Framework for using real driving data in automotive feature development and validation

Jacob Langner, Johannes Bach, Stefan Otten, Eric Sax
FZI Research Center for Information Technology
Email: {langner, bach, otten, sax}@fzi.de

Carl Esselborn, Marc Holzäpfel, Michael Eckert
Dr. Ing. h.c. F. Porsche AG
Email: {carl.esselborn1, marc.holzaepfel, michael.eckert}@porsche.de

***Abstract*—The increasing complexity and interconnectivity of automotive features raises the significance of comprehensive Verification and Validation (V&V) activities. High-level automotive features use the information provided by complex environmental perception sensors and systems. Due to the rising number of these sensors and the usage of enhanced digital maps, System level V&V for high-level features has become a challenge, that is often tackled by a combination of real world tests and simulation approaches. In this contribution we present a method, that combines the realism of real world tests with the scalability of simulation approaches. In the presented framework a feature under development is executed in a Software-in-the-loop (SIL) environment with the help of recorded real world driving data. With the steadily growing pool of recorded test drives from test campaigns and country approvals, large scale simulations have been facilitated. This enables statistically significant assertions, continuous maturity tracking as well as geolocation-dependent evaluation of the feature under test. The framework makes these large scale simulations feasible during automotive feature development by utilizing parallelization concepts to achieve simulation speeds of thousands of kilometers within minutes and by reducing adaptation overhead for changes in the feature's software code to a minimum.**

Keywords*—*Automotive Systems Engineering; Verification and Validation; Large scale simulations; Continuous Maturity Tracking; Geolocation-Dependent Testing; MATLAB

I. Introduction

The current and upcoming trends electrification, automation and connectivity in the automotive industry [1] affect the Automotive Systems Engineering (ASE) process. Exemplary changes are the increasing size, complexity and interconnections of functional components [2], the integration of back-end and mobile services, the introduction of car-to-x communications and changes in the deployment process from linear-stage gated releases to circular update-over-the-air releases [3]. The increasing vehicle's sphere of perception also raises the system complexity. Precise, high definition digital maps [4], advanced sensor systems [5] and internet services [6] open up system boundaries, which further challenges the established V&V approaches. All development activities, starting from early concept work and extending through to V&V activities, are affected by these changes. Tools, methods and principles for ASE have to be adopted to overcome these new challenges.

In order to reduce system complexity, a common solution in systems engineering is the decomposition of the system, in this case the feature under development, into smaller parts [7]. For automotive features the system, component and unit level are distinguished. Units are independent software blocks, implementing functionality defined by their specifications. Components integrate several related units, which in turn can be composed to the system [8].

V&V tasks during development are broken down respectively to the system decomposition. For each of the three abstraction layers, different V&V activities and goals are specified [9]. During unit tests a single software unit is tested to ensure specification compliance and failure-free operation. After each unit has been examined, integration tests on component level verify the unit-integration as well as the interface's specification compliance. V&V on system level reviews the system from an external and holistic point of view. Tests aim at verifying the correct integration and interaction of all components and at validating the feature against the expected user experience.

While the breakdown of V&V activities into unit, component and system level helps to structure the process, many challenges for the different levels remain. On system level sufficient test coverage as well as cost and time factors are critical for the feasibility of the test concept. Especially during preliminary development, new and innovative features require peculiar testing to ensure specification compliance. Prototypes enable a preview of the feature in it's future environment. However, recent analyses [10] of the current V&V requirements for upcoming autonomous features have shown that the increased complexity and higher automation levels entail a tremendous growth in V&V efforts. The necessary, sufficient test coverage can not be achieved by real world tests or any other real-time testing methods alone. Faster and better scaling virtual tests have to extend real world test coverage. Current simulation approaches still require a lot of manual work, e.g. for providing detailed models and parametrization [11]. They are often not suited for complex and connected features, e.g. features using back-end or car-to-car communication, features heavily relying on realistic environmental sensor data or features utilizing geolocation-dependent digital map information.

In this paper, we propose a framework that uses parallelization and distributed computing concepts in order to solve the above mentioned challenges for scalable, geolocation-dependent, realistic simulations. We utilize the continuously growing data pool of recorded prototype test drives and country approval campaigns, which Original Equipment Manufacturer (OEM) currently gather. Besides aggregated feature evaluation over the complete data pool, the framework also supports drill-down functionality for in-depth debugging of faulty situations utilizing the Reactive-Replay approach. In Section II we outline the state of the art in V&V activities during the

ASE process and refer to previous work regarding this topic. The concept of our framework is explained in Section III, whereas details of the architecture are shown in Section IV. Our prototypical implementation of the framework is evaluated in Section V. A conclusion and an outlook on future work are given in Section VI.

