Verification and Validation of Automated Systems' Safety and Security

V&V methods for SCP evaluation of automated systems

Document Type	Report
Document Number	D3.1
Primary Author(s)	Jose Luis de la Vara (UCLM), Rupert Schlick (AIT)
Document Date	2020-12-18
Document Version	1.2 Final
Dissemination Level	Public (PU)
Reference DoA	2020-05-05
Project Coordinator	Behrooz Sangchoolie, behrooz.sangchoolie@ri.se, RISE Research Institutes of Sweden
Project Homepage	www.valu3s.eu
JU Grant Agreement	876852

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.

Disclaimer

The views expressed in this document are the sole responsibility of the authors and do not necessarily reflect the views or position of the European Commission. The authors, the VALU3S Consortium, and the ECSEL JU are not responsible for the use which might be made of the information contained in here.

Project Overview

Manufacturers of automated systems and the manufacturers of the components used in these systems have been allocating an enormous amount of time and effort in the past years developing and conducting research on automated systems. The effort spent has resulted in the availability of prototypes demonstrating new capabilities as well as the introduction of such systems to the market within different domains. Manufacturers of these systems need to make sure that the systems function in the intended way and according to specifications which is not a trivial task as system complexity rises dramatically the more integrated and interconnected these systems become with the addition of automated functionality and features to them.

With rising complexity, unknown emerging properties of the system may come to the surface making it necessary to conduct thorough verification and validation (V&V) of these systems. Through the V&V of automated systems, the manufacturers of these systems are able to ensure safe, secure and reliable systems for society to use since failures in highly automated systems can be catastrophic.

The high complexity of automated systems incurs an overhead on the V&V process making it time-consuming and costly. VALU3S aims to design, implement and evaluate state-of-the-art V&V methods and tools in order to reduce the time and cost needed to verify and validate automated systems with respect to safety, cybersecurity and privacy (SCP) requirements. This will ensure that European manufacturers of automated systems remain competitive and that they remain world leaders. To this end, a multi-domain framework is designed and evaluated with the aim to create a clear structure around the components and elements needed to conduct V&V process through identification and classification of evaluation methods, tools, environments and concepts that are needed to verify and validate automated systems with respect to SCP requirements.

In VALU3S, 13 use cases with specific safety, security and privacy requirements will be studied in detail. Several state-of-the-art V&V methods will be investigated and further enhanced in addition to implementing new methods aiming for reducing the time and cost needed to conduct V&V of automated systems. The V&V methods investigated are then used to design improved process workflows for V&V of automated systems. Several tools will be implemented supporting the improved processes which are evaluated by qualification and quantification of safety, security and privacy as well as other evaluation criteria using demonstrators. VALU3S will also influence the development of safety, security and privacy standards through an active participation in related standardisation groups. VALU3S will provide guidelines to the testing community including engineers and researchers on how the V&V of automated systems could be improved considering the cost, time and effort of conducting the tests.

VALU3S brings together a consortium with partners from 10 different countries, with a mix of *industrial partners* (25 partners) from automotive, agriculture, railway, healthcare, aerospace and industrial automation and robotics domains as well as leading *research institutes* (6 partners) and *universities* (10 partners) to reach the project goal.

Consortium

RISE RESEARCH INSTITUTES OF SWEDEN AB	RISE	Sweden
STAM SRL	STAM	Italy
FONDAZIONE BRUNO KESSLER	FBK	Italy
KNOWLEDGE CENTRIC SOLUTIONS SL - THE REUSE COMPANY	TRC	Spain
UNIVERSITA DEGLI STUDI DELL'AQUILA	UNIVAQ	Italy
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO	ISEP	Portugal
UNIVERSITA DEGLI STUDI DI GENOVA	UNIGE	Italy
CAMEA, spol. s r.o.	CAMEA	Czech
IKERLAN S. COOP	IKER	Spain
R G B MEDICAL DEVICES SA	RGB	Spain
UNIVERSIDADE DE COIMBRA	COIMBRA	Portugal
VYSOKE UCENI TECHNICKE V BRNE - BRNO UNIVERSITY OF TECHNOLOGY	BUT	Czech
ROBOAUTO S.R.O.	ROBO	Czech
ESKISEHIR OSMANGAZI UNIVERSITESI	ESOGU	Turkey
KUNGLIGA TEKNISKA HOEGSKOLAN	KTH	Sweden
STATENS VAG- OCH TRANSPORTFORSKNINGSINSTITUT	VTI	Sweden
UNIVERSIDAD DE CASTILLA - LA MANCHA	UCLM	Spain
FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	FRAUNHOFER	Germany
SIEMENS AKTIENGESELLSCHAFT OESTERREICH	SIEMENS	Austria
RULEX INNOVATION LABS SRL	RULEX	Italy
NXP SEMICONDUCTORS GERMANY GMBH	NXP-DE	Germany
PUMACY TECHNOLOGIES AG	PUMACY	Germany
UNITED TECHNOLOGIES RESEARCH CENTRE IRELAND, LIMITED	UTRCI	Ireland
NATIONAL UNIVERSITY OF IRELAND MAYNOOTH	NUIM	Ireland
INOVASYON MUHENDISLIK TEKNOLOJI GELISTIRME DANISMANLIK SANAYI VE TICARET LIMITED SIRKETI	IMTGD	Turkey
ERGUNLER INSAAT PETROL URUNLERI OTOMOTIV TEKSTIL MADENCILIK SU URUNLER SANAYI VE TICARET LIMITED STI.	ERARGE	Turkey
OTOKAR OTOMOTIV VE SAVUNMA SANAYI AS - OTOKAR AS	OTOKAR	Turkey
TECHY BILISIM TEKNOLOJILERI DANISMANLIK SANAYI VE TICARET LIMITED SIRKETI - TECHY INFORMATION TECHNOLOGIESAND CONSULTANCY LIMITED COMPANY	TECHY	Turkey
ELECTROTECNICA ALAVESA SL	ALDAKIN	Spain
INTECS SOLUTIONS SPA	INTECS	Italy
LIEBERLIEBER SOFTWARE GMBH	LLSG	Austria
AIT AUSTRIAN INSTITUTE OF TECHNOLOGY GMBH	AIT	Austria
E.S.T.E. SRL	ESTE	Italy
NXP SEMICONDUCTORS FRANCE SAS	NXP-FR	France
BOMBARDIER TRANSPORTATION SWEDEN AB	BT	Sweden
QRTECH AKTIEBOLAG	QRTECH	Sweden
CAF SIGNALLING S.L	CAF	Spain
PERCEIVE3D SA	P3D	Portugal
MONDRAGON GOI ESKOLA POLITEKNIKOA JOSE MARIA ARIZMENDIARRIETA S COOP	MGEP	Spain
INFOTIV AB	INFOTIV	Sweden
BERGE CONSULTING AB	BERGE	Sweden

Executive Summary

Within the scope of WP3 - Design of SCP (Safety, Cybersecurity, and Privacy) V&V (Verification and Validation) methods for automated systems, D3.1 reviews state-of-the-art and state-of-the-practice V&V methods that are relevant to VALU3S. These methods are currently applied or could be applied in the project use cases and can improve how SCP requirements are addressed, ensured, and confirmed. Fifty-three methods are described by presenting their name, purpose, description, and tool support. The strengths and limitations of the methods are also analysed and the methods are mapped to other VALU3S assets, namely the evaluation scenarios of the use cases and the multi-dimensional layered framework. Relationships between the methods and with standards are specified. References and keywords are provided for each method.

For ease of reading, we have divided the SCP V&V methods for automated systems into groups:

- Injection-based V&V
  - Fault injection
  - Attack injection
- Simulation
- Testing
- Runtime verification
- Formal verification
  - Formal source code verification
  - General formal verification
- Semi-formal analysis
  - SCP-focused semi-formal analysis
  - General semi-formal analysis
- System-type-focused V&V

The reviewed methods cover a wide range of SCP evaluation needs of automated systems, from source code analysis and behaviour assessment to earlier needs in a system´s lifecycle such as quality analysis during design. The methods cover both formal and non-formal V&V and exploit different means such as models and ontologies. Some methods have been proposed in prior projects, e.g. AMASS (https://www.amass-ecsel.eu/) and MegaMart2 (https://megamart2-ecsel.eu/).

The review of V&V methods in this deliverable will guide the work of the next WP3 efforts, most notably (1) method gap analysis (Task 3.2) and (2) method development and improvement (Task 3.3).

Contributors

Jose Luis de la Vara	UCLM	Rupert Schlick	AIT
Arturo García	UCLM	Luis Alonso	TRC
Peter Folkesson	RISE	Pierluigi Pierini	INTECS
Silvia Mazzini	INTECS	Fabio Patrone	UNIGE
Tomas Vojnar	BUT	Fredrik Warg	RISE
Ales Smrcka	BUT	Jorge Valero	UCLM
Martin Skoglund	RISE	Georgios Giantamidis	UTRCI
Stefano Tonetta	FBK	Rui Melo	P3D
Stylianos Basagiannis	UTRCI	Bernhard Fischer	SIEMENS
Cristóvão Sousa	P3D	Alberto Griggio	FBK
Henrique Madeira	COIMBRA	A.Taha Arslan	TECHY
José Fonseca	COIMBRA	David Pereira	ISEP
José Proença	ISEP	Marco Bozzano	FBK
Johnny Öberg	KTH	Íñigo Elguea	ALDAKIN
Giorgio Malaguti	ESTE	Joseba Agirre	MGEP
Walter Tiberti	UNIVAQ	Leire Etxeberria	MGEP
Francesco Smarra	UNIVAQ	Gürol Çokünlü	OTOKAR
Joakim Rosell	RISE	Thanh Bui	RISE
Markus Borg	RISE	Rosemary Monahan	NUIM
Thomas Bauer	FHG	Ugur Yayan	IMTGD
Michael Doescher	NXP-DE	Giovanni Gaggero	UNIGE
Metin Ozkan	ESOGU	Davide Ottonello	STAM
Sandra König	AIT	Dejan Nickovic	AIT
Sebastian Chlup	AIT	Otto Brechelmacher	AIT
Nuno Laranjeiro	COIMBRA	Frederico Cerveira	COIMBRA
Pascual González	UCLM	Rafael Morales	UCLM
Raul Barbosa	COIMBRA	Betul Elcin Erdogan	IMTGD
Mustafa Karaca	IMTGD	Mateen Malik	RISE
Bernd Bredehorst	PUMACY	Hamid Ebadi	INFOTIV
Ricardo Ruiz	RGB	Lars Borchardt	BERGE
Kalle Ngo	KTH	Lukáš Maršík	CAMEA
Giorgio Malaguti	ESTE	Franco Fresolone	AIT
Luigi Pomante	UNIVAQ	Alessandro D'Innocenzo	UNIVAQ
Jimmy Tjen	UNIVAQ	Alper Kanak	ERARGE
Salih Ergün	ERARGE	Behrooz Sangchoolie	RISE

