VERIFICATION
VALIDATION
METHODS

Deliverable D09

Requirements on Simulation Systems and Models Posed by a Criticality Analysis for Automated Driving Systems

Version 1.0

Editor Dr. Christian Neurohr

Project coordinator Robert Bosch GmbH und BMW AG

Due date 31.08.2023

Creation date 30.08.2023

Publication 31.12.2023

Approval INT, LI, PU

INT: Release within the project, partners, scientific subcontractors and project sponsors (concerns the application of concepts)

LI: Release VDA-LI, other projects of the project families and members of the VDA lead initiative (concerns conceptual results)

PU: Release to public

PEGASUS FAMILY

Gefördert durch:

Bundesministerium für Wirtschaft und Klimaschutz

aufgrund eines Beschlusses des Deutschen Bundestages

Dokument Information

Authors

Christian Neurohr – DLR e.V., Institute of Systems Engineering for Future Mobility, Oldenburg
Lukas Westhofen – DLR e.V., Institute of Systems Engineering for Future Mobility, Oldenburg
Tuan Duong Quang – TÜD Süd AG, München
Tjark Koopmann – DLR e.V., Institute of Systems Engineering for Future Mobility, Oldenburg
Roman Gansch – Robert Bosch GmbH, Renningen
Stefan Schoenawa – Volkswagen Group, Wolfsburg

Reviewer

TP2

Contact

Dr. Christian Neurohr
German Aerospace Center (DLR) e.V.
Institute of Systems Engineering for Future Mobility
Escherweg 2
26121 Oldenburg, Germany

Phone: +49 441 770507-217
Email: christian.neurohr@dlr.de

Requirements on Simulation Systems and Models Posed by a Criticality Analysis for Automated Driving Systems*

Christian Neurohr[1], Lukas Westhofen[1], Tuan Duong Quang[2], Tjark Koopmann[1], Roman Gansch[3], Stefan Schnoenawa[4]

[1] German Aerospace Center (DLR) e.V., Institute of Systems Engineering for Future Mobility, Escherweg 2, 26129 Oldenburg, Germany
{firstname.lastname}@dlr.de
[2] TÜV Süd AG, Westendstraße 199, 80686 München, Germany
{tuan.duongquang}@tuvsud.com
[3] Volkswagen Group, Berliner Ring 2, 38440 Wolfsburg, Germany
{stefan.schoenawa}@volkswagen.de
[4] Robert Bosch GmbH, Renningen, Germany
{roman.gansch}@de.bosch.com

Abstract. A methodical criticality analysis for automated driving systems exhibits numerous valuable use cases for the application of computer simulations. Naturally, these simulation use cases pose requirements, specific to a criticality analysis, on the employed simulation system and the simulation models. In this work, we gather simulation use cases coming from the criticality analysis and delineate a process how requirements on simulation can be derived therefrom. This process is then instantiated for the open source simulation platform *openPASS* by elicitation of general as well as scenario-based requirements on openPASS necessary for conducting a criticality analysis.

Keywords: Requirements, Criticality Analysis, Automated Driving Systems, Simulation System, Simulation Models, Verification and Validation

* The research leading to these results is funded by the German Federal Ministry for Economic Affairs and Climate Action within the project 'Verification & Validation Methods for Automated Vehicles in Urban Environments'. The authors would like to thank the consortium for the successful cooperation.

Table of Contents

1 Introduction

As the complexity of the problem of verification and validation of automated driving systems (ADSs) and their various subsystems vastly exceeds the stakeholders' constraints on resources, i.e. time and costs, the process of simulation is likely to play a key role in the homologation of these technologies [1]. In order for computer simulations to fill this role adequately, besides the validity issue for simulation environments and models, their properties have to adhere to the requirements imposed upon them by the methods which embed simulation in the verification and validation process. One such method is the *criticality analysis* for ADSs, which aims at eliciting a finite set of artifacts that can be used to structure the open context of the operational domain [2]. For this, a criticality analysis incorporates simulations as an important tool to explore the emergence of criticality in interesting scenario classes. The value of these simulations rests their validity, i.e. on the properties of the employed simulation environment and models as well as their interactions. Naturally, this interaction leads to the criticality analysis imposing requirements on nearly all aspects of such computer simulations.

The main contributions of this work are as follows:

- provide a detailed description of simulation use cases within the framework of the criticality analysis,
- systematically derive general requirements on the simulation system and simulation models from these use cases, and
- perform a scenario-based derivation of requirements on openPASS for criticality analysis based on the functional uses cases of the VVM project.

This manuscript is structured as follows: After the introduction, Section 2 delineates the concept of the criticality analysis and explores simulation uses cases within this concept. In Section 3, we derive general requirements on simulation can be derived from the simulation use cases withinh a criticality analysis. Subsequently, in Section 4, we instantiate the process for the derivation of requirements for the openPASS simulator. Traditionally, we conclude the document with Section 5.

2 Simulation within a Criticality Analysis for Automated Driving Systems

In this section, after a brief introduction to the criticality analysis for ADSs in Section 2.1, we explore various simulation use cases therein.

2.1 Introduction to the Criticality Analysis

A *criticality analysis* for the verification and validation of ADSs is essentially a context analysis for an abstract class of such systems [2]. It precedes the concept phase and aims at structuring the open and complex context in which ADSs are expected to operate safely by eliciting a finite and manageable set of artifacts that can be used to structure the operational domain.

As can be seen from the flowchart of Figure 1, the criticality analysis is subdivided into three branches:

- the method branch: its main goal is to identify safety-relevant influencing factors within the open context that are associated with criticality, called *criticality phenomena* (CP), and to analyze the underlying causalities resulting in causal relations for these CP,
- the information branch is concerned with providing the method branch with knowledge and data as well as the acquisition, organization and management thereof,
- the scenario branch embeds the artifacts of the method branch in the scenario-based verification and validation approach.

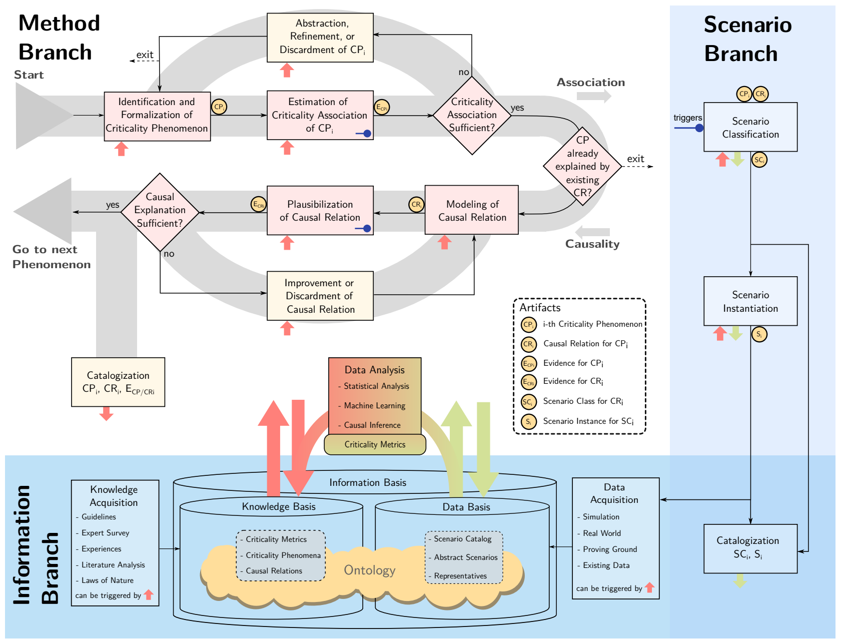


Fig. 1: Overview of the revised procedure of the criticality analysis, cf. [3, Figure 2].

Within the VVM project[5], the criticality analysis and its branches have been subject to various scientific publications that provide in-depth knowledge to the interested reader, e.g.