II. State of the Art in ASE

The V-Model [12] [13] is an established process model in the automotive domain, used to structure the development activities. The Automotive SPICE[1] reference model adapts the V-Model to ASE domain with it's specific demands and constraints [14]. It starts on the left-hand side with requirements elicitation and requirements analysis on system level. Needs and requirements from all stakeholders are collected and subsequently converted into a set of system requirements, which serve as a baseline for future design decisions and test evaluations. In the system architectural design phase a system architecture is designed and previously formulated requirements are mapped onto functional system elements. During the software requirements analysis and software design phases functional elements are mapped onto software components, for which software requirements and the software architecture are specified accordingly, resulting in several software units per software component. These software units are then implemented in the software construction phase. Software unit verification ensures design and requirements compliance of each unit. Integration and qualification on component, subsystem and system level follow subsequently during the software integration and qualification as well as the system integration and qualification phases. These tests ensure specification compliance and architecture consistency on the different abstraction levels.

With each integration level the V&V activities become increasingly realistic and resource intensive. Unit and component tests can be done virtually but provide limited validity in regard to overall system behavior, whereas subsystem and system tests usually comprise extensive Hardware-in-the-loop (HIL) and prototype tests, verifying and validating the functional behavior of the system in the real world.

As postulated in ISO29119-1 [15] dynamic testing is essential for validating the correct functional behavior. Boehm described "a software program as a mapping from a space of inputs into a space of outputs" [16]. Thereby, the functional quality of the system can be ensured by evaluating the system output while completely covering the input parameter space, treating the software itself as a black-box. But, the number and value ranges of input parameters alone span a wide parameter space. Limiting the value ranges to plausible values can reduce the parameter space significantly. Further reduction can be achieved by defining equivalence classes for each parameter's value range and testing the corner cases. However, due to the complexity and non-deterministic system behavior of Advanced Driver Assistant Systems (ADAS) features with internal states and time-dependencies, signal curves over time have to be considered. This requires realistic signal curves for all input parameters, which cover all significant parameter combinations.

[1] Automotive Software Process Improvement and Capability Determination

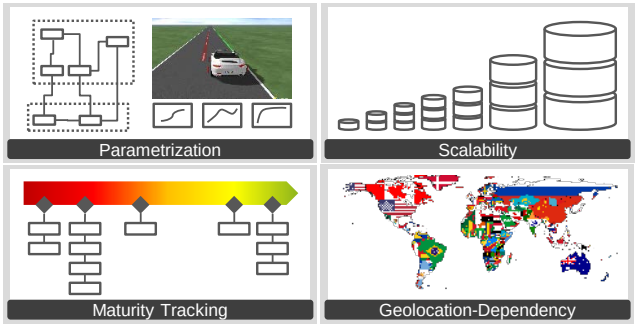


Fig. 1: Remaining challenges for feature V&V with our data-driven approach

Thus, one approach for V&V is to test the test item in the real world. In early development stages rapid prototyping can give proof of concept and help with design decisions. During the following development stages increasingly mature hardware prototypes are deployed. For system level V&V large HIL frameworks or complete prototype vehicles are used. In so called test campaigns these heavily equipped prototype vehicles are driven by test drivers and function developers over thousands of kilometers. Usually data loggers record FlexRay and CAN signals for debugging purposes. While these tests are vital and valuable to the feature maturity, they are also resource intensive and time consuming and therefore limitedly available. Furthermore, the test drives are not reproducible and the confined number of available prototype vehicles limits the scalability of this approach for repeated testing.

These real world tests can be complemented by virtual simulations to tackle the above mentioned challenges. The term X-in-the-loop (XiL) [17] [18] refers to a group of test methods, where a test item, e.g. the system, subsystem or component, is tested in a reproducible and controllable test environment. This is achieved by modeling system externals like the driver and the environment as well as other parts of the vehicle, the test item depends on or interacts with. Depending on the choice of the test item the XiL harness provides a reproducible and controllable test environment for every development phase. The key benefit is, that the test harness is provided once and then reused in every phase, enabling a consistent simulation environment. However, constructing a valid test harness with detailed models for all system externals remains a challenge. E.g. modeling the non-deterministic driver behavior or supplying an accurate street topology are resource intensive tasks and proving their validity is difficult. Geolocation-dependent features require environment models for different countries and sensors with increasingly extensive environmental perception pose immense modeling efforts.