Reviewers

Fredrik Warg	RISE	2020-11-23
Wolfgang Herzner	AIT	2020-11-19
Willibald Krenn	AIT	2020-11-24
Rupert Schlick	AIT	2020-11-25
Enrico Ferrari	RULEX	2020-11-25
Tomas Vojnar	BUT	2020-12-16
Behrooz Sangchoolie	RISE	2020-12-17, 2020-12-18

Revision History

Version	Date	Author (Affiliation)	Comment
0.1	2020-06-05	J.L. de la Vara (UCLM)	Initial ToC
0.2	2020-09-09	J.L. de la Vara (UCLM) et al.	Initial content and V&V method review examples
0.3	2020-10-12	J.L. de la Vara (UCLM) et al.	Inclusion of sections for V&V methods and description of further methods
0.4	2020-10-26	J.L. de la Vara (UCLM) et al.	Extension of general content and of V&V method description
0.5	2020-11-09	J.L. de la Vara (UCLM) et al.	Complete draft for analysis (completeness, consistency, etc.) before release for review
0.6	2020-11-16	J.L. de la Vara (UCLM) et al.	Deliverable ready for internal review
0.7	2020-12-04	J.L. de la Vara (UCLM) et al.	Deliverable ready for intermediary approval
0.8	2020-12-15	J.L. de la Vara (UCLM) et al.	Deliverable ready for final approval
1.0	2020-12-18	J.L. de la Vara (UCLM) et al.	Deliverable ready for EC submission
1.1	2020-12-18	Behrooz Sangchoolie	Review of the final draft while making minor formatting changes.
1.2	2020-12-18	Behrooz Sangchoolie	Final version to be submitted.

Table of Contents

Chapter 1 Introduction 17
Chapter 2 Background 19
Chapter 3 V&V Methods 27
3.1 Injection-Based V&V 31
3.1.1 Attack Injection 31
3.1.2 Fault Injection 38
3.2 Simulation 49
3.2.1 Simulation-Based Robot Verification 50
3.2.2 Simulation-Based Testing for Human-Robot Collaboration 52
3.2.3 Test Optimization for Simulation-Based Testing of Automated Systems 54
3.2.4 Virtual Architecture Development and Simulated Evaluation of Software Concepts 56
3.2.5 Virtual & Augmented Reality-Based User Interaction V&V and Technology Acceptance 58
3.2.6 V&V of Machine Learning-Based Systems Using Simulators 60
3.3 Testing 62
3.3.1 Behaviour-Driven Model Development and Test-Driven Model Review 63
3.3.2 Assessment of Cybersecurity-Informed Safety 65
3.3.3 Machine Learning Model Validation 67
3.3.4 Model-Based Mutation Testing 70
3.3.5 Model-Based Robustness Testing 72
3.3.6 Model-Based Testing 74
3.3.7 Risk-Based Testing 75
3.3.8 Signal Analysis and Probing 78
3.3.9 Software Component Testing 79
3.3.10 Test Parallelization and Automation 80
3.4 Runtime Verification 82
3.4.1 Dynamic Analysis of Concurrent Programs 82
3.4.2 Runtime Verification Based on Formal Specification 85
3.4.3 Test Oracle Observation at Runtime 88
3.5 Formal Verification 90
3.5.1 Formal Source Code Verification 90

3.5.2 General Formal Verification ... 97
3.6 Semi-Formal Analysis ... 107
3.6.1 SCP-Focused Semi-Formal Analysis ... 107
3.6.2 General Semi-Formal Analysis ... 128
3.7 System-Type-Focused V&V ... 141
3.7.1 CPU Verification ... 141
3.7.2 Penetration Testing of Industrial Systems ... 144
3.7.3 Failure Detection and Diagnosis (FDD) in Robotic Systems ... 145
3.7.4 Model-Based Formal Specification and Verification of Robotic Systems ... 148
Chapter 4 Conclusion ... 152
References ... 153
Appendix AMapping of the V&V Methods to the Multi-Dimensional Layered Framework ... 155

List of Figures

Figure 1.1 VALU3S focus on improving the V&V processes 17
Figure 2.1 Taxonomy of safety evidence [12] 21
Figure 3.1 Relationships between the groups of SCP V&V methods 31
Figure 3.2 Model-based fault injection 43
Figure 3.3 Architecture of the distributed platform [VUR1] 59
Figure 3.4 Methodology of tests that could support establishing cybersecurity-informed safety in an ADS 66
Figure 3.5 Datasets usage in ML model design, training and testing phases 68
Figure 3.6 Good vs. Bad Behaviours 88
Figure 3.7 Steps of Behaviour-Driven Formal Model Development 98
Figure 3.8 Illustration of the HISA method 109
Figure 3.9 TinyWIDS State estimation 111
Figure 3.10 Fault injection for safety analysis 116
Figure 3.11 Scheme of the algorithm underlying scenario-based cross-sectorial quantitative risk analysis 122
Figure 3.12 The 4-step evaluation process for cryptographic modules 123
Figure 3.13 Core elements of the RSHP information representation language 131
Figure 3.14 Ontology structure for Knowledge-centric system artefact quality analysis 131
Figure 3.15 Core elements of the RSHP information representation language 133
Figure 3.16 Main elements for Knowledge-centric traceability management [KCT1] 134
Figure 3.17 General process for model-based assurance and certification [MAC3] 136
Figure 3.18 Classification of FDD approaches 146
Figure 3.19 Components of a robotic system and mapping of FDD approaches 146
Figure 3.20 Examples of the modular structure of robotic system software: (a) General Control Scheme for mobile robot systems [BMF3], (b) General Control Scheme for Industrial robot system within UC11 149

List of Tables

Table 2.1 Previous projects related to VALU3S 22
Table A.1 V&V Method classification according to the Evaluation Environment dimension of the VALU3S framework 155
Table A.2 V&V Method classification according to the Evaluation Type dimension of the VALU3S framework 156
Table A.3 V&V Method classification according to the Type of Component under Evaluation dimension of the VALU3S framework 158
Table A.4 V&V Method classification according to the Evaluation Tool dimension of the VALU3S framework 161
Table A.5 V&V Method classification according to the Evaluation Stage dimension of the VALU3S framework 163
Table A.6 V&V Method classification according to the Type of the Component under Evaluation dimension of the VALU3S framework 164
Table A.7 V&V Method classification according to the Type of Requirement under Evaluation dimension of the VALU3S framework 168
Table A.8 V&V Method classification according to the Evaluation Performance Indicator dimension of the VALU3S framework 170

Acronyms

AADL	Architecture Analysis and Design Language
ACC	Adaptive Cruise Control
ADAS	Advanced Driver-Assistance System
ADS	Automated Driving System
AEB	Automatic Emergency Braking
AI	Artificial Intelligence
ARP	Address Resolution Protocol
CCA	Cause-consequence Analysis
CCD	Cause-consequence Diagrams
CPS	Cyber-Physical System
DoS	Denial of Service
EMI	Electromagnetic Interference
FDIR	Fault Detection, Isolation and Recovery
FLA	Failure Logic Analysis
FMEA	Failure Modes and Effects Analysis
FPGA	Field-Programmable Gate Array
FTA	Fault Tree Analysis
HARA	Hazard Analysis and Risk Assessment
HC	Healing Core
HiL	Hardware In the Loop
HMI	Human Machine Interaction
HRI	Human-Robot Interaction
IDS	Intrusion Detection System
KPI	Key Performance Indicator
LKA	Lane Keeping Aid
MBSA	Model-Based Safety Analysis
MIAI	Model-Implemented Attack Injection
MIFI	Model-Implemented Fault Injection
MiL	Model In The Loop
MiTM	Man In The Middle
ML	Machine Learning
NMT	Neuromuscular Transmission
NOC	Network-on-Chip

PLC	Programmable Logic Controller
PSD	Power Supply Distribution
RAM	Random Access Memory
REST	Representational state transfer
RSHP	RelationSHiP
RTR	Run-Time Reconfigurable
SCP	Safety, Cybersecurity, and Privacy
SEM	Soft-Error Mitigation
SEU	Single-Event Upset
SiL	Software in-the-Loop
SMV	Symbolic Model Verifier
SOAP	Simple Object Access Protocol
SoC	System on Chip
SUT	System Under Test
SWIFI	Software Implemented Fault Injection
UAV	Unmanned Aerial Vehicle
V&V	Verification and Validation
VHiL	Virtual Hardware in The Loop
VR	Virtual Reality
WPM	Weak Process Model
WSN	Wireless Sensor Network
WSDL	Web Services Description Language

Chapter 1 Introduction

As the use and complexity of automated systems are growing, system manufacturers and component suppliers require methods that help them to confirm that the SCP requirements of the systems are satisfied. This is necessary so that the systems can be deemed dependable. From a general perspective, a method corresponds to a particular procedure for accomplishing or approaching something, especially a systematic or established one [14]; in the scope of VALU3S, for automated systems' V&V. The methods in which we are interested are those that can be applied in some V&V activity to produce some V&V artefact. For example, according to ISO 26262 [10], fault injection can be applied in software unit verification as a part of the production of a software verification report. It is essential that V&V methods are effective, and it is very important that they are efficient.

In VALU3S, the aim of WP3 (Design of SCP V&V methods for automated systems) is to create a set of reference methods for automated system V&V. The set of methods includes V&V methods in the strict sense as well as methods providing the basis for V&V activities, like security and safety analysis. The set of reference methods will contain (1) commonly-used methods as well as (2) methods that will be improved and (3) new methods that will be created by combination of methods, chain of methods, and marriage (co-usage) of different methods. With an overview of methods currently available and of the scenarios that we wish to cover in the project's use cases, WP3 will identify the gaps between the methods that are available and the ones that are needed for V&V of the components in the use cases. As a final step we will address these gaps by improving existing V&V methods and developing new ones. This is a key element of the VALU3S focus (Figure 1.1).