[5] https://www.vvm-projekt.de/en

- the concept paper 'Criticality Analysis for the Verification and Validation of Automated Vehicles' [2],
- the review paper 'Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art' [4],
- the comprehensive analysis of the GIDAS database regarding the presence of CP in accident data [5], and
- from the information branch, 'Using Ontologies for the Formalization and Recognition of Criticality for Automated Driving' [6], and
- the '6-Layer Model for a Structured Description and Categorization of Urban Traffic and Environment' [7].

In the context of simulation, let us also mention 'Simulation of Abstract Scenarios: Towards Automated Tooling in Criticality Analysis' which takes first steps towards bringing abstract scenario simulaton into practice [8]. A summary of these publications can be found in the VVM Deliverable D08 'Advances on the Criticality Analysis for Automated Driving Systems' [3].

2.2 Simulation and Criticality Metrics

To connect computer simulations with the criticality analysis, criticality needs to be measurable in a simulation. In particular, it needs to be computable from data generated by simulations runs. For this, so-called *criticality metrics* are used, cf. [4]:

Definition 1 (Criticality Metric). *A criticality metric is a function $\kappa : \mathcal{S} \times \mathbb{R}^+ \to O$ that measures for a given traffic scene $S \in \mathcal{S}$ at a time $t \in \mathbb{R}^+$ aspects of criticality on a predetermined scale of measurement $O \subseteq \mathbb{R} \cup -\infty, +\infty$. Scenario level criticality metrics extend this definition from scenes to scenarios.*

Scenario-level criticality metrics extend this definition from scenes to scenarios, i.e. adding retrospective temporal aspects to the measurement [9]. Most criticality metrics only quantify over a subset of the influencing factors that are associated with criticality, such as spatial, temporal, dynamical, perceptual, or environmental circumstances. They can be applied for several purposes along the V-model, cf. [4, Section 3]. Here, we briefly list the identified applications:

- Objective Function (A.1)
- Run-Time Monitoring (A.2)
- Identification of Risk-Reducing States (A.3)
- Requirement Elicitation (B.1)
- Scenario Elicitation (B.2)
- Testing (B.3)
- Safety Argumentation (B.4)

Many criticality metrics rely on physical quantities related to (dynamic) objects present in traffic scenes or scenarios as inputs. These inputs are are then aggregated by the metric to a single value that claims to be a quantification of an aspect of criticality. For example, the Time-To-Collision (TTC) metric, defined on a constant velocity motion model, utilizes the velocities and distances of traffic participants in order to quantify the temporal dimension ('Time-To') of criticality ('Collision').

A comprehensive overview of criticality metrics, their classification and their properties can be found on this website: Criticality Metrics for Automated Driving – which is supplementary material to the publication [4].

2.2.1 Implementation and Evaluation Criticality Metrics The usage of criticality metrics within a simulation is a key enabler for the virtual parts of the criticality analysis. Many criticality metrics are easily implemented when the required parameters can be obtained from the simulation either via live-evaluation at scene level or at the end of a scenario, e.g. using logged data. For example, the CARLA simulator offers a comprehensive Python API [10]. The evaluation of criticality metrics for a single simulation run can either be done during the simulation on scene-level or after the simulation run has finished, using the necessary logged data - on scene level or on scenario level.

In the VVM project, ZF implemented a simulator-agnostic framework for the evaluation of criticality metrics, called *CriSys* (Criticality Identification System) that can be used to evaluate a variety of criticality metrics on simulated traffic scenarios based on logged data [11]. Besides standard metrics such as TTC or BTN, several criticality metrics that originated within the VVM project have been implemented in CriSys, e.g. MerLin [12], Evasion Threat Metrics (ETM) [13], or PrET [3, Section 4.1.4].

2.2.2 Engineering, Target Values, and Calibration of Criticality Metrics If a simulator allows the implementation and evaluation of criticality metrics in general, it can, of course, be used for all kinds of exploratory tasks such as the engineering of novel criticality metrics or the adaption of existing ones. In this regard, simulation is an inexpensive tool that can be employed before validating criticality metrics on real-world data.

Moreover, for many applications of criticality metrics, adequate target values are needed. For example, when selecting critical traffic scenarios from a data set of intersection scenarios, the analyst could filter this data set by requiring that the Post-Encroachment-Time (PET) is lower than 1.5 seconds (the target value). In order to find suitable target values, depending on the criticality metric and the application at hand, scenario-based microscopic traffic simulation can be employed. A collection of target values published in the literature for a variety of criticality metrics are available online[6].

Another advantage of simulations is the calibration of criticality metrics: some criticality metrics have free parameters in their definition that need be fine tuned for the respective application, e.g. α and β in the conflict index (CI) [14].

[6] `http://purl.org/criticality-metrics`

2.3 Refinement of Criticality Phenomena

Criticality phenomena (CP) are defined as *influencing factors associated with an increase in measured criticality*, cf. [2, Definition 2]. Let us immediately mention that this association might not be causal. One of the goals of the criticality analysis is to create a collection of CP and to estimate their relevance. In order to keep this collection manageable in extent but still reach a sufficient saturation, we leverage on the concept of *abstraction.* Nonetheless, we are also interested in refining a rather abstract CP, since analyses of different causal relations for this abstract CP may be required for the derivation of specific and effective safety principles [15]. We now first describe this refinement process in general, and subsequently highlight how simulation can aid in this procedure.

For describing the process, let us use the example of 'occlusion', a CP that has various important concretizations such as 'occluded pedestrian in urban area' or 'occlusion of area in front of lead vehicle'. Whereas a valid safety principle for the first would be a reduction of speed around areas where pedestrians may appear, the latter can be addressed by increasing the safety distance to the lead vehicle by some margin. Thus, to address the abstract CP 'occlusion', at least two rather different safety principles are required. Moreover, many cases of occlusion are completely irrelevant, such as occluded but uninvolved traffic participants far away from the ego vehicle.

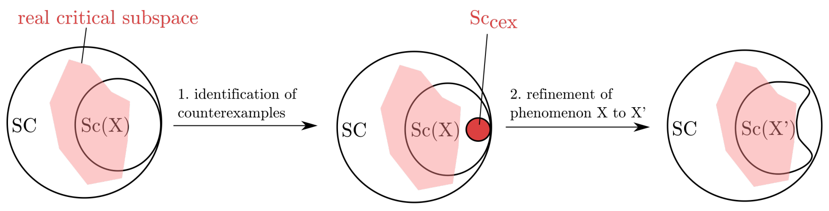
The problem is generalized as follows: Given a CP X, if X is described too abstractly it is likely not at all or only weakly associated with measured criticality. Thus, in order to uncover the potential association, we might need to describe X more concretely. For this, we propose a two-step concept of refining a given CP X:

(1) Find a set of counterexamples Sc_{cex} to the statement 'all scenarios in which X is present are significantly associated with criticality, as measured by an (aggregate) criticality metric'. Thus, we need to find a set of scenarios Sc_{cex} with X occurring who are not at all or only weakly associated with criticality.
(2) If this set exists, we use Sc_{cex} to refine X through (manual or semi-automated) refinement by looking at the *reasons* why Sc_{cex} is at most weakly associated with criticality.

Figure 2 provides a visualization of this two-step concept for the refinement of CP. Let us consider an example.

Example 1 (X = 'Occlusion').

1. We identify a set of concrete scenarios Sc_{cex} for all of which holds that there is a vehicle on the opposite lane not being perceived by the ego due to occlusion. There is nothing interesting happening, all participants just follow their trajectories.
2. We manually analyze Sc_{cex} to find that, for occlusion to be relevant, there must be a possibility of the occluded objects to have intersecting trajectories.
3. We thus refine X to the more concrete CP $X' =$ 'Occlusion of an object with a non-zero probability of intersecting the ego's trajectory' which effectively excludes the scenarios in Sc_{cex}.


real critical subspace
SC
Sc(X)
1. identification of counterexamples
Sc_{cex}
SC
Sc(X)
2. refinement of phenomenon X to X'
SC
Sc(X')

SC is the set of all scenarios

Sc(X) is the set of all scenarios with phenomenon X present

$\mathrm{Sc}_{\mathrm{cex}} \subseteq \mathrm{Sc(X)}$ is a set of counterexample scenarios not or only wekaly associated with criticality

Sc(X') is the set of all scenarios with refined phenomenon X' present

Fig. 2: Visualization of the two-step concept for refining a criticality phenomenon using criticality metrics, taken from Neurohr et al. [2].