Referring to Boehm, the complete space of inputs has to be covered to achieve sufficient test coverage during system level V&V. Since time and resources for testing are limited, a generic approach to cover the complete input space for complex ADAS features is not suitable. A common solution to this problem is the definition of test scenarios. Virtual test environments allow the detailed modeling of test scenes. However, reaching sufficient test coverage by specifying scenarios manually is still very resource intensive. Approaches to reduce scenario definition overhead by standardizing test

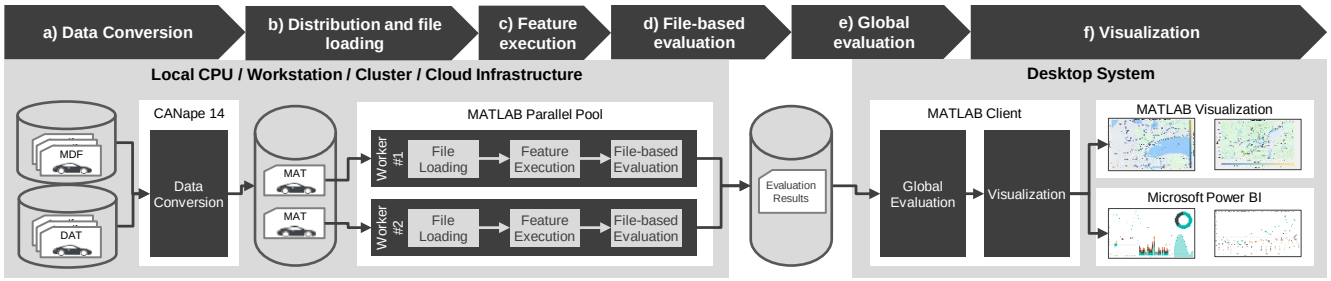


Fig. 2: The framework's concept and architecture

specifications and scenario definitions[2] have been made. Still, the lack of realism in virtual test scenarios is often criticized.

In other domains, video based acquisition is a common approach to collect test data. This automatic gathering method provides a simple way to collect large amounts of test data. It can be adapted to the automotive domain by recording the FlexRay and CAN-bus messages during test campaigns and prototype drives.

In previous work we suggested the use of this recorded real world driving data throughout the whole development process [19]. Bach et al. [20] have introduced the Reactive-Replay approach, that uses recorded real world driving data to virtually execute a closed-loop control feature on system level. The data is recorded during real world system level tests. The driven routes can then virtually and repeatedly be replayed and reviewed. Furthermore, in [21] we extended this approach by selecting carefully chosen test sets from the recorded data to reduce redundancy and therefore simulation time without loosing test coverage.

However, some challenges remain as shown in Fig. 1. The approach lacks scalability, since it is primarily tailored for manual, in-depth debugging of a few selected test drives. During the transition from software construction to software integration and qualification the need for statistically significant evaluations arises. Large scale simulations over thousands of kilometers have to be aggregated to track continuous feature maturity and qualitative feature progression on a daily, weekly or monthly basis. Especially before upcoming release approvals and test campaigns, exhaustive test coverage is valuable, e.g. simulating the feature with 10.000 km of country-specific test data and fixing all virtually occurring bugs and problems before traveling there elevates the benefits drawn from the test campaign. The risk of invalid or inoperable features during the test campaign is minimized and the focus can e.g. be set on the feature's perception-based user experience. The parametrization of the feature is another unsolved challenge. It is often used for last minute tweaks of the feature's performance before upcoming release approvals. These adjustments, even for seemingly small parameter changes, can have unforeseen effects, which might only occur in a specific test scenario. Therefore, parametrization has to be accompanied by exhaustive testing over as many scenarios as feasible.

III. Concept for the Framework

Our goal was to create a framework, which enables large-scale virtual simulations of open-loop as well as closed-loop control features based on real world driving data. For a given feature an easy integration method for it's proprietary software code shall be given to enable it's simulation with the real world driving data in order to compute assessment metrics for large scale, statistically significant feature evaluation and maturity tracking. For our prototypical implementation of the proposed framework, we adapted the Reactive-Replay approach to an open-loop feature, that predicts future vehicle and driver states. Recorded real world driving data is used to execute the feature and to derive a ground truth, the feature's predictions are validated against.