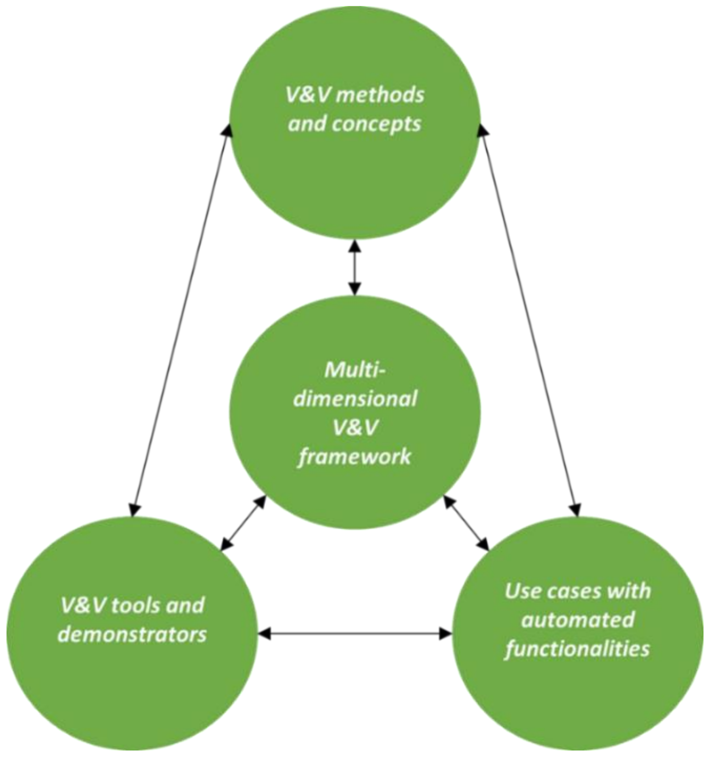

V&V methods and concepts
Multi-dimensional V&V framework
V&V tools and demonstrators
Use cases with automated functionalities

Figure 1.1 VALU3S focus on improving the V&V processes

D3.1 represents the first step towards the achievement of the purpose of WP3. The deliverable reviews commonly-used as well as state-of-the-art experimental and analytical V&V methods useful for evaluation of SCP requirements of automated systems. An applicability analysis has also been conducted to map the methods identified to the different layers of VALU3S' multi-dimensional V&V framework. In addition, D3.1 sets the basis for VALU3S KPI3 "Improve at least 14 V&V methods in order to create VALU3S repository of improved V&V methods". Among all the V&V methods reviewed, we will have to show by the end of the project that at least 14 have been improved.

Other reviews of V&V methods can be found in sources such as software and systems engineering bodies of knowledge [8,9], software engineering standards in general [7] and for critical systems in particular [5], functional safety standards [6], books [4], papers [20], project deliverables [1], and the Web [3]. These reviews completement the one presented in this deliverable, which focuses on V&V methods that VALU3S partners are particularly interested in. On the other hand, the other reviews typically cover a wider set of methods, but the analysis is shallower.

D3.1 relates to other VALU3S' deliverables that either provide input or will use D3.1 as a basis for their development:

- D1.1 (Description of use cases as well as scenarios) [15] and D1.2 (SCP requirements as well as identified test cases) [16] provide the use case scenarios to consider for application of the V&V methods reviewed in D3.1.
- D2.1 (Initial multi-dimensional layered framework) [17] and D2.2 (Final multi-dimensional layered framework) [18] specify the framework that VALU3S will propose for V&V method characterisation, which is used in D3.1 for method review.
- D3.2 (Updated web-based repository, linking state-of-the-art V&V Methods to use cases and scenarios), D3.3 (Identified gaps and limitations of the V&V methods listed in D3.1), D3.4 (Initial description of methods designed to improve the V&V process), D3.5 (Interim description of methods designed to improve the V&V process), and D3.6 (Final description of methods designed to improve the V&V process) will largely base their work on the insights provided in D3.1.
- D6.5 (Initial report on the results of the standardisation survey (methods, tools, concepts suggested by the standards)) [19] provides input in relation to the methods from standards in which VALU3S partners are interested.

The following chapters introduce the background of the deliverable (Chapter 2), review a selection of V&V methods (Chapter 3), and present our main conclusions (Chapter 4). Appendix A synthesises the mapping of the reviewed V&V methods to the multi-dimensional layered framework as a way to show general method classifications according to different perspectives, i.e. according to the different dimensions of the framework.

Chapter 2 Background

Different concepts and areas affect D3.1 and are part of the basis for its development and understanding. These concepts and areas characterise what needs to be considered for analysing, as well as developing, V&V methods in VALU3S. The concepts and areas are presented below using selected definitions and references. Nonetheless, we acknowledge that the information can be different in other sources, e.g. the standard from which definitions are chosen[1].

From a formal perspective, and according to ISO/IEC/IEEE 24765 [11], the key concepts can be defined as follows:

- **V&V**: the process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfil the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements.
- **Verification**: confirmation, through the provision of objective evidence, that specified requirements have been fulfilled.
- **Validation**: confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled.

Verification can be roughly defined as the activity to confirm that someone has built a system right, and validation as the activity to confirm that someone has built the right system.

V&V encompasses different methods and areas, and different system artefacts are involved. For example, Figure 2.1 presents a taxonomy of safety evidence, i.e. of artefact types that contribute to developing confidence that a system can be deemed safe, in compliance with applicable standards. On the one hand, the taxonomy includes specific V&V artefact types such as Testing Results, Simulation, and Model Checking. On the other hand, all the artefact types are subject to V&V, as any artefact created during the lifecycle of a highly-critical systems must undergo activities to confirm their suitability.

Typical general V&V areas, and their definitions [11], are:

- **Conformity assessment**: demonstration that specified requirements relating to a product, process, system, person, or body are fulfilled.
- **Formal specification**: specification that is used to prove mathematically the validity of an implementation or to derive mathematically the implementation.
- **Inspection**: visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications.
- **Quality assurance**: all the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfil requirements for quality.

[1] A glossary of terms for VALU3S is currently under development

- **Simulation**: process of developing or using a model that behaves or operates like a given system when provided a set of controlled inputs.
- **Static analysis**: process of evaluating a system or component based on its form, structure, content, or documentation.
- **Testing**: activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.

The specific V&V areas of interest in VALU3S include:

- Fault and attack injection
- Run-time verification
- Machine learning-based analysis
- Model checking
- Formal requirements analysis
- Model-based safety analysis
- Contract-based design
- Model-based functional and non-functional properties verification
- Fault injection with HiL
- Modelling of context awareness via intelligible analytics
- Model-based automated test case generation & execution
- Compliance- and certification-targeted activities
- Hardware-based attacks through cryptographic modules
- Intrusion Detection Systems
- Risk assessment
- Simulation

These method areas and specialisations are reviewed in Chapter 3.

Prior projects and their results are also relevant for our analysis of V&V methods. The projects developed methods from which VALU3S results will build, by directly applying the methods and thus extending their application, or by enhancing the methods towards improved automated system V&V. Table 2.1 includes projects in which VALU3S partners participated and whose results are a basis for the work in VALU3S. The relation of the projects with VALU3S is summarised and specific V&V methods relevant to VALU3S are listed.