Simulation can be of great benefit in this process, as it provides the means to generate large amounts of data for finding and analyzing counterexamples.

- In step (1), the search of counterexamples: Simulation can be used for searching through $Sc(X)$ (obviously, not exhaustively). If this search is done cleverly (e.g. by optimization of a criticality metric), the simulative search can hopefully identify a non-empty set Sc_{cex} (under the assumption that such a set exists), i.e. uncritical regions of $Sc(X)$.
- In step (2), the analysis of the resulting counterexamples: Once Sc_{cex} is identified, we need to (semi-automatically or manually) identify why those scenarios are less critical than expected. Looking at raw trajectories in plain text files is unfeasible for a manual approach, but may be useful when going towards automation. Hence, we can employ:
  - methods and tools to visualize the simulation results, e.g. by a 2D- or 3D-visualization for single runs in the set Sc_{cex}, and even a visualization of multiple runs at the same time using data visualization techniques.
  - textual simulation output to allow for more sophisticated data analysis methods to analyze the counterexamples.

2.3.1 Requirements for Criticality Phenomena Refinement In order for simulation to be useful in refining CPs, we require several features, such as searching in scenario spaces. These are listed in the following.

1. If the simulation supports an optimization/search approach, the user shall be able to specify the direction of the search, e.g. by a function to optimize. In this case, the optimization goal that we feed into the simulation could for example be to minimize the Break-Threat-Number (BTN).
2. It is not possible to enumerate all logical scenarios in which a certain CP X is present. Hence, we need a possibility of specifying that the simulation shall search through the space spanned by an abstract scenario, e.g. the set of all

scenarios where X is occurring without needing to specify anything more concretely. The method shall give coverage guarantees, as far as possible.

3. The simulation shall create a broad and highly varied data basis of scenarios where the CP X is present.
   a. In the beginning, this can be on an abstract level (e.g. through abstract models only considering a discrete position of TPs).
   b. If no counterexamples can be identified using an abstract simulation, model precision needs to be iteratively refined (e.g. through more concrete models using a continuous representation of the TPs' positions) until
      i. a level of detail is reached where a counterexample can be identified,
      ii. or the analyst decides to give up and states that there is a very low chance of identifying a counterexample.
4. The simulation shall be able to sample and simulate 'abnormal' concrete scenarios from an abstract one, i.e. those in the long tail of distributions, for example scenarios in which:
   a. Traffic participants break traffic rules or do not adhere to norm behavior
   b. Unlikely weather events occur
   c. Unusual configurations of static objects such as road geometry or traffic sign constellations are present
5. As to combine 1., 2., and 3., we may use guided simulation techniques over a given abstract scenario, where the simulation actively tries to change probability distributions of such a scenario in way that certain conditions will be met in the simulation.
   a. In this simulation use case, the direction of optimization is towards identifying scenarios in a given scenario space that are not at all or only weakly associated with criticality, thus minimizing criticality in a given abstract scenario.
   b. For this case, the employed criticality metrics should be able to identify cases with a low criticality, i.e. they may be different from over-approximating criticality measures (such as the WTTC).
6. The simulation shall provide a useful visualization of the results, e.g. using a 2D- or 3D-visualization of the resulting happenings, possibly combining multiple simulation runs in one visualization to allow for an easy comparison of runs.
7. The simulation shall enable a consecutive data analysis of the results, e.g. by providing data logging possibilities of single runs.

2.4 Instantiation of Causal Relations on Synthetic Data

Koopmann et al. propose a method to assess the causes behind such increases in criticality [15]. This process relies on data, which can also be created synthetically, i.e., from simulation. We first sketch the proposed process and then discuss how synthetically created data can be incorporated.

The central element of the process – a CP's *causal relation* – is now defined as a pair $(\mathfrak{S}, \mathfrak{C})$ consisting of a causal structure $\mathfrak{S} = (V \cup U, E)$ – constraining possible causalities to the edges – and a context $\mathfrak{C}$ – in which the causal structure is valid –, together with some additional conditions, cf. [15, Definition 2]. A generic example for such a causal relation is provided by Figure 3.

Furthermore, they define a causal relation to be *(partially) instantiated* with respect to $N \subseteq V$ by a dataset D, if the conditional probability distributions (CPDs) corresponding to the variables in N can be estimated from the data set D.

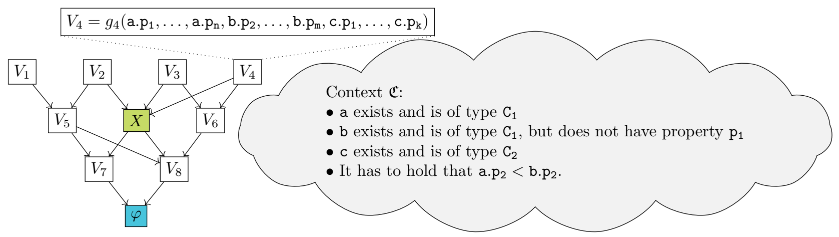

$V_4 = g_4(\mathtt{a.p_1}, \ldots, \mathtt{a.p_n}, \mathtt{b.p_2}, \ldots, \mathtt{b.p_m}, \mathtt{c.p_1}, \ldots, \mathtt{c.p_k})$
V_1
V_2
V_3
V_4
V_5
X
V_6
V_7
V_8
φ
Context $\mathfrak{C}$:
• a exists and is of type $\mathtt{C_1}$
• b exists and is of type $\mathtt{C_1}$, but does not have property $\mathtt{p_1}$
• c exists and is of type $\mathtt{C_2}$
• It has to hold that $\mathtt{a.p_2} < \mathtt{b.p_2}$.

Fig. 3: An exemplary causal relation with causal structure $\mathfrak{S}$ on the left, error terms omitted, and context $\mathfrak{C}$ on the right. X takes the values $\mathrm{Im}(X) = \{cp, \neg cp\}$, where cp corresponds to a criticality phenomenon, and φ is a criticality metric. C_1 and C_2 are classes and a, b and c are individuals in the domain ontology. The inputs of the random variable V_4 are indicated by a function g_4 taking as inputs the properties of the individuals in the ontology.

At this point, it is not clear whether we can use also artificial data-generating processes, such as simulations, or have to stick to real-world observations.

Nonetheless, let us assume we want to generate synthetic data D for the causal relation of Figure 3 for $N = \{V_2, V_3, V_4, X, \varphi\}$, i.e. for the partial instantiation regarding N by D using a simulator for automated driving, e.g. CARLA or openPASS. In order generate such a data set D, the following must be fulfilled:

1. the context $\mathfrak{C}$ of the causal relation must be realizable by a suitable, sampleable scenario class, e.g. as an abstract or logical scenario, compatible with the simulator,
2. the variables in N must be measurable (and logged) for each concrete simulation run within the context,
3. these measurements (i.e. the number of executed simulation runs) allow for the valid estimation of the corresponding CPDs of the variables in N.

Whether these requirements are fulfilled or not highly depends on the context $\mathfrak{C}$, the definition of the random variables in N, and, of course, the simulation system and models that are being used.

Therefore, there is, in general, no obstruction to the synthetic generation of data sets for the instantiation of causal relations. However, the usefulness of this approach rests heavily on the validity on the simulation. In particular for the plausibilization of causal relations, cf. [15, Sections 3.2 and 5], real-world data are indispensable. If adequate real-world data for a causal relation's instantiation and subsequent plausibilization have been collected, then no knowledge can extracted from re-instantiating the causal relation with synthetic data. At most, this can be used as a tool to validate the simulation itself regarding i) whether the underlying real-world causalities are implemented correctly in the simulation and ii) in how far the employed synthetic data generation process approximates the, likely unknown, data generating process of real traffic in the given context.