We identified the following core concepts as critical for the achievement of these goals:

a) Data Accessibility: The framework shall be able to handle automotive full-log files from different vehicles, car types and feature versions. Missing signals shall be substituted, if possible, to secure the utilization of the full data basis for any target or source vehicle. If necessary, matching vehicle models shall be loaded for the feature execution based on the source vehicle.

b) Parallelization: The simulation needs to be parallelizable to enable scalable feature execution over the continuously growing data basis. Parallelization is the core concept for this approach, since analyses need to be fast but also need to cover a wide parameter range with as much test data as possible.

c) Scalability: The framework needs to scale beyond the computational power of a desktop computer. Parallel execution ensures multi-core CPU usage. With cluster and cloud computing concepts, further scalability shall be secured. The framework must be able to keep up with the steadily growing data pool in order to utilize all available data to maximize validity and statistic significance of evaluations and continuous maturity tracking.

d) Modularization: The framework shall support easy extensibility and exchangeability of the evaluation metrics, visualization options and especially the feature under test to allow maturity tracking and functional comparison of different release or working versions. Therefore, a modular design shall be pursued. Clear interfaces and data structures shall further enhance the modularization. The feature under test shall be directly linked from the production source code. Signals from the real world driving data shall be passed to and from the feature via the bus interfaces.

e) Data Management: Executed analyses of different feature versions shall be comparable without re-execution of the feature simulation or evaluation. Aggregation and global

[2]e.g.: OpenScenario, http://www.openscenario.org/

metrics shall give a summarized overview over multiple test drives. Problems, errors and faulty situations shall be highlighted and in-depth breakdowns shall be provided. Persistent and scalable storage systems need to be supported.

f) Visualization: A flexible set of visualization options shall be provided. Selecting and adjusting evaluation contents shall require minimal effort. For the included metrics an easily understandable visualization must be provided. The implementation of new visualizations and the integration of external visualization tools must be possible.

We selected MATLAB as our main tool, because it enables easy parallelization, dynamic linking of C-code and has many visualization capabilities. Furthermore, MATLAB is an established tool in the automotive industry.

Parallelization is achieved with the help of MATLAB's Parallel Computing Toolbox. Each physical CPU core can host one MATLAB worker. The prototype framework's target is a 16 core workstation. Further performance increases can be achieved with MATLAB's Distributed Computing Server, which extends the parallel pool of workers to clusters and cloud computing services. For now, we achieved satisfactory results on the workstation.

Converter scripts for common data logger formats ensure data accessibility in MATLAB. Due to comparatively long conversion times, the conversion is done in batches after each test campaign. The converted files are stored persistently and can then be loaded by MATLAB's parallel workers, which simulate the dynamically linked feature with the measured input signals from the real world driving data. Existing vehicle and environment models, specified in proprietary C-code, are reused in the framework. The necessary digital maps are rebuild with the information from the real world driving data. The feature's output interface is returned to MATLAB, where assessment metrics evaluate the feature execution. Each recorded test drive is evaluated locally on the MATLAB worker, which also computed the simulation, saving further computation time. Only the global evaluation is done on the client computer after gathering the results. For visualization purposes we used maps with metric highlighting and other MATLAB plots as well as third-party visualization tools for more interactive dashboards.

IV. The framework's architecture

Fig. 2 depicts our framework's architecture. We ignored hardware aspects and solely focused on creating a SIL test environment, which of course is scheduled correctly, but without the real-time constraints. This allows functional evaluation of the feature under test on the developers desktop computer instead of the target hardware, but simultaneously limits it's utility for integration testing. Our parallelization concept regards each test drive as an atomic unit. The parallelization of a single test drive is not provided, since slicing up time-dependent data is not a trivial task and the massive overhead for correct slicing bears no proportion to the relative small computation times for each single test drive, which supersede the parallelization of single test drives in the first place. We focused on an easy to use, modular and extendable framework, which allows for fast functional evaluation of an integrated C-code feature with recorded real world driving data.

Fig. 2 shows the 6 steps, the framework is comprised of:

a) Data Conversion: The real world driving data is recorded in either MDF[3] or DAT[4] format, which both can not be accessed by MATLAB directly and therefore have to be converted to MAT files first. Since MATLAB's built-in converter is only able to convert 8-bit values, which are not feasible for FlexRay or CAN signals due to their bandwidth and size limitations, the CANAPE 14[5] converter library was used. A MATLAB script batch-processes any number of MDF files. For DAT files there is no direct conversion to MAT files yet and therefore they are converted to MDF and then to MAT. Signal selection, data format options and further settings can be specified by INI files and label-lists.

b) Distribution and file loading: As long as a local or remote computer is used, the test data can be accessed directly via the local or remote file system. When computing in cluster or cloud environments, distributed file systems should be put in place to ensure execution of tasks on cluster nodes with existing local files. HDFS [22] and Amazon's EMRFS [23] offer distributed file system services. Combined with a scheduler like Hadoop's MapReduce or Tez [22], Amazon's EMR [23] or MATLAB's MJS [24] efficient parallel computations can be assured. For flexibility reasons the complete automotive full-logs are converted in the conversion step. The execution of a single feature however only requires a small subset of signals. Each parallel MATLAB worker extracts this subset from it's current test file. Some signals may have to be interpolated or regrouped into their original message format in order to meet the feature's input interface requirements, e.g. the feature's cycle time.