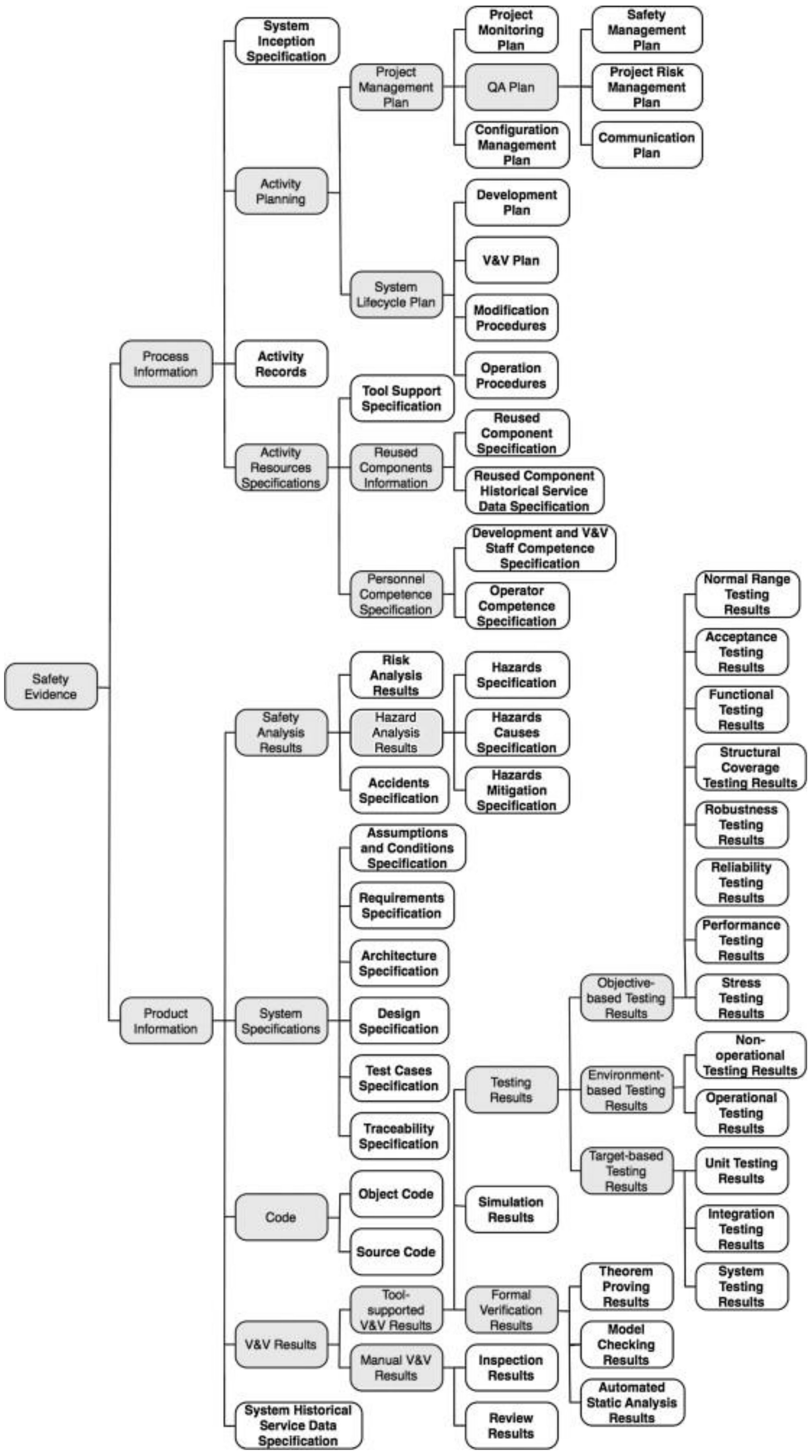

Safety Evidence
Process Information
Product Information
System Inception Specification
Activity Planning
Activity Records
Activity Resources Specifications
Project Management Plan
System Lifecycle Plan
Project Monitoring Plan
QA Plan
Configuration Management Plan
Safety Management Plan
Project Risk Management Plan
Communication Plan
Development Plan
V&V Plan
Modification Procedures
Operation Procedures
Tool Support Specification
Reused Components Information
Reused Component Specification
Reused Component Historical Service Data Specification
Personnel Competence Specification
Development and V&V Staff Competence Specification
Operator Competence Specification
Safety Analysis Results
Risk Analysis Results
Hazard Analysis Results
Accidents Specification
Hazards Specification
Hazards Causes Specification
Hazards Mitigation Specification
System Specifications
Assumptions and Conditions Specification
Requirements Specification
Architecture Specification
Design Specification
Test Cases Specification
Traceability Specification
Code
Object Code
Source Code
V&V Results
Tool-supported V&V Results
Manual V&V Results
System Historical Service Data Specification
Testing Results
Simulation Results
Formal Verification Results
Inspection Results
Review Results
Objective-based Testing Results
Environment-based Testing Results
Target-based Testing Results
Theorem Proving Results
Model Checking Results
Automated Static Analysis Results
Normal Range Testing Results
Acceptance Testing Results
Functional Testing Results
Structural Coverage Testing Results
Robustness Testing Results
Reliability Testing Results
Performance Testing Results
Stress Testing Results
Non-operational Testing Results
Operational Testing Results
Unit Testing Results
Integration Testing Results
System Testing Results

Figure 2.1 Taxonomy of safety evidence [12]

Table 2.1 Previous projects related to VALU3S

Project	Relation
ADVANCE **(**Addressing Verification and Validation Challenges in Future Cyber-Physical Systems) https://cordis.europa.eu/project/id/823788 https://www.advance-rise.eu/	Techniques, methods, and tools applicable to different phases of the system lifecycle, with the objective of improving the effectiveness and efficacy of the V&V process in CPS. *Methods*: System modelling for V&V, testing, fault injection, fault forecast, and systematic processes like FMEA
AI4EU (A European AI On Demand Platform and Ecosystem) https://www.ai4eu.eu/	Verification of automated/autonomous systems, AI, safety, privacy and security *Methods*: Fault injection to produce data failure.
AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of CPS) http://www.amass-ecsel.eu/	Multi-concern, multi-domain, certification, CPS *Methods*: Model-based assurance & certification, Knowledge-centric system artefact quality analysis, Knowledge-centric traceability management, Multi-concern Failure Logic Analysis (FLA), V&V in Model-Based Design
AQUAS (Aggregated Quality Assurance for Systems) http://aquas-project.eu/	Safety, security, performance co-Engineering, interaction points concept *Methods*: Modelling of performance, Safety and security assessment, Lightweight static analysis, Formal analysis and verification, Dynamic analysis, Model-Based Threat Analysis
ARAMIS I https://www.projekt-aramis.de/	Embedded systems safety, verification of concurrent behaviour *Methods*: Virtual architecture development and simulated evaluation of software concepts
ARAMIS II https://www.aramis2.com/	Processes, platforms, and tools for testing and verification of embedded systems in automotive industry and avionics *Methods*: Virtual architecture development and simulated evaluation of software concepts
AutARK (Autonomous assistance system for supporting HRC in assembly processes) https://www.autark-projekt.de/	Safety in human-robot-interaction, data steam analytics and machine learning *Methods*: Machine Learning methods used for Data Stream Analytics

Project	Relation
AutoDrive (Advancing fail-aware, fail-safe, and fail-operational electronic components, systems, and architectures for fully automated driving to make future mobility safer, affordable, and end-user acceptable) https://autodrive-project.eu/	Fault-tolerant systems development *Methods*: Model-Based Mutation Testing (Use of event-structures)
CRYSTAL (Critical System Engineering Acceleration) http://www.crystal-artemis.eu	V&V of safety-critical embedded/cyber-physical systems, safety engineering. *Methods*: Model-Based Mutation Testing (for Event-B)
EMBEET (Environment for Model-Based Embedded Systems Engineering and Testing) http://embeet.com/	V&V of embedded systems, model-based testing *Methods*: Behaviour-Driven Model Development and Test-driven Model Review
EMC2 (Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments) https://www.artemis-emc2.eu/	Safe and secure platform development of CPS *Methods*: The platform tool NSG will be used to create multi-core FPGA platforms in UC10
ENABLE-S3 (European Initiative to Enable Validation for Highly Automated Safe and Secure Systems) https://www.enable-s3.eu/	Consideration of safety and security for highly automated systems, V&V patterns *Methods*: Runtime Monitoring and Verification, Behaviour-Driven Formal Model Development
HEAVENS (HEAling Vulnerabilities to ENhance Software Security and Safety) https://www.vinnova.se/en/p/heavens-healing-vulnerabilities-to-enhance-software-security-and-safety/	Automotive security, functional safety, security modelling, interplay between safety and security *Methods*: Model-based fault injection and attack injection
HoliSec (Holistic Approach to Improve Data Security) https://autosec.se/holisec-results/	Automotive security, mechanisms, interplay between safety and security, V&V *Methods*: Model-based fault injection and attack injection
in.nav (Advanced Intra-Operative Navigation in Arthroscopy Surgery) https://perceive3d.com/en/products/innav	V&V of AI-based medical device *Methods*: Runtime verification and static analysis of software for surgical navigation
KARYON (Kernel-based ARchitecture for safetY-critical cONtrol)	New paradigms for embedded systems, monitoring and control towards complex

Project	Relation
https://cordis.europa.eu/project/id/288195	systems engineering Smart vehicles talk to each other *Methods*: Fault injection in wireless communication
MegaMart2 (MegaModelling at Runtime - scalable model-based framework for continuous development and runtime validation of complex systems) https://megamart2-ecsel.eu/	Framework incorporating methods and tools for continuous development and validation *Methods*: Back-trace analysis of runtime logs on model elements
REASSURE http://www.cister.isep.ipp.pt/projects/reassure/	Runtime verification, secure runtime monitoring architectures, domain-specific languages, safety properties, CPS *Methods*: Runtime Monitoring and Verification
SAFECOP (Safe Cooperating CPS using Wireless Communication) http://www.safecop.eu/	Safety and security strategies for cooperative CPS *Methods*: WPM-based Intrusion Detection System for WSN platforms, Runtime Monitoring and Verification, Contract-based safety and security analysis
SafePower (Safe and secure mixed-criticality systems with low power requirements) http://safepower-project.eu/	Safety and security of mixed-critical systems, FPGA system *Methods*: Fault Injection in FPGAs
SECREDAS (Cyber Security for Cross Domain Reliable Dependable Automated Systems) https://secredas-project.eu/	Safety, security, privacy, automated systems, cross-domain, standardisation *Methods*: Model-implemented attack injection
SESAMO (SEcurity and SAfety MOdelling) http://sesamo-project.eu/	Safety and security modelling, methods, tools *Methods*: Model-based methodology jointly addressing safety and security aspects
SMILE II (Safety analysis and verification/validation of MachIne LEarning based systems) https://www.vinnova.se/en/p/smile-ii---safety-analysis-and-verificationvalidation-of-machine-learning-based-systems/	Safe AI V&V for AI Simulation *Methods*: Verification and validation of machine learning based systems using simulators
UPTIME (Unified Predictive Maintenance System) https://www.uptime-h2020.eu/	Predictive maintenance through data analytics in manufacturing and aviation *Methods*: Data Mining and Data Stream Analytics

Project	Relation
VeTeSS (Verification and Testing to Support Functional Safety Standards) https://artemis-ia.eu/project/43-vetess.html	Automotive safety, V&V methods, embedded systems, simulation-based testing *Methods*: Test port-based fault injection, Model-implemented fault injection

Chapter 3 V&V Methods

This chapter includes the review of V&V methods that are relevant to VALU3S. A sub-chapter has been created for each method group. As mentioned in the introduction, other reviews of V&V methods can be found in the literature.

For V&V method review, a template has been defined with the following fields:

- *Name*: a set of words that identifies the method.
- *Purpose*: the reason for which the method can be used.
- *Description*: a textual characterisation of the method.
- *Relationship with other methods*: the connections that exist between a given method and others in this Chapter, e.g., specialisation or the fact that a method can be used as a part of another.
- *Tool support*: the software tools that support method usage.
- *Layers of the multi-dimensional framework*: a characterisation of the method according to the layers of the framework.
- *Use case scenarios*: the scenarios in which the method is or could be applied.
- *Strengths*: the main advantageous quality aspects of the method.
- *Limitations*: the main disadvantageous quality aspects of the method.
- *References*: other sources where information about the method can be found.
- *Related standards*: industry standards whose scope the method could be used in, e.g., functional safety standards that recommend the method or some generalisation of the method.
- *Keywords*: a list of concepts that characterise the method.

It must be noted that the information provided about the methods in some fields might not be complete yet or might evolve. For instance, the mappings to the multi-dimensional framework and the use case scenarios might evolve as these VALU3S assets change. Further relationships between methods might be determined in the future when possible combinations are studied in more depth.

