was established. The DKB contains primary information such as device states and device commands, which needs to be shared across departments. In this case, the requirements for sharing data drives the definition of the meta-model of the DKB and lead naturally to an adequate level of abstraction. Public interfaces to the DKB based on this abstraction (e.g. to enable data web-frontends) give the test engineers direct access to primary device data, rather than having them resort to secondary data sources (manuals). The DKB turned out to act as a promising and straightforward method of finding the right level of abstraction for the involved models and the meta-model design.

Regarding the MDML tool, automated data integration and re-use was established in the following way: on the one hand, the MDML editor allows for creating model skeletons. These include an interface definition based on the device's name, firmware version, commands and states. In addition, auto-completion and tool-tip information can be provided. This is because, the DKB provides a certain amount of semantic information (e.g. the purpose of a particular command). These improvements are likely to increase the efficiency of writing device tests since the UX evaluation in DLUX has shown that test engineers spend more time on searching and navigating data sources than on the actual modelling task.

In DLUX, micro injections are applied in parallel to a second use case relevant for firmware development (a detailed description of this use case would go beyond the scope of this paper). This parallel application matches the idea of *separation of concerns*, since the model design for both areas focus on specific needs of the corresponding end-users to

maximize their UX: consequently, firmware models are partly comprised of different model components compared to test models but also consist of common elements. If both approaches thus remain entirely separated and – even worse – would be incompatible due to using different semantics of their common language constructs (such as device states) the MDE micro injection characteristic of *composability* would be violated. Therefore, elements such as a device state used in both models are based on the same semantics (e.g. as defined for UML state diagrams). Furthermore, it was ensured that these common elements are editable only in one model (the primary source of truth), whereby references to them are used in the other model. In case of device states, the firmware model is the primary source, while the test model uses states only as references. Corresponding activities of the involved injection designers and cross-department system architects are required to ensure this compatibility. Only then, composability of MDE approaches becomes guaranteed, which enables effective and even automated data re-use and data forwarding across company departments.

### 5.4 Success Factors and Measurable Impact

In this section, we sketch further success factors derived from improving the UX aspects (2) – (4) and their measurable impact, where applicable. Through the iterative coordination with the test engineers, the language meta-model and semantics had soon settled into a form which was well suited to their understanding of the application domain, as well as to their intuitive modelling workflow.

First, they collect information about the involved device states and device commands for a specific measurement device. In the ideal cases, this information can be entirely imported from the DKB at the time of model creation as described in the previous section. In order to complete this information in alignment with their mental model of test case definitions (2), they look for corresponding state machine and/or command descriptions in related requirement documents and user manuals. Based on these two sources, the test engineers populate a corresponding MDML model with the corresponding behavioral information (3). They experience this process as quite straightforward, since - due to the design of MDML – populating the model is feasible in small incremental steps and meaningful test cases can even be derived in early states of the modelling process. The only significant impediment is caused by contradictory information still present in some of the non-re-usable source material. Generally, the test engineers perceive UX of the language and tool prototypes in their current state to be well-tailored to their purpose, but still flexible enough to ensure an efficient modelling process (4). The efficiency is further increased, if the tooling would actively suggest meaningful model examples depending on the given device. For instance, a model knowledge base would enable corresponding user suggestions based on previously created models, which adhere to a similar device class or variant.

Due to the minimized semantic gap and proper user guidance, we achieved a tremendous increase in modelling speed. An initial device model can be created within *1-2 hours*, rather than in *12 hours*, as evaluated during the TRUFAL project. Even without a previous expertise in the application domain, the language and tooling still exhibits an adequate learning curve. In our experience, new test engineers can be introduced to the modelling methodology in a matter of *1-2 hours*, at which point they are qualified to write suitable test models. This is a significant improvement over an estimated *6-hour* introduction phase within the TRUFAL project.

Due to this effort reduction, the acceptance rate of the test engineers has been significantly improved. More or less autonomously, our current portfolio of measurement device models has been steadily grown over the course of the TRUCONF project. Fine-tuning these models is still a necessary part of the model life cycle. However, we expect the necessary effort to be greatly diminished in comparison to the TRUFAL workflow. With the previously described results, we consider the main obstacles of the TRUFAL project as eliminated. We are convinced to be on the way to a good industrial adoption of our model-based testing methodology and if further issues should arise, the continued practice of MDE micro injections gives us the agility and flexibility to counter them.