2.5 Quantification of Causal Effects

Once a causal relation has been instantiated by a data set D with respect to a suitable set of variables N, the so-called *do-calculus* can be applied to calculate various causal effects within that causal relation, cf. [15, Section 3.3]. If the graph structure and the context are fixed, these quantities are solely determined by the data set D that is used to instantiate the causal relation. In that regard, there is no advantage in using synthetic data to calculate such causal effects as the true causal effects are determined by real-world traffic data. However, a comparison between causal effects obtained from real-world vs. synthetic data may serve as a tool to assess a simulation's validity regarding the implementation of the causal relations in its context. If the synthetic and real causal effects are close, this could be taken as positive evidence for a simulation's validity in that setting.

2.6 Evaluation of Safety Principles

If a plausible causal relation for a CP has been estbalished and its causal effects are validly reproducable in a simulation system together with adequate simualtion models, this virtual setup can be used to evaluate the effectiveness of safety principles aiming to reduce either the causal effect of the CP on criticality or the exposure the CP in the given context. Such highly valid simulations that implement causal relations plausibilized with real-world date present a valuable tool for the engineering of safety principles for automated driving and have the advantage that experimental safety principles can be rapidly validated without endangering traffic participants in public testing phases. For further details on the derivation and evaluation of safety principles we refer to Neurohr et al. [3, Section 3.3].

3 Derivation of Requirements on Simulation for Criticality Analysis

In the previous section, we have seen various cases of how simulation can be applied in a criticality analysis. This usage implies certain features the employed simulation shall exhibit. Otherwise, we may not be able to uncover associations or causal effects on synthetic data, engineer and calibrate criticality metrics, or evaluate safety principles. In other words, the more relevant features are supported the more simulation becomes valuable for a criticality analysis.

But how to assess this relevancy? We propose a *requirement-based* approach that relies on exemplary *scenarios* subjected to a simulative criticality analysis. Thus, we start with a scenario class (e.g., a functional scenario) that the simulation has to (partially) generate concrete scenarios for, which is an overarching theme for applying simulation in a criticality analysis. The most basic requirement that then arises is:

(Req-Top) The simulation shall be able to sample realistic concrete scenarios from a given scenario class.

However, this top-level requirement is too abstract to be helpful in improving the simulation. We hence coarsely divide the overall simulation platform into *(a)* the simulation system (responsible for executing the simulation) and *(b)* the simulation models (implementing the dynamics of the simulated objects). Within these sub-systems of the overall simulation, useful sub-requirements of *Req-T* can be allocated.

3.1 Derivation of Requirements on the Simulation System

In order to sample concrete scenarios, we generally require a certain setup. This concerns the simulation input (e.g., taking a certain scenario description) and its output (e.g., the shape of the log files). Moreover, any simulation system exposes a user interface such as a command line or graphical user interface. Either way, the criticality analysis imposes requirements on the user interface, depending on its current process step. For example, when engineering a new criticality metric, live debugging output and visualization of metrics is desired. However, when conducting a massive sampling process for the plausibilization of a causal relation, one may be more interested in controlling the simulation via the command line and suitable configuration files. These requirements, called *general requirements*, are often independent of the scenario class considered for deriving requirements. However, some requirements on the simulation system can also be derived from the examined scenario class, e.g., matching the available input languages against the features described in the scenario class. For example, we may find that the simulation system shall support an input language which allows the specification of pedestrian crossings.

Note that when actually conducting this process, a finer distinction of the overall simulation components will be required.

3.2 Derivation of Requirements on the Simulation Models

Note that the top requirement *Req-T* requires the samples to be 'realistic'. An important group of requirements is therefore concerned with ensuring realism of the behaviors exhibited by the simulation models. In this scope, realism means that the models show behavior that would happen in real-world settings as well (and, if the models are probabilistic, these behaviors are distributed similarly). Various components of the models contribute to realism:

1. The selected parameter values or distributions for the models are realistic and match the circumstances of the scenario (e.g., using a valid distribution of bicyclist speeds in urban areas).
2. Models having appropriate reaction to their environment and adapting their planned trajectories in accordance with changes in their environment (e.g., stopping at red traffic lights).
3. Models communicating appropriately to their environment (e.g., setting turn signals).
4. The granularity and behavior of their internal sub-models (e.g., having camera and actuator models that exhibit realistic behavior in rainy scenarios).

These examples highlight that, at this point, our scenario-based approach becomes valuable: The requirements engineer is able to imagine certain realistic behaviors that may occur in the concrete scenarios, e.g., a bicyclist driving over a pedestrian crossing without dismounting. This leads to very specific, actionable requirements, e.g., the bicyclist model shall cross pedestrian crossing with a valid dismounting probability.

Besides realism, there are often non-functional requirements. This can concern performance, as the desired sample size can grow large in a criticality analysis. Moreover, if the models are not real-time capable, a real-time visualization or output is not possible, which can be required for the engineering of criticality metrics.

3.2.1 Derivation of Requirements on Sensor Models As a special case of scenario-based requirements, we consider the derivation of requirements on sensor models from criticality phenomena (CP). This is a special case as CP, such as 'Occlusion', define abstract scenario classes, cf. [2, Definition 4].

The guiding principle to derive CP-based requirements on sensor models can be formulated as 'What properties of the sensor model are necessary so that the effect of the CP is visible?'. Implementing such properties in the sensor model greatly enhances their value to the criticality analysis and validity.

Table 1 sketches some examples for CP-based requirements on a phenomenological respectively physical camera model. If resources are available, this could be easily extended to cover many other CP as well as other sensor technologies such as lidar and radar sensors.

4 Requirements on openPASS for Criticality Analysis

In this section we illustrate the derivation of requirements on simulation for criticality analysis in the concrete case of the open source simulation platform openPASS [16]. Within the VVM project, the development of openPASS is facilitated as explained in Section 3 in order to realize the simulation use cases of the criticality analysis, cf. Section 2. Therefore, after a brief introduction to openPASS in Section 4.1, we elicit requirements for criticality analysis on openPASS, both in general and scenario-based, in Section 4.2. We close the section with

Table 1: Exemplary criticality phenomena-based requirements on a phenomenological respectively physical camera model (CM) for criticality analysis.

ID	CP	Required Effect on Phenomenological CM	Required Effect on Physical CM	Intention (Effect on Criticality)
$C_{\#1}$	Glare	Objects which are close to the intensive light sources on the 2D projection are perceived with reduced accuracy or not at all	Pixels in the 2D camera image are over-modulated due to overly intense lighting; seems very unpractical due to high dependence on actual camera	Scenarios become more critical on average due to non-perception/mis-perception (increased false-negative rate/classification uncertainty) of occluded objects
$C_{\#2}$	Occlu-sion	Objects in line-of-sight block the perception of occluded objects according to transparency and degree of the occlusion	Occlusion is mapped to the 2D camera image according to rays of light being blocked by the occluding objects and its transparency and shape	Scenarios become more critical on average due to non-perception (increased false-negative rate) of occluded objects
$C_{\#3}$	Mirror-ing	Objects in near reflecting surfaces appear twice in the environment according to light source angle, distance and position relative to reflecting surface	Projection of the object is mapped to the 2D camera image according to the rays of light emitted by the reflecting surface	Simulation of ghost-objects (increased false-positive rate); Mis-perception of objects due to strong reflecting surfaces
$C_{\#4}$	Air Parti-cles	Reduce the classification and edge detection sensitivity	Impact of the particles on the single light rays is modeled (absorption, scattering), or the 2D camera image can be distorted using 2D effects	Air particles can disturb camera image such that edge and object detection becomes complicated
$C_{\#5}$	Lim-ited Global Light Source	Reduce the classification and edge detection sensitivity	Reduce the saturation of the 2D camera image a posteriori, or reduce the actual number of rays hitting the camera sensor, i.e. implement the exposure in the CM	Low illumination leads to underexposure of images, resulting in difficulties for edge and object detection

a summary of the developments in openPASS within the context of the VVM project, and an outlook regarding further necessary developments in Section 4.3. Note that the requirements on and developments in openPASS are due to the

VVM project's GUA5, which is a working group within VVM to develop and adjust openPASS according to the needs of the criticality analysis.