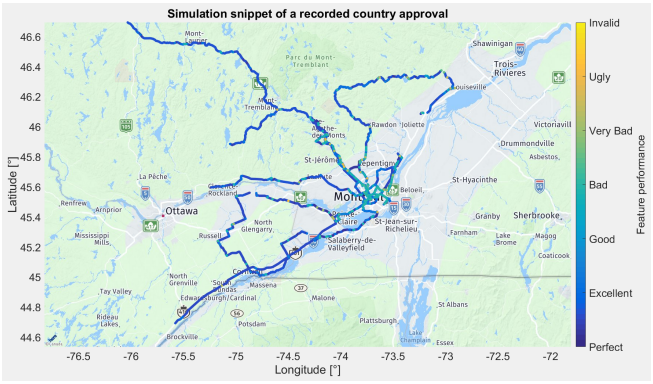
c) Feature execution: During feature execution the feature's software functions are executed. Since the feature is usually implemented in C-code, integration into MATLAB has to be provided. For each feature a DLL[6] file has to be written, where signals are passed into and out of the feature interfaces. The feature itself is considered a black box. Therefore, the DLL file has to be changed only, when the interfaces are changed. An early commitment to interface definitions reduces manual changes in the DLL file to a minimum. The DLL file also has an internal memory structure mimicking the ECU's internal storage, where loaded vehicle and environmental models are stored as well. The DLL file is dynamically linked to MATLAB with the help of a MEX-function. This wrapper function enables the forwarding of data from MATLAB to the C-code and back. For the feature execution each MATLAB worker dynamically links to a local copy of the feature DLL. After initializing the internal memory structures of the feature, the feature's input interface is served with the input signals from the recorded data and the included functionality is executed. Input signals from the recorded test file are sampled in the original feature execution cycle time until the complete file is processed. Asynchronous messages, like digital map data, are loaded when they occur in the

[3]Measurement Data Format, Vector Informatik GmbH, https://vector.com/vi_mdf_en.html

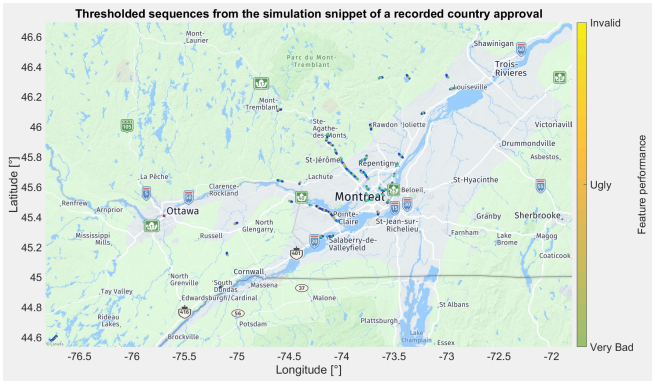
[4]ADTF's logging format, Elektrobit, https://www.elektrobit.com/products/eb-assist/adtf/

[5]CANAPE, Vector Informatik GmbH, https://vector.com/vi_canape_en.html

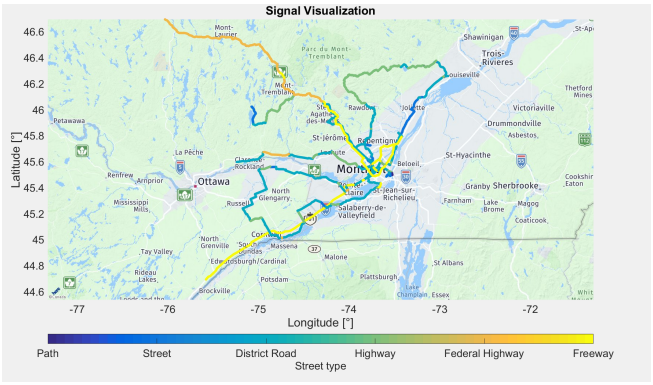
[6]What is a DLL?, Microsoft Corporation, https://support.microsoft.com/en-us/help/815065/what-is-a-dll