The dimensions and layers of framework[2] considered for mapping are:

- Evaluation environment
  - In the lab
  - Closed
  - Open
- Evaluation type
  - Experimental – Monitoring
  - Experimental – Simulation
  - Experimental – Testing
  - Analytical – Formal
  - Analytical – Semi-Formal
- Type of component under evaluation

[2] Framework version on November 9th, 2020

-
    - Model
    - Software
    - Hardware
- Evaluation tool
    - Proprietary
    - Open Source
- Evaluation stage
    - Validation
    - Verification
- Logic of the component under evaluation
    - Sensing
    - Thinking
    - Acting
- Type of requirements under evaluation
    - Functional
    - Non-functional – Safety
    - Non-functional – Cybersecurity
    - Non-functional – Privacy
    - Non-functional – Others
- Evaluation performance indicator
    - V&V process criteria
    - SCP criteria

Details about the dimensions and the layers can be found in D2.1 [17] and D2.2 [18].

The use case scenarios[3] are:

- Aerospace use cases
    - VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
    - VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
    - VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
    - VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery
- Agriculture use cases
    - VALU3S_WP1_Agriculture_1 - Vehicle switching from parallel guidance to manual mode
    - VALU3S_WP1_Agriculture_2 - Vehicle switching from manual mode to parallel guidance
    - VALU3S_WP1_Agriculture_3 - Transmission line disturbances
    - VALU3S_WP1_Agriculture_4 - Disturbances in IMU communication
- Automotive use cases
    - VALU3S_WP1_Automotive_1 - Radar/camera advanced detection and tracking
    - VALU3S_WP1_Automotive_2 - Radar + camera cooperation

[3] Scenarios version on November 9th, 2020

 - VALU3S_WP1_Automotive_3 - Node connection to cloud
 - VALU3S_WP1_Automotive_4 - Transmission line under different performance conditions
 - VALU3S_WP1_Automotive_5 - Transmission line optimal settings
 - VALU3S_WP1_Automotive_6 - Transmission line switching
 - VALU3S_WP1_Automotive_7 - Safety of vehicle during switch between routers
 - VALU3S_WP1_Automotive_8 - Automatic Emergency Braking (AEB)
 - VALU3S_WP1_Automotive_9 - Failure detection of Software and Hardware subsystem components
 - VALU3S_WP1_Automotive_10 - System certification
 - VALU3S_WP1_Automotive_11 - System performance
 - VALU3S_WP1_Automotive_12 - ADAS system has to reliable and has to comply with Safety standards
 - VALU3S_WP1_Automotive_13 - SoC validation with intensive use of SoC internal self-tests
- Healthcare use cases
  - VALU3S_WP1_Healthcare_1 - Bone segmentation baseline performance
  - VALU3S_WP1_Healthcare_2 - Safety analysis and certification
  - VALU3S_WP1_Healthcare_3 - Certification needs of the NMT device
  - VALU3S_WP1_Healthcare_4 - HiL and SiL benchmark platform
  - VALU3S_WP1_Healthcare_5 - Patient modelling with NMT drugs
  - VALU3S_WP1_Healthcare_6 - Assurance needs of the NMT device
  - VALU3S_WP1_Healthcare_7 – V&V for the NMT controller
  - VALU3S_WP1_Healthcare_8 – Generalization of bone segmentation to typical conditions
  - VALU3S_WP1_Healthcare_9 – Robustness of bone segmentation to challenging conditions
- Industrial Robotics/Automation use cases
  - VALU3S_WP1_Industrial_1 - Manipulation of sensor data
  - VALU3S_WP1_Industrial_2 - Server and PLC communication
  - VALU3S_WP1_Industrial_3 - Safety trajectory optimization
  - VALU3S_WP1_Industrial_4 - Anomaly detection at component and system level
  - VALU3S_WP1_Industrial_5 - Motor speed control
  - VALU3S_WP1_Industrial_6 - Fault tolerance for motor position sensor data
  - VALU3S_WP1_Industrial_7 - Safety behaviour for missing motor position sensor data
  - VALU3S_WP1_Industrial_8 - Safety behaviour for remote control terminal connection failure
  - VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal
  - VALU3S_WP1_Industrial_10 - Localization of human
  - VALU3S_WP1_Industrial_11 - Handling and gripping of product/parts
  - VALU3S_WP1_Industrial_12 - Knocking off product/part from robot gripper by human worker

  - VALU3S_WP1_Industrial_13 - Corruption of input/output signal at robot gripper.
  - VALU3S_WP1_Industrial_14 - Data manipulation in human-robot-interaction
  - VALU3S_WP1_Industrial_15 – Worker position/action monitoring
  - VALU3S_WP1_Industrial_16 – Recognition of workers' voice commands
  - VALU3S_WP1_Industrial_17 – Responses to external control devices
  - VALU3S_WP1_Industrial_18 – AI capabilities to work in the system
- Railway use cases
  - VALU3S_WP1_Railway_1 - Inject, detect and recover
  - VALU3S_WP1_Railway_2 - Controlled vs random injection
  - VALU3S_WP1_Railway_3 - Systematic and random failures verification
  - VALU3S_WP1_Railway_4 - Railway signal detector

This list of use case scenarios corresponds to those that have been considered for D3.1 preparation. They have been extracted from D1.1 [15] and D1.2 [16]. The set of scenarios might evolve during the project.

The methods have been divided into several groups to ease reading. It must be noted that the groups are not exclusive, and some methods could be included in more than one group. In these cases, VALU3S partners have selected the group regarded as the most representative one. Different groups could be selected if other criteria were used. For example, someone could decide that all the methods that use some formal method should be classified as formal verification. The methods are ordered in the groups alphabetically, thus the order does not reflect e.g. method importance.

Figure 3.1 synthesises the relationships that have been specified between the V&V methods reviewed by showing what groups include methods that relate to those of another group. For example, Knowledge-centric system artefact quality analysis is a general semi-formal method that relates to Formal verification methods. The figure depicts the connections between V&V methods that VALU3S is currently interested in. The relationships are not shown in the figure at method level because in many cases the relationships have been established at group level in the reviews below to indicate connections between a method and a set of methods.

As can be observed, the abstraction level of the reviewed methods is heterogeneous. Some methods refer to very specific procedures, e.g. Knowledge-centric traceability management. Others refer to procedures for which specialisations exist, e.g. Model checking. This is an intentional feature of the review of V&V methods that is presented. In some cases, VALU3S partners have a very focused work area for the methods to address during the project, whereas other partners have a clear general work area, but details still need to be defined. This also relates to the fact that the work on some methods stems from specific needs e.g. from tool vendors and a more concrete focus can already be established. A different situation corresponds to methods and method areas for which larger research is needed, including on the selection of the methods to best tackle VALU3S challenges.

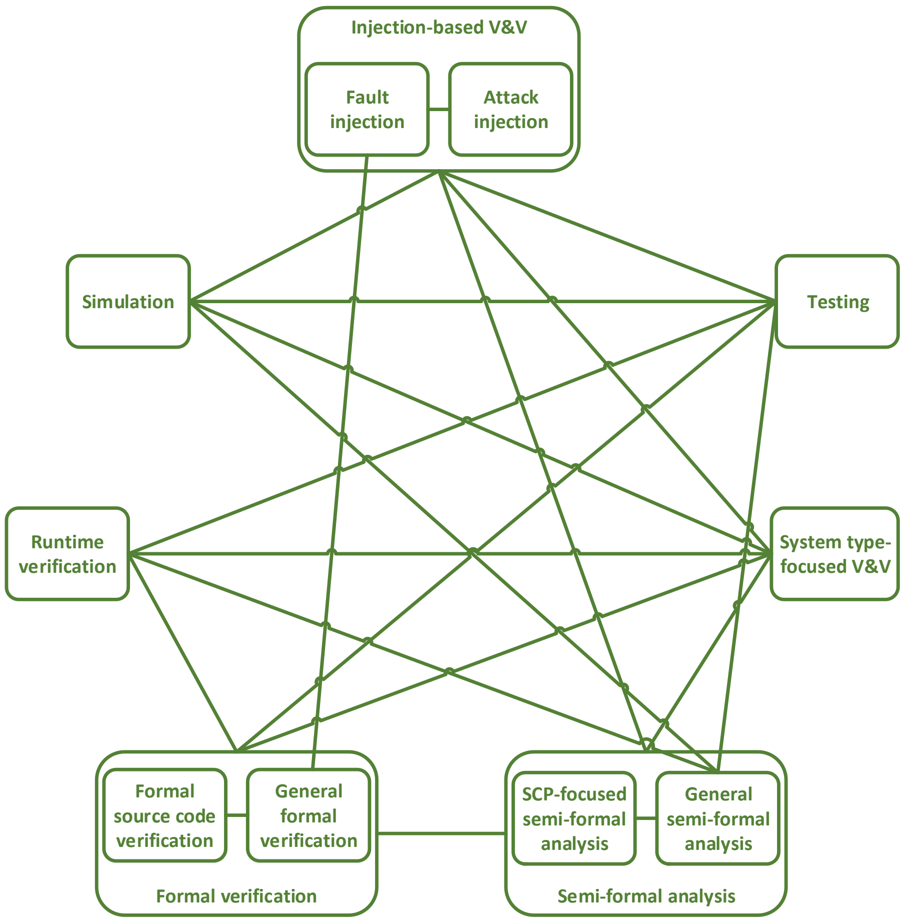


Figure 3.1 Relationships between the groups of SCP V&V methods

3.1 Injection-Based V&V

This group of methods focuses on introducing certain characteristics in a system, providing a certain type of input, or triggering certain events, to confirm that the system behaves suitably under the corresponding conditions. Two types of injection are considered: attack injection and fault injection.

3.1.1 Attack Injection

In cybersecurity, an attack is an intentional human-made (manual or automatic) effort to deliberately breach a system from external system boundaries. The system breach exploits the vulnerabilities in a system and could result into a compromised system. The compromised system could result in a system failure such as, software or hardware complete failure or degraded performance. In terms of safety and security, there could be a direct correlation between a fault and an attack. Avizienis et al. [2] define an attack as a special type of fault which is human made, deliberate and malicious, affecting hardware or software from external system boundaries and occurring during the operational phase. Thus, attack injection in a system is analogous to fault injection, see Section 3.1.2. However, the aim is to evaluate the impact of cybersecurity attacks on the overall security of a system.