4.1 Introduction to openPASS

Within the VVM project multiple simulation tools are applied to tackle the simulation tasks coming from the criticality analysis. One of those tools is the *open Platform for Assessment of Safety Systems* (openPASS) [16].

The rise of advanced driver assistance systems (ADASs) and automated driving systems comes along with the need to assess these systems and their effects in computer simulations. This particularly refers to safety effects in traffic. There are various methods and tools for prospective evaluation of safety systems with respect to traffic safety. Implementing the method by creating and maintaining the openPASS platform will support reliability and transparency of results obtained by simulation. The growing number, complexity, and variety of automated driving functionality make simulation an essential part in research, development, testing, public rating, and potentially even homologation. It is thus, directly or indirectly, required by all stakeholders in vehicle safety, such as manufacturers, suppliers, insurance companies, legislators, consumer advocates, academia, and others. The aim is to provide a software platform that enables the simulation of traffic situations to predict real-world effectiveness of ADASs and ADSs.

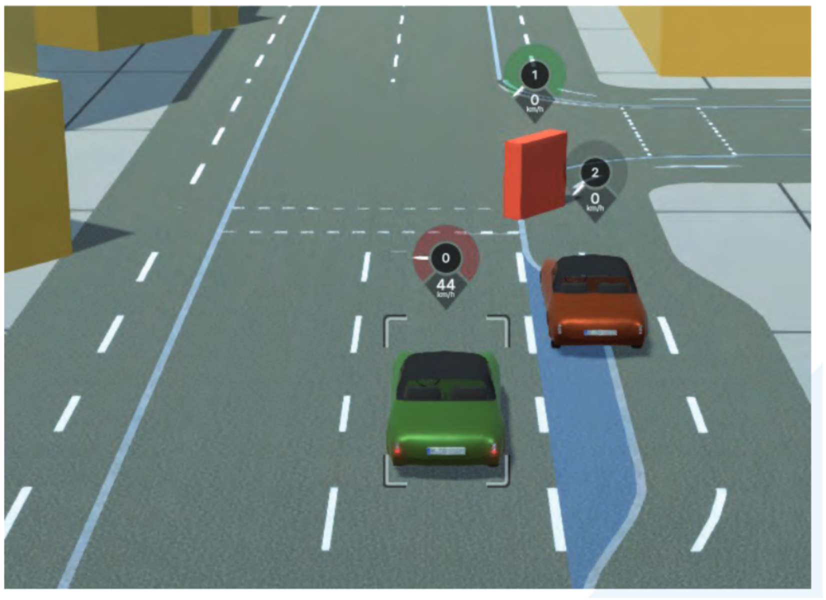


Fig. 4: Screenshot of simultating FUC¸2-3, cf. Figure 12, using openPASS. The green car is equipped with a simple HAD function. The bicyclist is occluded by the parking car [17].

In short, openPASS specifies an open source platform for a prospective safety assessment of those technologies. Originally, the term 'openPASS' formed a backronym for 'Open Platform for the Assessment of Safety Systems', but it was expanded beyond the limitations of safety systems towards any kind of driving automation functionality. Thus, one of the main target objectives is to become a broadly accepted platform for the assessment of the effectiveness of ADASs and ADSs. For this, as a first step, the openPASS working group seeks to develop a trustworthy, reliable and transparent platform.

OpenPASS is a framework for simulating the interactions between traffic participants for the evaluation and calibration of active safety systems. While each simulation run is based on a specific configuration, there is the possibility of parameter variation. Therefore, multiple slightly different simulation runs based on the same configuration are possible.

Historically, the software suite of openPASS started as a set of stand-alone applications, which can be installed and configured individually. Over time the graphical user interface evolved to be a single entry point.

4.2 Derivation of Requirements on openPASS for Criticality Analysis

Here, we follow through with the derivation of requirements as indicated in Section 3. We start with general, scenario-independent requirements openPASS in Section 4.2.1, provided as a table. Then, we derive requirements from the so-called *functional use cases* (FUCs) which are the 12 characteristic functional scenarios that have been used for the continuous evaluation of methods in the VVM project [18]. These 12 functional scenarios are comprised of three different urban intersections (FUC1, FUC2, FUC3) with four variations of increasing complexity each (FUCx.0,...,FUCx.3). The corresponding derived requirements are presented in Section 4.2.2, Section 4.2.3, and Section 4.2.4 respectively.

It is important to mention that we do not repeat requirements that reappear for a subsequent scenario, i.e. if a requirement has already been listed for FUC1.1, but it is relevant to FUC1.2 as well, we do not list it again. As the number of scenarios used for requirement elicitation grows, we do expect the number of novel requirements to decrease. Such an *saturation effect* would indicate that the scenario-based requirement eliciation for criticality analysis is indeed a convergent process.

The tables which are used to organize the elicited requirements in the following have four columns, featuring (i) a requirement ID, (ii) the requirement itself, (iii) the type of requirement, and (iv) how the requirement was covered. Regarding (iii), note that, we refine the simple classification between 'simulation system' and 'simulation models' a litte further by subdividing the simulation system into 'simulation core', '(external) components', and '3D models' (i.e. visualization) and 'models' has 'model interaction' as a subcategory.

4.2.1 General Requirements for Criticality Analysis The elicitation of general, i.e. scenario-independent, requirements on openPASS is documented in Table 2.

4.2.2 Scenario-based Requirements for Functional Use Case 1 The elicitation of requirements on openPASS derived from FUC1-0, cf. Figure 5, is documented in Table 2. Likewise, the requirements derived from FUC1.1, cf. Figure 6, FUC1.2, cf. Figure 7, and FUC1.3 cf. Figure 8, are documented in the Tables 4, 5, and 6 respectively.

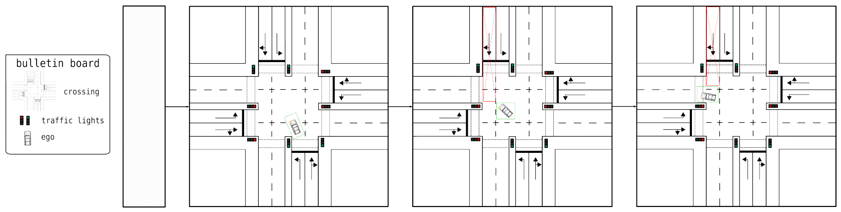


Fig. 5: Functional Use Case 1.0: Left turn on an X-crossing with traffic lights.

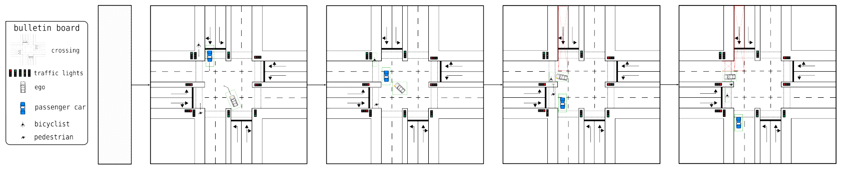


Fig. 6: Functional Use Case 1.1: Traffic with Right of Way.

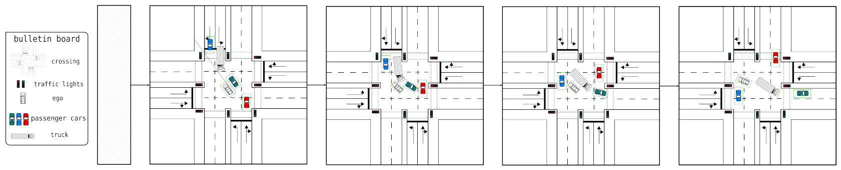


Fig. 7: Functional Use Case 1.2: Traffic with Right of Way and Occlusion.