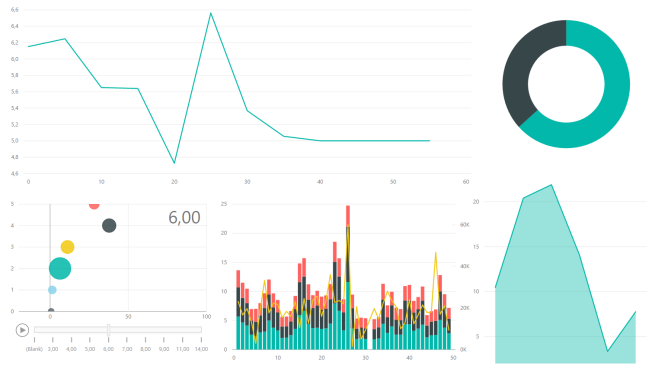
(a) Feature evaluation based on a calculated metric



(b) Debug facilitation with highlighted sequences of bad feature performance



(c) Measured signal visualization



(d) Global performance overview via interactive dashboards

Fig. 3: Exemplary visualization of a feature evaluation simulated with data from a country approval

recorded test file and are handled in a second function binding, which processes these messages accordingly. After each cycle step, the output signals from the feature's output interface are extracted and passed back to MATLAB, where they are stored for evaluation. Furthermore, the dynamic linking allows the readout of internal memory states, which may be used for evaluation or debugging purposes.

d) File-based evaluation: In the feature evaluation block assessment metrics can be included, which may use input, output and other measurement signals for evaluation. The modular nature of the framework enables a flexible, on-demand metric selection and adding of new metrics. The feature evaluation block is executed in parallel, which means, each MATLAB worker only has access to it's own feature execution results. Therefore, the evaluation is limited to the one test file in this step. After gathering all evaluation results, global analyses can be performed in the next step.

e) Global evaluation: The global evaluation is triggered, when all test drives have been simulated. MATLAB automatically gathers the results from each MATLAB worker, allowing analyses over the complete test set. Locally computed metrics are aggregated to achieve a summarized overview with drill-down functionality for selected cases. Additionally, an export script is provided for further usage of the evaluation results in third-party visualization tools. Any kind of storage system, e.g. a database or data warehouse, can be integrated to store execution results and analyses persistently. This allows downstream comparison of different versions and reuse of execution results for different analyses. It also opens up the data for data analytics or usage in other tools.

f) Visualization: The visualization part is built modularly to best suit the needs of the current user. We supplied map visualization scripts as well as aggregated and in-depth MATLAB plots for specific metrics or selected measurement signals. Fig. 3a shows a map snippet of a feature specific metric, computed for a simulation over a complete country approval campaign. The used metric is displayed via coloring on the map, parts, where the feature performed poorly, are highlighted on the map to ensure discoverability. Additionally, thresholds provide a method for sequence selection based on the current needs, which further facilitates the visualization, as shown in Fig. 3b. The automated slicing of test drives into sequences based on thresholds will be linked to a debug tool for the developer in the future. This enables a sequential debugging of situations, where the feature performed substandard. Interactive dashboards are used for a global overview of the analysis. With little modeling effort, a concise general evaluation of the feature's performance can be displayed. An abstracted, exemplary dashboard is shown in Fig. 3d.

Provided evaluation signals allow context-based filtering and classification of the calculated metrics, e.g. an evaluation metric segmented by street type or vehicle velocity. Fig. 3c depicts a simple signal visualization, which may be used to better understand the test data or to visually search for potential relationships between metric results and measurement signals.

V. Results and Evaluation

The framework was implemented for an open-loop feature. We showed, that the usage of real world driving data is a valid approach for virtual simulation. Thousands of test kilometers can simply be derived from recorded country approvals and prototype test drives. The integration of the feature's source code is a major advantage, since minimal adaptation efforts are required, when moving from implementation to testing. Thus, reducing the time for each iteration loop significantly. The framework's execution on a standard office laptop achieved a simulation speed of 160 kilometers per minute with two CPU cores for the integrated feature. This translates into 1.280 simulated test kilometers per minute on a 16 core workstation.