This sub section is comprised of three attack injection methods. *Model-Implemented Attack Injection* and *Simulation-based Attack Injection at System-level* evaluate the system's cybersecurity properties by targeting system simulations. The former method uses attack models injected as part of the system models to perform the attacks while the latter method uses a simulator to control the injection of the attacks. The method *Vulnerability and attack injection* on the other hand focuses on injecting attacks into the actual physical systems to assess their cybersecurity properties.

3.1.1.1 Model-Implemented Attack Injection

Name: **Model-Implemented Attack Injection**
Purpose: Model-Implemented Attack Injection (MIAI) is defined as a technique where attack injection mechanisms are developed as model blocks which can be inserted into simulated system models at an early development phase. The purpose of MIAI is to evaluate the impact of cybersecurity attacks on system security.
Description: Attacks can be defined as human made, intentional malicious activity to effect hardware or software from external system boundaries during the operational phase of a system [MIA1]. MIAI is a model-based test and verification framework which enables to test and evaluate the impact of cybersecurity threats by injecting attack models into the target system model in early design and development phases [MIA3]. The MIAI methodology supports the use of cybersecurity attack models capable of jamming, replay, denial of service and intercept, etc. [MIA2] [MIA3] [MIA4].
Relationship with other methods: Model-implemented attack injection is related to methods such as *Intrusion Detection System for WSN, Wireless interface network security assessment* and *Simulation-based attack injection at system-level.*
Tool support: MODIFI (https://www.ri.se/en/what-we-do/expertises/fault-injection-and-attack-injection)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Experimental - Simulation • Type of component under evaluation: Model • Evaluation tool: Proprietary • Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Non-functional – Cybersecurity • Evaluation performance indicator: V&V Process criteria, SCP criteria
Use case scenarios • VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults • VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation • VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
Strengths • MIAI is aligned with the shift-left approach where the focus of the test and verification activities are shifted towards the early design and development process to find and improve the weaknesses of the software as much as possible and as early as possible with less effort and resources [MIA5].

- MIAI is used for testing and verification of the cybersecurity of the simulated model of the intended software. This gives an early evaluation of the software behaviour under the presence of attacks.
- MIAI gives valuable input to the design allowing the development engineers to get a holistic view of the cybersecurity bottlenecks.
- MIAI can be used to evaluate the intrusion detection and handling mechanisms as well as system behaviour under the presence of attacks.
- Measurements from MIAI may be useful in later V&V.

Limitations

- The MIAI is limited to the attack injection on the simulation level (simulation-based attack injection). It is not possible to evaluate the actual physical system. There are other techniques used to inject attacks on physical level such as vulnerability attack injection.
- Accuracy of the attack models with respect to the actual attacks in the physical system may not be adequate.
- Any change in the system design in the later stages of the product development cycle might decrease the usefulness of the measurements from the attack model and cannot be used for the comparison of the results between verification and validation stages.

References

- [MIA1] B. Sangchoolie, P. Folkesson, and J. Vinter, "A study of the interplay between safety and security using model-implemented fault injection," in 2018 14th Eur. Dep. Comp. Conf. (EDCC). IEEE, 2018, pp. 41–48.
- [MIA2] B. Sangchoolie, P. Folkesson, Pierre Kleberger and J. Vinter, "Analysis of Cybersecurity Mechanisms with respect to Dependability and Security Attributes," in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops.
- [MIA3] P. Folkesson, B. Sangchoolie, and J. Vinter, "HoliSec D3.3 - Interplay between Safety, Security and Privacy." The HoliSec Consortium, Mar. 19, 2019.
- [MIA4] https://www.microsoft.com/security/blog/2007/09/11/stride-chart/
- [MIA5] Bjerke-Gulstuen K., Larsen E.W., Stålhane T., Dingsøyr T. (2015) High Level Test Driven Development – Shift Left. In: Lassenius C., Dingsøyr T., Paasivaara M. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2015. Lecture Notes in Business Information Processing, vol 212. Springer, Cham. https://doi.org/10.1007/978-3-319-18612-2_23

Related standards: ISO-TC22-SC32-WG11_N0613_ISO_SAE_DIS_21434_(E), NIST 800, IEC 62443, SAE J3061, IEC TR 63069, IEC TR 63074, ISO TR 22100-4, ISO 24089

Keywords: Cybersecurity, Attack injection, Fuzzing, Penetration testing, Vulnerability scanning, Threat, Attack models, Intrusion.

3.1.1.2 Simulation-Based Attack Injection at System-level

Name: **Simulation-based attack injection at system-level**
Purpose: The purpose of simulation-based attack injection at system-level is to evaluate system's cybersecurity properties by injecting attacks, e.g., using simulator control commands during target system simulations.

Description: System-level simulation is comprised of hardware and software models of a cyber physical system (CPS). The attack injection could be performed on different abstraction layers such as logical, functional, hardware, software, or system level etc. In this case, we focus on the simulation-based attack injection on system level.
In our case the simulation-based attack injection at system level is done through injection of attacks on systems modelled in traffic simulators. Simulators of interest are SUMO and CARLA [SAI2] [SAI3]. The simulation-based attack injection at system level using simulators may be used for security testing of automated systems such as autonomous vehicles [SAI1].
Simulation-based attack injection is a V&V method where the attacks are injected into system software in a simulation environment. This type of attack injection is applicable when:

- A software is available to run in a simulation environment. This type of testing is called Software in the Loop (SiL) testing and the software under evaluation is called SiL component*[1] [SAI4]
- The hardware is not available.
- The software needs to be verified and validated in a simulation environment.

Simulation-based attack injection is useful for both development and deployment stages to identify and resolve different types of vulnerabilities relevant for each stage.

*[1] A SiL component is an executable code written for a specific system, adjusted to run only in a simulation environment for software testing. This type of testing is useful especially when the hardware is not existing, when it is in the development phase, or when the verification results are required in short span of time. The latter could be facilitated by parallel execution of the tests in a cluster. Hardware requirements are taken away (e.g., end-to-end protection) from the SiL component so that it can run in a completely simulated or model-based environment. Note that the SiL testing is complemented by Hardware in the Loop (HiL) testing, when the hardware is available, in order to also evaluate the system when the software resides in the intended hardware, such as a particular mechatronic system.

Relationship with other methods: *Simulation-based attack injection at system-level* method is related to almost all simulation-based methods in this deliverable, such as *Simulation-based robot verification, Simulation-based fault injection at system-level, Intrusion Detection System for WSN,* and *Wireless interface network security assessment*.

Tool support: SUMO (Simulation of Urban MObility; https://www.eclipse.org/sumo/), CARLA (Open-source simulator for autonomous driving research; https://carla.org/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental-Simulation
- Type of component under evaluation: Software
- Evaluation tool: Open source
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional – Cybersecurity
- Evaluation performance indicator: V&V Process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Automotive_4 - Transmission line under different performance conditions

• VALU3S_WP1_Automotive_7 - Safety of vehicle during switch between routers
Strengths The simulation-based attack injection at the system level can be useful for: • End-to-end resilience assessment of a complete system specially in edge case scenarios*1 • Introducing attacks in different parts of a system such as sensors, functions, and actuators to evaluate that specific part or even a complete system behaviour. • Introducing attacks in automated systems, which may be hard to do through other verification methods. • It is possible to introduce multiple attacks by using this method. • Measurements from simulation-based attack injection may be useful in later V&V activities. *1The edge cases are realised by injecting attacks in the system to create a test scenario which is otherwise rarely tested or testable in the real-world.
Limitations • The simulation-based attack injection at system level is limited to the injection of attack in simulations only, so it is not possible to evaluate the actual physical system. • The use of simulation-based attack injection techniques for ML based systems showed promising results in the initial experimentation. However, there is a need to further explore this test technique for ML or deep learning-based systems.
References • [SAI1] Eduardo dos Santos et al., "Towards a Simulation-based Framework for the Security Testing of Autonomous Vehicles" • [SAI2] Michael Behrisch, Laura Bieker et al., "SUMO – Simulation of Urban Mobility, An Overview", Institute of Transportation Systems, German Aerospace Center, Rutherfordstr. 2, 12489 Berlin, Germany. • [SAI3] Alexey Dosovitskiy, German Ros et al., "CARLA: An Open Urban Driving Simulator". • [SAI4]https://www.add2.co.uk/applications/sil/#:~:text=The%20term%20'software%2Din%2D,prove%20or%20test%20the%20software.
Related standards: ISO 26262, IEC 62061, IEC TR 63074, ISO PAS 21448, ISO 13849, IEC 61508, ISO/IEC TR 24028:2020, ISO/IEC WD 23053
Keywords: Safety, Fault injection, Fault insertion, Failure injection, Fuzzing, Safety verification, Fault modelling, Fault handling, Data corruption, Communication, Tele-operation.

3.1.1.3 Vulnerability and Attack Injection

Name: **Vulnerability and attack injection**
Purpose: The purpose of Vulnerability and Attack Injection (VAI), in real systems or prototypes, is to evaluate globally how the system copes with attacks and to assess specific security mechanisms in the target systems. The security problem of web applications is addressed by analysing typical web application vulnerabilities, how developers fix them, and how attackers exploit them, in order to implement a realistic Vulnerability and Attack Injector Tool that can be used to automate the process of evaluating security mechanisms in custom scenarios.

Description: The methodology consists of injecting realistic vulnerabilities in a component exposed to the Internet and subsequently exploiting such vulnerabilities to launch attacks automatically in order to evaluate existing security mechanisms in the entire system [VAI1, VAI2]. That is, a component is specifically injected with realistic vulnerabilities to make it possible to evaluate security attributes and mechanisms in the rest of the system, through the attacks that become possible through the injected vulnerabilities. The component where the vulnerabilities are injected (target component) is not part of the target system under evaluation. The injected vulnerabilities are considered realistic because they are derived from the extensive field study on real web application vulnerabilities presented in [VAI3], and are injected according to a set of representative restrictions and rules defined in [VAI4]. The whole approach is considered realistic since web applications (or any component exposed to the Internet) may have vulnerabilities, and even so the possible exploitation of such vulnerabilities to attack the system must be detected and handled adequately.

The automated attack of a web application is a multi-stage procedure that includes: preparation stage, vulnerability injection stage, attack load generation stage, and attack stage. These stages are described in the next paragraphs.