4.2.3 Scenario-based Requirements for Functional Use Case 2 Here, we list the requirements derived from FUC2.0, cf. Figure 9, and its variations

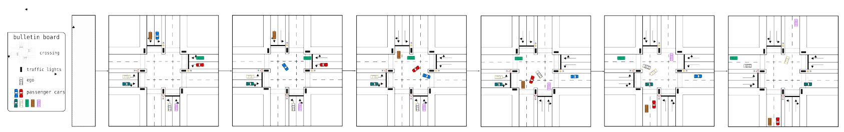


Fig. 8: Functional Use Case 1.3: Dysfunctional Traffic Lights at Rush Hour.

FUC2.1, FUC2.2, FUC2.3, visualized by Figures 10, 11, and 12 respectively. The requirements derived from FUC2.0 are documented in Table 7 and the requirements derived from FUC2.1, FUC2.2, and FUC2.3 appear in Table 8.¸

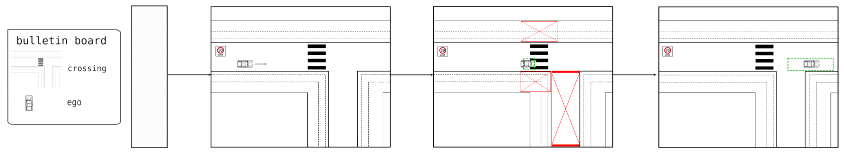


Fig. 9: Functional Use Case 2.0: Straight Passing of a T-Crossing with Pedestrian Crossing.

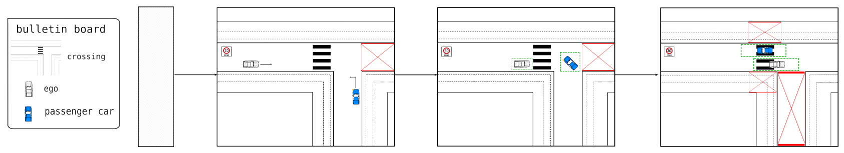


Fig. 10: Functional Use Case 2.1: Traffic with Right of Way.

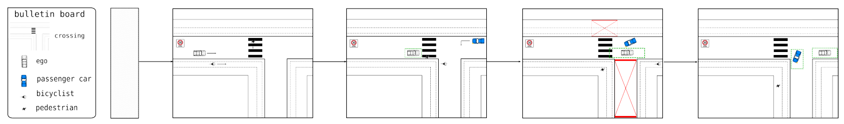


Fig. 11: Functional Use Case 2.2: Pedestrian, Bicyclist, and Oncoming Traffic.

4.2.4 Scenario-based Requirements for Functional Use Case 3 Due to a saturation effect already reducing the number of additional requirements, we display the requirements derived from FUC3.0, FUC3.1, FUC3.2, and FUC3.3 in a single Table 9. The visualizations of these functional use cases are given by the Figures 13, 14, 15, and 16 respectively.

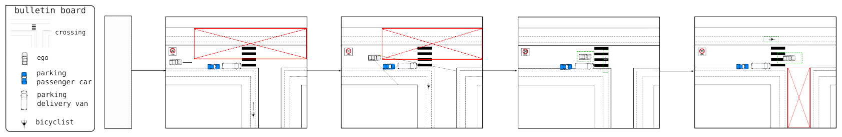


Fig. 12: Functional Use Case 2.3: Occlusion of Bicyclist through Parking Cars.

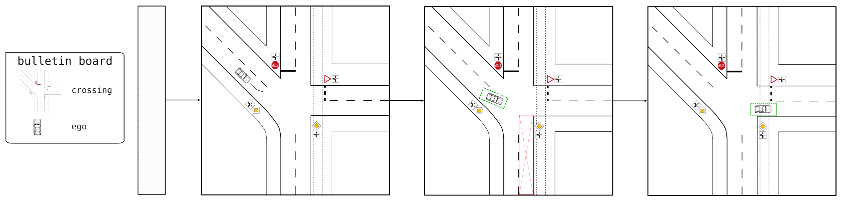


Fig. 13: Functional Use Case 3.0: Left Turn on a Bending Right of Way Crossing.

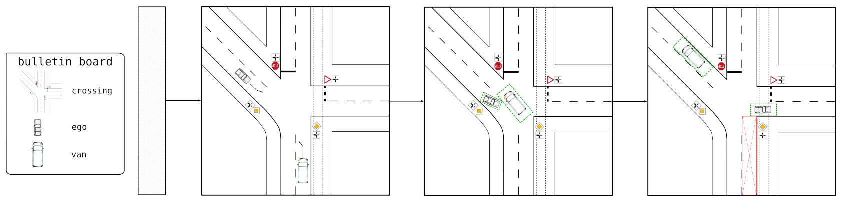


Fig. 14: Functional Use Case 3.1: Traffic with Right of Way.

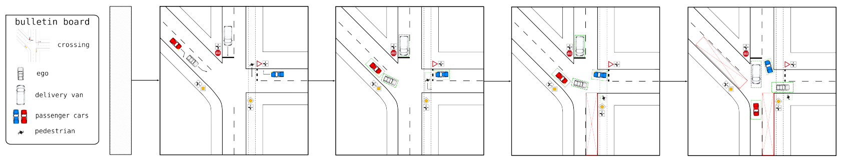


Fig. 15: Functional Use Case 3.2: Traffic and Pedestrian with Right of Way.

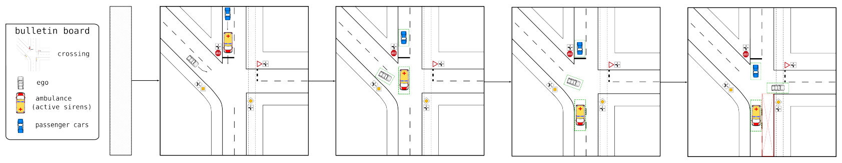


Fig. 16: Functional Use Case 3.3: Ambulance with Following Car.

4.3 Developments in openPASS for Criticality Analysis

Having derived general and scenario-based requirements on openPASS in Section 4.2, we now summarize which of them were realized within the VVM project, particularly through GUA5. The respective Tables 3, 4, 5, 6, 7, 8, and 9 already contain that information in their last column called *covered by*.

There are essentially three categories of contributions: developments directly from the openPASS working group, developments within the SET Level project and developments within the VVM project.

Regarding the general requirements of Section 4.2.1, the VVM GUA5 mainly contributed to implementing the requirements $G_{\#1}$ and $G_{\#5.1}$ (weather and environmental conditions). Notably, $G_{\#2}$ (evaluation of criticality metrics) was realized within VVM by integrating, as an external component, the tool *CriSys* which was developed by ZF in the context of the VVM criticality analysis [11].

On the infrastructure level, FUC1 demanded traffic lights. These were available in openDRIVE, but could previously not be handled by openPASS. Implemented by VVM GUA5, traffic lights are now available in openPASS, corresponding to requirement $FUC1.0_{\#1}$. Another development from VVM GUA5 regarding the infrastructure is the availability of pedestrian crossings in openPASS, as requested by $FUC2.0_{\#1}$.

As FUC1.1, FUC2.2, and FUC2.3 contain a bicyclist and no bicycle model was available to us, an openPASS-specific bicycle model was developed within VVM GUA5. The developed bicycle model even implements stochastic behavior in FU2.3: Either the model stops the bicycle at the pedestrian crossing, the cyclist dismounts and pushes the bicycle across the street or the model rides over the pedestrian crossing without dismounting, automatically resulting in a higher speed when crossing the intersection. The development of such a bicycle model directly addresses the requirements $FUC1.1_{\#4}$.