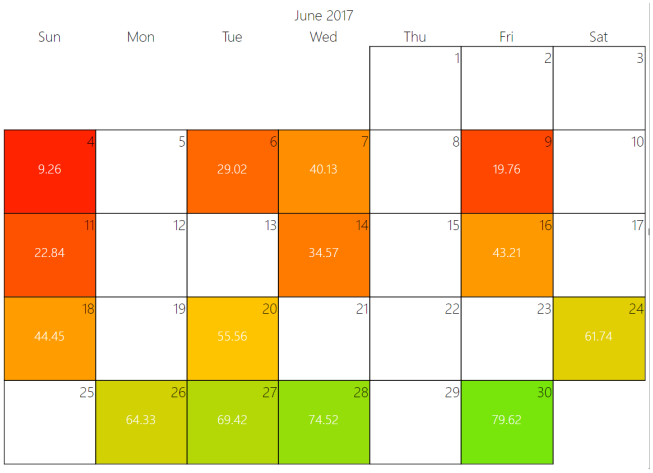
We identified the following use cases for the framework:

For one, **the feature can be evaluated and debugged during implementation phases**. As long as the interfaces remain the same, the new source code can be executed directly after successful compilation. The persistent storage of evaluation results enables maturity tracking and **comparison of different software releases and live versions**. Aggregated metrics over huge data sets deliver more reliable, statistically significant evaluations in addition to single drive in-depth debugging. Visually supported, direct comparison of the current version with the last release version can reveal wanted and unwanted changes in the feature's functional behavior.

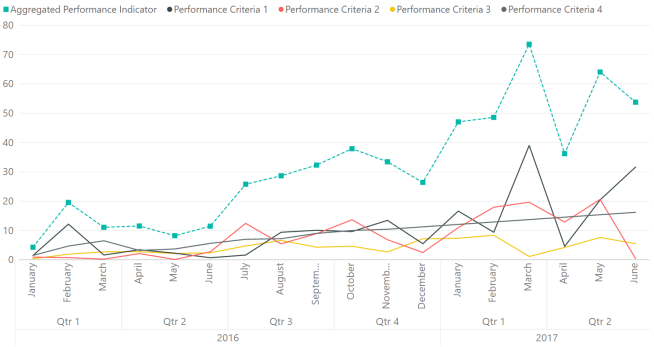
The framework can also be used to **optimize parametrization** of the feature. The feature can easily be executed with different parameter sets, allowing comparison of different parameter settings and manual parameter optimization with the help of feature execution results. The utilization of huge data sets prevents overfitting of parameters on specific test drives. An automated parameter optimization approach will be evaluated in the future.

Furthermore, the feature's **geolocation-dependent behavior in different countries and regions can be evaluated** on a desktop computer. Large scale simulations with complete test campaigns from different countries allow statistically significant prognoses for it's performance in the field. The influence of geolocation-dependent criteria like digital map quality, left- and right-hand traffic, highly diverse traffic signs as well as different weather conditions and road topologies can be evaluated with the framework. This enhances feature maturity and enables debugging before testing in the real world.

And last but not least, the framework supports large scale simulations over the complete data set or a relevant subset. Keeping this set the same and executing the latest release in a periodic schedule allows for overall **continuous maturity tracking**, as shown in Fig. 4, supporting the lead developer with test scheduling and release planning. In a calendric view (see Fig. 4a) a single performance indicator can be displayed.



(a) Calendric view for a single performance indicator



(b) Maturity tracking for multiple performance criteria

Fig. 4: Continuous feature maturity tracking

For a more detailed analysis, multiple performance criteria can be visualized over time in Fig. 4b. The dashed line shows the aggregated performance indicator and the other lines single performance criteria. These can be different metrics from the functional system-level testing or external criteria from e.g. static code analyses like code coverage or compliance to coding guidelines. Even user experience ratings from prototype test drives can be integrated into the aggregated performance indicator.

VI. Conclusion and Future Work

Our framework offers a parallel simulation environment, which integrates a feature's production source code and executes it with the input from recorded real world driving data. This opens up a large data pool of real world test drives and driving scenarios for the use in the feature's simulation and evaluation as well as parameter and signal analyses. Feature-specific assessment metrics are applied to evaluate the feature's performance globally across all test drives in the data pool as well as locally with the help of additional drill-down mechanics.

We have shown that the recorded real world driving data can be used to execute a feature in a SIL test environment on a common desktop computer. While established simulation approaches struggle with realism and test coverage and prototype drives with scalability and reproducibility, the reuse of real

world driving data in the simulation environment combines the benefits of both test methods while negating their downsides. Furthermore, in many cases a ground truth, e.g. for predicted vehicle states, can be derived from the recorded test data. This ground truth can then be used to enhance the evaluation of the feature's performance.

The framework can be used for multiple purposes, such as debugging and functional comparison of different software versions, parameter optimization, analysis of geolocation-dependent behavior as well as continuous maturity tracking. We implemented this framework prototypically for a geolocation-dependent open-loop feature and are currently working on the closed-loop adaptation of the framework.