## 6 CONCLUSION AND OUTLOOK

In this paper, we argued that the lack of methodologies for a *step-wise introduction* of MDE approaches within an *agile industrial context* is a common barrier to MDE adoption. We further illustrated how the application of *MDE micro injections* help to overcome this lack. We showcased how failing to adjust for MDE micro injections has contributed to rejecting MDE in an industry research project (TRUFAL). Namely, the principle of *selecting the right abstraction level* had been violated

by attempting to fit the problem domain onto an existing general purpose modelling language, which was able to express the problem, but not in an elegant way, rather than designing a language that targets the end-user modelers. Furthermore, the introduction of the modelling method had been attempted in one large step, rather than in *tiny doses*, spanning several short iterations. This left little room for responsiveness to end users' needs, which resulted in a *non-desirable* modelling language and tooling.

However, the application of MDE micro injections helped to ensure a successful introduction of MDE methodologies within the TRUCONF project. Here, the modelling approach was designed to fit the test engineers' needs from the start to make it *desirable* and let it become *viral*. Giving them control over the language design through regular but non-intrusive interviews ensured a minimized semantic gap and resulted in a steep learning curve. This way, the test engineers gained trust in the new approach and subsequently became fascinated with it. Choosing the *right level of abstraction* early on helped us to incorporate re-usable data and enable *composability* of different MDE approaches across various company departments. Structuring the development process into many, shorter iterations kept it on the right track through repeated feedback and ensured that the test engineers received new information and improvements in *tiny doses* rather than being overburdened by introducing all aspects of MDE at ones.

User experience (UX) has turned out as a key factor to ensure end user acceptance and is essential for a *seamless and step-wise migration* from legacy to model-based approaches. However, there is still room for improvement from a UX point of view for the mentioned use cases: For instance, the test engineers must still perform manual steps to incorporate generated test suites into the test automation system. Future versions of the model-based testing tool are planned to automate the step of test suite submission, trigger test executions and play the results back into the model, which should give the test engineers a hint about the uncovered errors. In addition, the current version of the MDML models is purely textual. Although attempts at the definition of an appropriate graphical representation to extend the textual one have been made, developing a definitive solution including acceptable tool support turned out to be infeasible within the TRUCONF project.

In future work, we will systematically collect stakeholders' feedback on the practise of MBE micro injections at AVL. In addition, we will review related IDE and DSL evaluation methods (e.g., USE-ME, FQAD) for inclusion into revised MDE micro injections.

## ACKNOWLEDGEMENTS

*TRUFAL* was funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "FIT-IT - Trust in IT Systems" between Mar 2011 and Feb 2014.
*TRUCONF* and *DLUX* are funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between Nov 2014 – Jan 2018 and Feb 2017 – Mar 2018, respectively.

## REFERENCES

Aichernig B.K., Auer J., Jöbstl E., Korosec R., Krenn W., Schlick R. & Schmidt B.V., 2014. Model-Based Mutation Testing of an Industrial Measurement Device. *International Conference on Tests and Proofs. Springer*, pp. 1-19.

Aichernig B.K., Jöbstl E. & Tappler M., 2015. Model-based mutation testing via symbolic refinement checking. Science of Computer Programming, 97:383-404, 2015.

Auer, J., 2014. *Automated Integration Testing of Measurement Devices*, Graz: Graz University of Technology, Institute for Softwaretechnology.

AVL List *GmbH*, 2015. *Product Guide MSSplus - AVL Micro Soot Sensor plus.* Graz.

Bordeleau, F. & Edgard, F., 2014. Model-Based Engineering: A New Era Based on Papyrus and Open Source Tooling.. *OSS4MDE@ MoDELS*, pp. 2-8.

Burghard, C., Stieglbauer, G. & Korošec, R., 2016. Introducing MDML-A Domain-specific Modelling Language for Automotive Measurement Devices. *Joint Proceedings of the International Workshop on Quality Assurance in Computer Vision and the International Workshop on Digital Eco-Systems*, pp. 28-31.

Jöbstl, E., 2014. *Model-based Mutation Testing with* Constraint *and SMT Solvers*, PhD thesis: Graz University of Technology, Institute of Software Technology.

Mahatody, T., Sagar, M. & Kolski, C., 2010. State of the Art on the Cognitive Walkthrough Method, Its Variats and Evolutions. *International Journal of Human-Computer Interaction*, 26(8), pp. 741-785.

Stieglbauer, G. & Rončević, I., 2017. Objecting to the Revolution: Model-Based Engineering and the Industry-Root Causes Beyond Classical Research Topics. *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 629-639.