Due to incompatibility with the VVM functional use cases, we could not use the ADS model from SET Level, but had to develop our own driving automation. Based on the *Closed-Loop Driver Model* and the *Object-based Camera Object Model* from the SET Level project[7], we implemented driving functionality that could handle the requirements imposed by the FUCs, e.g. ego perceiving bicyclist ($FUC2.2_{\#1}$) or implementing the CP 'Occlusion' ($FUC2.3_{\#3}$). Moreover, the developed ADS model fulfills the requirements $FUC1.0_{\#3}$, $FUC2.0_{\#2}$, $FUC2.0_{\#3}$, and $FUC2.0_{\#5}$.

5 Conclusion

In this work, we examined in-depth the relation between a methodical criticality analysis and virtual simulation of automated driving systems. In particular, we described various simulation use cases coming from the criticality analysis. These simulation use cases impose requirements on the simulation system as well as on simulation models. Furthermore, we provided a process how these requirements can be derived systematically, using a scenario-based approach. This process was subsequently evaluated in case of the open source simulation platform *openPASS*.

Future work could analyze which simulators are best suited for criticality analysis by checking the amount of satisfied requirements. Clearly one could also invest more time and effort in writing down scenario-based requirements

[7] https://gitlab.setlevel.de/open

as to facilitate completeness of a *catalog of requirements for criticality analysis* exploiting saturation effects. Finally, the elicitation of requirements on sensor models could be expanded to the entire CP-catalog and to other sensor technologies.

References

1. Eckard Böde, Matthias Büker, Ulrich Eberle, Martin Fränzle, Sebastian Gerwinn, and Birte Kramer. Efficient Splitting of Test and Simulation Cases for the Verification of Highly Automated Driving Functions. In *Computer Safety, Reliability, and Security: 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19-21, 2018, Proceedings 37*, pages 139–153. Springer, 2018.
2. Christian Neurohr, Lukas Westhofen, Martin Butz, Martin Herbert Bollmann, Ulrich Eberle, and Roland Galbas. Criticality Analysis for the Verification and Validation of Automated Vehicles. *IEEE Access*, 9:18016–18041, 2021.
3. Christian Neurohr, Lukas Westhofen, Martin Butz, Roman Gansch, Martin Bollmann, Michael Knoop, Lina Putze, Tjark Koopmann, Armin Rasch, Bogdan Cojocaru, and Johannes Daube. Advances on the Criticality Analysis for Automated Driving Systems, 2023.
4. Lukas Westhofen, Christian Neurohr, Tjark Koopmann, Martin Butz, Barbara Schütt, Fabian Utesch, Birte Neurohr, Christian Gutenkunst, and Eckard Böde. Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art. *Archives of Computational Methods in Engineering*, 30(1):1–35, 2023.
5. Stefan Babisch, Christian Neurohr, Lukas Westhofen, Stefan Schoenawa, and Henrik Liers. Leveraging the GIDAS Database for the Criticality Analysis of Automated Driving Systems. *Journal of Advanced Transportation*, 2023, May 2023. Publisher: Hindawi.
6. Lukas Westhofen, Christian Neurohr, Martin Butz, Maike Scholtes, and Michael Schuldes. Using Ontologies for the Formalization and Recognition of Criticality for Automated Driving. *IEEE Open Journal of Intelligent Transportation Systems*, 2022.
7. Maike Scholtes, Lukas Westhofen, Lara Ruth Turner, Katrin Lotto, Michael Schuldes, Hendrik Weber, Nicolas Wagener, Christian Neurohr, Martin Herbert Bollmann, Franziska Körtke, Johannes Hiller, Michael Hoss, Julian Bock, and Lutz Eckstein. 6-layer model for a structured description and categorization of urban traffic and environment. *IEEE Access*, 9:59131–59147, 2021.
8. Jan Steffen Becker, Tjark Koopmann, Birte Neurohr, Christian Neurohr, Lukas Westhofen, Boris Wirtz, Eckard Böde, and Werner Damm. *Simulation of Abstract Scenarios: Towards Automated Tooling in Criticality Analysis*, pages 42–51. Autonomes Fahren. Ein Treiber zukünftiger Mobilität, Zenodo, 2022.
9. Barbara Schütt, Markus Steimle, Birte Kramer, Danny Behnecke, and Eric Sax. A taxonomy for quality in simulation-based development and testing of automated driving systems. *IEEE Access*, 10:18631–18644, 2022.
10. Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.
11. Bogdan Cojocaru and Johannes Daube. Criticality Identification System (CriSys). VVM Project Final Event, 2023.

12. Tobias Merk, Andreas Linnemann, Martin Bollmann, and Mladjan Radic. Merlin-a potential based criticality measure to quantify danger and risk in trac scenarios, 11 2022.
13. Roland Galbas Michael Knoop, Thomas Kirschbaum. Evasion Threat Metrics (ETM) Criticality Metrics derived from Societal Goals. VVM Project Final Event, 2023.
14. Wael KM Alhajyaseen. The Integration of Conflict Probability and Severity for the Safety Assessment of Intersections. *Arabian Journal for Science and Engineering*, 40:421–430, 2015.
15. Tjark Koopmann, Christian Neurohr, Lina Putze, Lukas Westhofen, Roman Gansch, and Ahmad Adee. Grasping causality for the explanation of criticality for automated driving. *arXiv preprint arXiv:2210.15375*, 2022.
16. Jan Dobberstein, Joerg Bakker, Lei Wang, Timo Vogt, Michael Düring, Lukas Stark, Jason Gainey, Alexander Prahl, Ralph Mueller, and Gaël Blondelle. The eclipse working group openpass-an open source approach to safety impact assessment via simulation. In *Proc. 25th ESV Conference*, 2017.
17. Tuan Duong Quang and Christian Neurohr. Criticality Analysis with openPASS. VVM Project Final Event, 2023.
18. Lukas Westhofen and Christian Neurohr. Functional Use Cases-Characteristic Scenarios for the Evaluation of Urban Driving Automation. VVM Project Midterm Event, 2022.

Table 2: General requirements, i.e. scenario-independent, on openPASS for criticality analysis.

ID	Requirement	Type	Covered By
$G_{\#1}$	openPASS is able to implement weather conditions as specified using OpenSCENARIO v1.0	Simulation Core	VVM GUA5 & and SET Level
$G_{\#2}$	openPASS supports the evaluation of criticality metrics, either by live-evaluation during simulation run time, or evaluation after the simulation run using logged data	External Component	First option: TTC, THW & 'speeding' available in openPASS; Second option: Integration of CriSys in toolchain
$G_{\#3}$	openPASS supports the ingestion of parameter ranges and distributions as well as the deterministic & stochastic variation of such scenario parameters, e.g. specified by openSCENARIO v1.0 (timing, position, speed, triggers, ...) and specified by openDRIVE v1.6 (lane width, number of lanes, ...)	External Component	SET Level (dSpace-Script) only OpenSCENARIO; no variation of OpenDRIVE
$G_{\#4}$	(Optional) openPASS allows a rollback during a scenario run back to a certain point in the history of the run (enabling rare event simulation)	Simulation Core	not covered
$G_{\#5}$	Models (sensor, vehicle, driver) used in openPASS react realistically to environmental conditions (such as weather)	Model Interaction, Simulation Core	see sub-requirements #5.1 and #5.2
$G_{\#5.1}$	Environmental conditions from OpenSCENARIO are available in openPASS	Simulation Core	VVM GUA5
$G_{\#5.2}$	Models in openPASS make use of environmental conditions (such as weather)	Model Interaction	not covered
$G_{\#6}$	openPASS supports Visualization (at least 2D) of all simulated entities over time	External Component	Open Source Visualization of BMW
$G_{\#7}$	openPASS enables (synchronized?) parallelized execution of multiple simulation runs	Simulation Core	not covered
$G_{\#8}$	openPASS supports a headless mode (mode without GUI)	Simulation Core	available in openPASS
$G_{\#9}$	openPASS can batch-execute simulation runs	Simulation Core	available in openPASS
$G_{\#10}$	openPASS supports the deterministic and stochastic variation of model parameters	Models	available in openPASS

Table 3: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Case 1.0.