With the basic framework in place, current and future work focuses on improved visualization capabilities and automated analyses. Due to the steadily growing data pool, data analytic and machine learning concepts and methods will be used to extract assessment parameters, correlations and new knowledge from the data, which then can be used for the feature evaluation. Assisting feature parametrization with an automated optimization approach is planned as well. Further scalability concepts will be implemented, e.g. migrating the framework to a cluster or cloud infrastructure, if faster simulations prove necessary.

REFERENCES

[1] K. Bengler, K. Dietmayer, B. Färber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 4, pp. 6–22, 2014.

[2] J. Bach, S. Otten, and E. Sax, "A taxonomy and systematic approach for automotive system architectures - from functional chains to functional networks," in *3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, 2017.

[3] S. Vöst and S. Wagner, "Towards continuous integration and continuous delivery in the automotive industry," *arXiv preprint arXiv:1612.04139*, 2016.

[4] J. T. Kessels and P. Van den Bosch, "Electronic horizon: Road information used by energy management strategies," *International Journal of Intelligent Information and Database Systems*, vol. 2, no. 2, pp. 187–203, 2008.

[5] M. Aeberhard, S. Rauch, M. Bahram, G. Tanzmeister, J. Thomas, Y. Pilat, F. Homm, W. Huber, and N. Kaempchen, "Experience, results and lessons learned from automated driving on germany's highways," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 42–57, 2015.

[6] C. Sun, S. J. Moura, X. Hu, J. K. Hedrick, and F. Sun, "Dynamic traffic feedback data enabled energy management in plug-in hybrid electric vehicles," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 1075–1086, 2015.

[7] B. Lightsey, "Systems engineering fundamentals," DTIC Document, Tech. Rep., 2001.

[8] ISO, "Road vehicles - functional safety - part4: Product development at system level," *ISO 26262-4:2011(E)*, pp. 1–36, Nov 2011.

[9] P. Bourque, R. E. Fairley *et al.*, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[10] J. Mazzega, F. Köster, K. Lemmer, and T. Form, "Testing of highly automated driving functions," *ATZ worldwide*, vol. 118, no. 10, pp. 44–48, 2016.

[11] T. Helmer, L. Wang, K. Kompass, and R. Kates, "Safety performance assessment of assisted and automated driving by virtual experiments: Stochastic microscopic traffic simulation as knowledge synthesis," in *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE, 2015, pp. 2019–2023.

[12] E. Sax, *Automatisiertes Testen Eingebetteter Systeme in der Automobilindustrie*. München: Hanser, Carl, 2008.

[13] J. Weber, *Automotive Development Processes*. Springer-Verlag, 2009.

[14] *Automotive SPICE Process Assessment / Reference Model*, 3rd ed., VDA QMC Working Group 13 and Automotive SIG, Berlin, Germany, 7 2015. [Online]. Available: http://www.automotivespice.com/

[15] ISO, "Software and systems engineering - software testing - part 1: Concept and definitions," *ISO/IEC/IEEE 29119-1:2013(E)*, pp. 1–68, Sept 2013.

[16] B. W. Boehm, "Software engineering," *IEEE Transactions on Computers*, vol. 25, no. 12, pp. 1226–1241, Dec. 1976. [Online]. Available: http://dx.doi.org/10.1109/TC.1976.1674590

[17] H. Shokry and M. Hinchey, "Model-based verification of embedded software," *Computer*, vol. 42, no. 4, pp. 53–59, 4 2009.

[18] A. Albers, T. Düser, O. Sander, C. Roth, and J. Henning, "X-in-the-loop-framework für fahrzeuge, steuergeräte und kommunikationssysteme," *ATZ elektronik*, vol. 5, pp. 60–65, Oct. 2010.

[19] J. Bach, J. Langner, S. Otten, M. Holzäpfel, and E. Sax, "Data-driven development, a complementing approach for automotive systems engineering," in *2017 IEEE International Symposium on Systems Engineering (ISSE)*, 2017.

[20] J. Bach, K.-L. Bauer, M. Holzäpfel, M. Hillenbrand, and E. Sax, "Control based driving assistant functions test using recorded in field data," in *7. Tagung Fahrerassistenzsysteme*, 2015. [Online]. Available: https://mediatum.ub.tum.de/node?id=1285215

[21] J. Bach, J. Langner, S. Otten, M. Holzäpfel, and E. Sax, "Test scenario selection for system-level verification and validation of geolocation-dependent automotive control systems," in *23rd International Conference on Engineering, Technology and Innovation (ICE/IEEE)*, 2017.

[22] Apache, "Hadoop documentation," http://hadoop.apache.org.

[23] Amazon, "Amazon web services overview," https://aws.amazon.com.

[24] MathWorks, "Matlab documentation," https://de.mathworks.com/help/matlab.