ID	**Requirement**	**Type**	**Covered By**
FUC 1.0#1	openPASS is able to interpret complex X-crossings including markings and traffic lights specified using openDRIVE v1.6	Simulation Core	Crossing and Markings available in openPASS; Traffic Lights: VVM GUA 5
FUC 1.0#2	openPASS is able to interpret a vehicles trajectory (or sequence of maneuvers; here: turn left if traffic light is green and no intersecting traffic) specified using openSCENARIO v1.0	Simulation Core	available in openPASS
FUC 1.0#3	Model of an AD system (ego) including perception, planning and decision making	Models	Driver and camera model: SET Level; ADS: VVM GUA5 based on SL models
FUC 1.0#4	Vehicle Model including actuators	Models	SET Level
FUC 1.0#5	Sensor Models (Radar, Lidar, Camera, ...)	Models	SET Level
FUC 1.0#6	3D-Model for passenger car, crossing, infrastructure	3D-Models	not covered
FUC 1.0#7	openPASS supports ”nowhere boxes” (areas in which certain entities are not allowed to spawn / occur)	Simulation Core	available in openPASS (by design)

Table 4: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Case 1.1.

ID	Requirement	Type	Covered By
FUC 1.1#1	Driver model(s) for passenger car following a trajectory (or sequence of maneuvers); Driver model supports pseudo-randomized behavior	Models	Driver Model from SET Level follows trajectories; No pseudo-randomized behavior
FUC 1.1#2	Interaction between ego and passenger car: ego perceiving passenger car, ego adhering to traffic rules by giving right of way	Model Interaction	Implemented in actor models
FUC 1.1#3	Pedestrian model(s) for pedestrian crossing half of the street (following a trajectory or sequence of maneuvers); Pedestrian model supports pseudo-randomized behavior	Models	Pedestrian Model from SET Level follows trajectories; No pseudo-randomized behavior
FUC 1.1#4	Bicyclist model(s) for bicyclist crossing the street (following a trajectory or sequence of maneuvers); Bicycle model supports pseudo-randomized behavior	Models	VVM GUA5 (openPASS-internal bicyclist model)
FUC 1.1#5	3D-Models for pedestrian, bicyclist, blue passenger car	3D Models	not covered

Table 5: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Case 1.2.

ID	Requirement	Type	Covered By
FUC 1.2#1	Driver model(s) for Truck following a trajectory (or sequence of maneuvers); Driver model supports pseudo-randomized behavior	Models	not covered
FUC 1.2#2	Interaction between ego, green passenger car and truck in the middle of the intersection	Model Interaction	not covered
FUC 1.2#3	Sensor models accurately reflect limited information about blue passenger car hidden by the truck, thus implementing the criticality phenomenon 'occlusion'	Model Interaction	SET Level
FUC 1.2#4	3D-Models for truck, green/red passenger car	3D Models	not covered
FUC 1.2#5	Vehicle model for Truck including actuators	Models	not covered

Table 6: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Case 1.3.

ID	**Requirement**	**Type**	**Covered By**
FUC 1.3#1	openPASS is able to interpret suspended traffic lights, traffic signs	Simulation Core	Traffic signs available in openPASS; Suspended traffic lights not available
FUC 1.3#2	AD system (ego) reacts to suspended traffic lights and aheres to traffic signs	Models	SET Level Driver Model reacts to traffic signs, but does recognize suspended traffic lights
FUC 1.3#3	Driver models react to suspended traffic lights and adhere to traffic signs; Driver model supports pseudo-randomized behavior	Models	SET Level driver model reacts to traffic signs, but does recognize suspended traffic lights
FUC 1.3#4	Interaction between ego and passenger cars: ego perceiving passenger cars, ego adhering to traffic rules	Model Interaction	Implemented in actor models
FUC 1.3#5	A variety of different 3D models for passenger cars is available	3D Models	not covered

Table 7: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Case 2.0.

ID	Requirement	Type	Covered By
FUC 2.0#1	openPASS is able to interpret T-crossings including bicycle paths and pedestrian crossing specified using openDRIVE v1.6	Simulation Core	Pedestrian crossing: VVM GUA 5; other: already available in openPASS
FUC 2.0#2	AD system (ego) perceives absence of pedestrians/bicyclists	Models	VVM GUA 5 (based on driver and camera model from SET Level)
FUC 2.0#3	AD system (ego) perceives absence of road users with right of way	Models	VVM GUA 5 (based on driver and camera model from SET Level)
FUC 2.0#4	3D-Model for T-crossing, pedestrian crossing, traffic sign	3D-Models	Open Source Visualization (BMW) supports these features
FUC 2.0#5	AD system (ego) perceives pedestrian crossing	Models	VVM GUA 5 (based on driver and camera model from SET Level)

Table 8: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Cases 2.1, 2.2, and 2.3 respectively.

ID	Requirement	Type	Covered By
FUC 2.1#1	Interaction between ego and passenger car: ego perceiving passenger car, ego adhering to traffic rules by giving right of way	Model Interaction	Driver and camera model from SET Level
FUC 2.2#1	Interaction between ego, pedestrian, bicyclist and passenger car: ego perceives pedestrian, bicyclist, passenger car; ego adhering to traffic rules by letting the pedestrian pass, but passing the intersection in front of the blue passenger car	Model Interaction	Driver and camera model from SET Level; VVM GUA5 regarding bicycle
FUC 2.3#1	AD-system (ego) perceives absence of oncoming traffic	Models	Driver and camera model from SET Level
FUC 2.3#2	openPASS is able to interpret parking vehicles specified using openSCENARIO v1.0	Simulation Core	avaialble in openPASS
FUC 2.3#3	Sensor models accurately reflect limited information about the bicyclist hidden behind the parking vehicles, thus implementing the criticality phenomenon 'occlusion'	Model Interaction	Camera model from SET Level
FUC 2.3#4	3D-Model for a (delivery) van	3D-Models	not covered

Table 9: Scenario-based requirements on openPASS for criticality analysis, derived from VVM Functional Use Cases 3.0, 3.1, 3.2, and 3.3.

ID	**Requirement**	**Type**	**Covered By**
FUC 3.0#1	openPASS is able to interpret an X-crossing with non-orthogonal arms (ideally, at arbitrary angles) and bicycle lane including markings and traffic signs specified using openDRIVE v1.6	Simulation Core	available in openPASS
FUC 3.1#1	Driver model(s) for a Van following a trajectory (or sequence of maneuvers); Driver model supports pseudo-randomized behavior	Models	Driver model from SET Level follows a trajectory; does not support pseudo-randomized behavior
FUC 3.1#2	Interaction between ego and van: ego perceiving van, ego adhering to traffic rules by giving right of way before leaving the major road	Model Interaction	Driver and camera model from SET Level; Priority rules too complicated for driver model here
FUC 3.2#1	AD system (ego) perceives all actors (red/blue passenger car, pedestrian, van) and interprets the situation according to traffic rules	Models	Driver and camera model from SET Level; Priority rules too complicated for driver model here
FUC 3.3#1	Driver model(s) for an amublance following a trajectory (or sequence of maneuvers) with/without active sirens; Driver model supports pseudo-randomized behavior	Models	SET Level driver model can be used; does not support pseudo-randomized behavior
FUC 3.3#2	Driver model(s) for a vehicle conditionally following another vehicle	Models	SET Level driver model implements vehicle following; adherence to traffic rules is a separate consideration
FUC 3.3#3	AD system (ego) perceives all actors (blue passenger car, ambulence) and interprets the situation according to traffic rules	Models	Driver and camera model from SET Level; Priority rules too complicated for driver model here
FUC 3.3#4	Interaction of ego, ambulence and blue passenger car: ego recognizes active sirens of the ambulence and gives right of way, ego leaves major read before the blue passenger car	Interaction of models	not covered