In the *Preparation Stage*, the web application is interacted (crawled) executing all the functionalities that need to be tested. Meanwhile, both HTTP and SQL communications are captured by two probes and processed for later use. The interaction with the web application is always done from the client's point of view (the web browser). The outcome of this stage is the correlation of the input values, the HTTP variables that carry them and their respective source code files, and its use in the structure of the database queries sent to the back-end database (for SQL injection - SQLi) or displayed back to the web browser (for cross-site scripting - XSS). Later on, in the Attack Stage, the malicious activity applied is based on tweaking the values of the variables, which correspond to the text fields, combo boxes, etc., discovered in this Preparation Stage.

In the *Vulnerability Injection Stage*, the vulnerabilities are injected into the web application. For this purpose, it needs information about which input variables carry relevant information that can be used to execute attacks to the web application. This stage starts by analysing the source code of the web application files searching for locations where vulnerabilities can be injected. The injection of vulnerabilities is done by removing the protection of the target variables, like the call to a sanitizing function. This process follows the realistic patterns resulting from the field study presented in [VAI3]. Once it finds a possible location, it performs a specific code mutation in order to inject one vulnerability in that particular location. The change in the code follows the rules derived from [VAI3], which are described and implemented as a set of Vulnerability Operators presented in [VAI4].

After having the set of copies of the web application source code files with vulnerabilities injected, we need to generate the collection of malicious interactions (attackloads) that will be used to attack each vulnerability. This is done in the *Attackload Generation Stage*. The attackload is the malicious activity data needed to attack a given vulnerability. This data is built around the interaction patterns derived from the Preparation Stage, by tweaking the input values of the vulnerable variables. This stage also generates the payload footprints that have a one-to-one relationship with the attack payloads. The payload footprints are the expected result of the attack. They can be the malicious SQL queries text sent to the database, for the case of an SQLi attack; or the HTML of the web application response, for the case of a XSS attack. These payload footprints are fundamental, since they are used to assess the success of the attack.

In the *Attack Stage*, the web application is, once again, interacted. However, this time it is a "malicious" interaction since it consists of a collection of attack payloads in order to exploit the vulnerabilities injected. The attack intends to alter the SQL query sent to the database server of the web application (for the case of SQLi attacks) or the HTML data sent back to the user (for the case of XSS attacks). The vulnerable source code files (from the Vulnerability Injection Stage) are applied to the web application, one at a time. Once again, the two probes for capturing the HTTP and SQL communications are deployed and the collection of attackloads is submitted to exploit the vulnerabilities injected. The interaction with the web application is always done from the web client's point of view (the web browser) and the attackload is applied to the input variables (the text fields, combo boxes, etc., present in the web page interface). At the end of the attack, we assess if the attack was successful. The detection of the success of the attack is done by searching for the presence of the payload footprint in the interaction data (HTTP or SQL communications) captured by the two probes. The process is repeated until all the injected vulnerabilities have been attacked.

Relationship with other methods: The vulnerability injection is a particular case of a *Software-implemented fault injection* (SWIFI). The attack injection is a kind of simulation of an attack using real components and systems (as opposed to models). The two elements together (i.e., vulnerability injection in an Internet exposed component + attacks launched through the exploitation of such vulnerabilities) make this approach particularly effective.

Tool support: VAIT Infection

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental - Testing
- Type of component under evaluation: Software
- Evaluation tool: Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Thinking
- Type of requirements under evaluation: Non-functional- Cybersecurity
- Evaluation performance indicator: V&V Process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_4 - Anomaly detection at component and system level

Strengths

- Injects realistic vulnerabilities
- Attacks the vulnerabilities based on the characteristics of the vulnerabilities, like an attacker would do
- Allows testing security mechanisms in place
- Allows training security teams

Limitations

- Needs to access the source code of the application or system
- For the current tool, the application is web-based and must be written in PHP
- Heavily dependent on the programming language of the target application
- Only two vulnerabilities (SQL Injection and XSS) are implemented so far, but other injection type vulnerabilities are likely to be easily implemented as well

References

- [VAI1] J. Fonseca, N. Seixas, M. Vieira, and H. Madeira, "Analysis of Field Data on Web Security Vulnerabilities", IEEE Transactions on Dependable and Secure Computing, accepted for publication in 2014.
- [VAI2] J. Fonseca, M. Vieira, and H. Madeira, "Evaluation of Web Security Mechanisms using Vulnerability & Attack Injection", IEEE Transactions on Dependable and Secure Computing, accepted for publication in 2014.
- [VAI3] J. Fonseca and M. Vieira, "Mapping Software Faults with Web Security Vulnerabilities," Proc. IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2008, June 2008.
- [VAI4] J. Fonseca, M. Vieira, and H. Madeira, "Training Security Assurance Teams using Vulnerability Injection", Proc. IEEE Pacific Rim Dependable Computing Conference, PRDC 2008, December 2008
- [VAI5] Neves, N., Antunes, J., Correia, M., Veríssimo, P., Neves R., "Using Attack Injection to Discover New Vulnerabilities", the 36th Annual IEEE/IFIP International Conference Dependable Systems and Networks, 2006.
- [VAI6] Powell, D., Stroud, R., "Conceptual Model and Architecture of MAFTIA", Project MAFTIA, deliverable D21, 2003.
- [VAI7] SPI Dynamics Inc., http://www.spydynamics.com/products/webinspect/, 2008
- [VAI8] Watchfire Corporation, http://www.watchfire.com, 2009
- [VAI9] Acunetix Web Vulnerability Scanner, 2008.
- [VAI10] webSPHINX, http://www.cs.cmu.edu/~rcm/websphinx/, 2012
- [VAI11] Robert A., Web Application Scanners Comparison. Available: http://www.cgisecurity.com/2009/01/web-application-scanners-comparison.html, 2009.
- [VAI12] Elia, I., Fonseca, J., Vieira, M., "Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental Study", The 21st annual International Symposium on Software Reliability Engineering (ISSRE 2010), November, 2010

Related standards: IEC TR 63074, ISO/IEC TR 24028:2020, ISO/IEC 27001

Keywords: Cybersecurity, Vulnerability injection, Attack injection, Realistic vulnerabilities.

3.1.2 Fault Injection

Fault injection consists of the deliberate insertion of artificial (yet realistic) faults in a computer system or component in order to assess its behaviour in the presence of faults and allow the characterization of specific dependability measures and/or fault tolerant mechanisms available in the system. According to the well-known concepts and terminology proposed by Avizienis[2], a fault is the "adjudged or hypothesized cause of an error", and an "error is the part of the total state of the system that may lead to its subsequent service failure". In other words, the faults injected may lead to errors that, subsequently, may cause erroneous behaviour of the target component. These errors may propagate in the system and may cause failures in other components or even system failures. Fault injection can be seen as an approach to accelerate the occurrence of faults in order to help in the verification and validation of fault handling mechanisms available in the system under evaluation.

Fault injection can be used in different phases of the systems development to evaluate (or even predict) how systems and specific components behave in the presence of faults, or to assess dependability properties such as safety, security, availability or reliability. Typically, faults injected in models (structural or behaviour-based models) are useful in the early stages of system development, while faults injected in prototypes or in real systems in controlled experiments allow the verification and validation of actual properties of deployed systems.

In general, there is a clear agreement among researchers and practitioners on the fault models used to represent the different types of faults. In a very condensed way, those faults models include the following types:

- Hardware faults: single or multiple bit flips, typically injected at processor register level.
- Software faults: small code changes that represent the most common types of software faults (i.e., bugs) found in the field in deployed software.
- Interface faults: corruption of input parameters defined according to the domain of such parameters
- Software vulnerabilities: small changes that mimic the most common types of vulnerabilities.

The rest of this subsection presents six fault injection methods: *Fault-Injection in FPGAs*, *Interface Fault Injection*, *Model-Based Fault Injection for Safety Analysis*, *Model-Implemented Fault Injection*, *Simulation-based Fault Injection at System-level*, and *Software-Implemented Fault Injection*.

3.1.2.1 Fault Injection in FPGAs

Name: **Fault injection in FPGAs**
Purpose: To explore and evaluate the results of fault injection in an FPGA-based Hardware Platform and its propagation to other system layers. To relate the fault injection on the low-level hardware layer to potential faults on the higher layers to reduce test space.
Description: Healing Core IP (HC IP) is a means that can inject faults at any desired place in an FPGA Design. It uses Soft-Error Mitigation (SEM)-cores to detect errors and uses Run-Time Reconfiguration (RTR) techniques to correct Single- and Multiple-Event Upsets (bit-flips) in the FPGA's configuration memory. Further, it has a classification system that can report and initiate appropriate countermeasures for some faults. Thus, it can be used to implement self-repairing functionality in an FPGA system. The HC IP can be used to inject, detect and fix faults in the HW designs resident in the FPGA. By flipping bits in the configuration RAM of the FPGA, we can see how the system reacts, how the fault manifests on the system level, and how it affects up-time, robustness and availability of the component, and evaluate what it implies according to relevant safety standards. Based on historical data, the V&V process should minimize the steps needed for (re-)certification.
Relationship with other methods: *Model-implemented fault and attack injection* - The Fault-injection on FPGAs config-ware complements other Fault-injection methods targeted for software. Since the injection methods only works in the programmable logic part of the FPGA, Software-methods for injecting faults must be used in hard-core parts of the FPGA (the ARM processor system).
Tool support: Xilinx Vivado tool suite (www.xilinx.com), NoC System Generator (https://github.com/Noctegra/NSG)

Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Experimental – Monitoring, Experimental – Simulation, Experimental – Testing • Type of component under evaluation: Hardware, Software • Evaluation tool: Proprietary • Evaluation stage: Validation • Logic of the component under evaluation: Sensing, Thinking, Action • Type of requirements under evaluation: Non-functional - Safety • Evaluation performance indicator: V&V process criteria, SCP criteria
Use case scenarios • VALU3S_WP1_Railway_1 - Inject, detect and recover • VALU3S_WP1_Railway_2 - Controlled vs random injection • VALU3S_WP1_Railway_3 - Systematic and random failures verification
Strengths Fault injection in FPGAs allows to: • Inject, Detect and Heal Faults caused by Single-Event Upsets in the FPGA fabric. This will allow to assess the effects of Single-Event Upsets in the Hardware Platform and see how it will manifest itself, both in Software and on System Level. • Inject, Detect and Heal Faults during Run-Time. This will be useful in V&V, when assessing the Safety of the FPGA System. Fault injection in FPGAs paired with the Healing Core has the potential to: • Enable usage of FPGAs in Safety Critical Functions in a System. This will lead to reduced Time, Cost, and Efforts when developing Safe Hardware Platforms.
Limitations • Using the Healing Core has some Time, Cost and Effort overhead associated with it. At the moment it is unclear if the costs of using it outweigh the costs of not using it. • The Healing Core is also subject to Faults. • There may be faults in the FPGA that cannot be healed without resetting and rebooting the entire FPGA, resulting in system downtime.
References • [FIF1] E. Kyriakakis, K. Ngo, J. Öberg, "Mitigating Single-Event Upsets in COTS SDRAM using an EDAC SDRAM Controller", In Proc. of 2017 IEEE Nordic Circuits and Systems Conference (NorCAS-2017), Linköping, Sweden, Oct 24-25, 2017. • [FIF2] E. Kyriakakis, K. Ngo, J. Öberg, "Implementation of a Fault-Tolerant, Globally-Asynchronous-Locally-Synchronous, Inter-Chip NoC Communication Bridge on FPGAs", In Proc. of 2017 IEEE Nordic Circuits and Systems Conference (NorCAS-2017), Linköping, Sweden, Oct 24-25, 2017. • [FIF3] K. Ngo, T. Mohammadat, J. Öberg, "Towards a Single Event Upset Detector Based on COTS FPGA", In Proc. of 2017 IEEE Nordic Circuits and Systems Conference (NorCAS-2017), Linköping, Sweden, Oct 24-25, 2017. • [FIF4] Öberg, J., Robino, F., "A NoC System Generator for the Sea-of-Cores Era", In Proc. of FPGAWorld 2011, Copenhagen, Stockholm, Munich, September, 2011, ACM Digital Libraries.
Related standards: ISO 13849, ISO 26262, EN 50129, IEC 62061, IEC 61508

Keywords: Safety, Fault injection, Fault insertion, Failure injection, Safety validation, Safety verification, Fault modelling, Fault handling, Fault propagation, Soft-error mitigation.

3.1.2.2 Interface Fault Injection

Name: **Interface fault injection**
Purpose: Interface fault injection, frequently known as robustness testing, consists in the injection of faults at the interface of components (OS calls, APIs, services…) through data corruption at interface level [IFI1], with the purpose of evaluating the behaviour of the system or component under test in the presence of invalid inputs or stressful interface conditions [IFI42].
Description: Interface faults (or robustness testing) requires that the system/component under test faces erroneous input conditions, which are usually defined based on typical developer mistakes or wrong assumptions. Erroneous input conditions can be also generated at random in some robustness testing scenarios. In a more general fault injection context, erroneous inputs injected at the interface of a given component can represent failures in preceding components that forward their erroneous outputs to the target component. Information regarding the system interface (e.g., a WSDL document in case of SOAP [IFI3] web services, or an OpenAPI document in case of REST services) is normally used as input for the generation of the set of invalid inputs, which are combined with valid parameters and sent in requests to the system under test [IFI4]. Examples of invalid parameters are cases null, empty, and boundary values, strings in special formats, or even malicious values. System responses are inspected for suspicious cases of failure (e.g., the presence of exceptions in the response or, response codes referring to internal server errors) and should be analysed regarding their severity.
Relationship with other methods: Interface fault injection (i.e., robustness testing) complements the *Fault injection* methods (SWIFI) that inject software faults inside software components. From a fault model perspective, one can claim that the injection of erroneous inputs in the interface of a component may represent failures in the components that provide inputs to the target component. Interface fault injection also complements *Attack injection* methods, as interface faults could be one of the possible consequences of cyberattacks. Another specific relationship is with *Model-based robustness testing*.
Tool support: wsrBench (http://wsrbench.dei.uc.pt/), bBOXRT (https://git.dei.uc.pt/cnl/bBOXRT)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Experimental - Testing • Type of component under evaluation: Software • Evaluation tool: Proprietary • Evaluation stage: Verification, Validation • Logic of the component under evaluation: Thinking • Type of requirements under evaluation: Non-functional - Others (Robustness) • Evaluation performance indicator: V&V Process criteria, SCP criteria
Use case scenarios • VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation • VALU3S_WP1_Healthcare_1- Bone Segmentation • VALU3S_WP1_Industrial_13 - Corruption of input/output signal at robot gripper

Strengths • Current approaches are the result of years of research; thus, they are mature and potentially allow for easy application to specific systems. • Low effort required for the generation of robustness test cases • Easiness of use and integration of current tools
Limitations • Classification of results is highly dependent on expert knowledge • The quality of generated workloads by tools often limits the disclosure of robustness problems
References • [IFI1] N. Laranjeiro, M. Vieira and H. Madeira, "Experimental Robustness Evaluation of JMS Middleware," *2008 IEEE International Conference on Services Computing*, Honolulu, HI, 2008, pp. 119-126, doi: 10.1109/SCC.2008.129. • [IFI2] J. Cámara, R. de Lemos, N. Laranjeiro, R. Ventura and M. Vieira, "Robustness-Driven Resilience Evaluation of Self-Adaptive Software Systems," in *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 50-64, 1 Jan.-Feb. 2017, doi: 10.1109/TDSC.2015.2429128. • [IFI3] N. Laranjeiro, M. Vieira, & H. Madeira, A robustness testing approach for SOAP Web services. *J Internet Serv Appl* **3,** 215–232 (2012). https://doi.org/10.1007/s13174-012-0062-2 • [IFI4] N. Laranjeiro, M. Vieira and H. Madeira, "A Technique for Deploying Robust Web Services," in *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 68-81, Jan.-March 2014, doi: 10.1109/TSC.2012.39.
Related standards: ISO 26262, IEC 62061, IEC TR 63074, ISO PAS 21448, ISO 13849, IEC 61508, ISO/IEC TR 24028:2020, ISO/IEC WD 23053
Keywords: Robustness testing, Robustness assessment, Interface fault injection, Software fault injection, SWIFI

3.1.2.3 Model-Based Fault Injection for Safety Analysis

Name: **Model-based fault injection for safety analysis**
Purpose: Analyse the behaviour of a given system in presence of faults, in the early product development phase, in order to examine the safety characteristics of the system, such as its reliability, availability, and its capability to tolerate faults.
Description: In model-based fault injection, user-specified faults may be (automatically) injected into the nominal model (i.e., without faults) of the system of interest (see Figure 3.2). In this approach, the user can specify the faults to be injected manually, or by picking them from a predefined library. For instance, the xSAP tool [MBI1] provides libraries to specify the most common failure patterns (e.g., stuck-at faults, non-deterministic faults, ramp-down, etc.), to specify their dynamics (e.g., permanent faults, sporadic faults, etc.) and to instantiate them using specific parameters. The resulting model after fault injection is called extended model and can be analysed using safety assessment techniques to produce artefacts such as FTs and FMEA tables. In xSAP, both the nominal and the extended model are written in the SMV language, and the extended model is generated automatically using the given fault specifications. Similar techniques can be used to analyse the effectiveness of fault detection, isolation, and recovery procedures. Faults to be injected may include different classes of faults (HW/SW/functional) at different levels of abstraction.

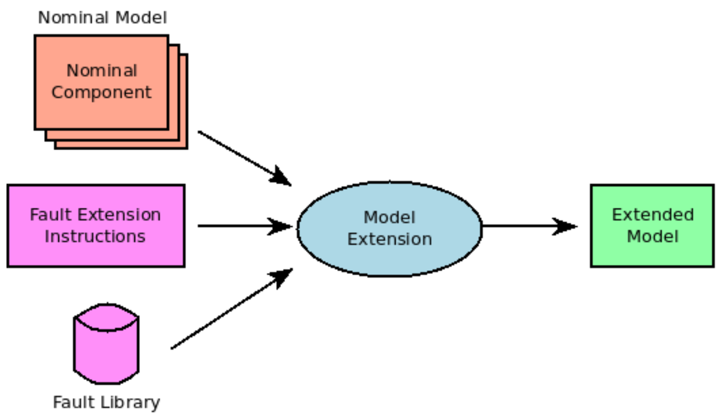


Figure 3.2 Model-based fault injection

Relationship with other methods: *Model-based safety analysis* makes use of the current method as a building block. *Model checking* is the enabling underlying technology.

Tool support: The xSAP (https://xsap.fbk.eu/) Safety Analysis Platform [MBI1] (developed by FBK), provides functionalities for automatic model extension of modes written in the SMV language. The COMPASS toolset (http://www.compass-toolset.org/) [MBI2, MBI3], based on AADL, uses xSAP as a back-end (developed by FBK in several studies funded by the European Space Agency).

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical - formal
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_ Agriculture_1 - Vehicle switching from parallel guidance to manual mode
- VALU3S_WP1_ Agriculture_2 - Vehicle switching from manual mode to parallel guidance
- VALU3S_WP1_ Agriculture_3 - Transmission line disturbances

Strengths

- Automated fault injection enables fault specification using a library and the automatic generation of the extended model, thus preventing modelling errors due to manual activities.
- Improvement of the communication between the system engineer and safety experts, by facilitating understanding of the logic and the eventual failures of the system.
- Achievement of a systematic and comprehensive safety assessment that allows to early identify the greatest number of possible critical problems related to the impact of failures on the functionality of the system.
- The method facilitates the safety assessment analysis (e.g., FTA, FMEA) on different variants of a given model, obtained by enabling only a specific subset of possible faults, thus supporting incremental verification and trade-off analyses.

Limitations

- Tool support usability can be improved, e.g., editing and customization of fault libraries.
- The automated analysis, in presence of a high number of faults, may be subject to the state-explosion problem, impacting the effectiveness of verification.

References

- [MBI1] B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli and G. Zampedri. The xSAP Safety Analysis Platform. In Proceedings of TACAS 2016. Eindhoven, The Netherlands, April 2-8, 2016.
- [MBI2] M.Bozzano, A.Cimatti, J.-P.Katoen, V. Y.Nguyen, T.Noll and M.Roveri. Safety, Dependability, and Performance Analysis of Extended AADL Models. The Computer Journal, 54(5):754-775, 2011.
- [MBI3] M. Bozzano, A. Cimatti, J.-P. Katoen, P. Katsaros, K. Mokos, V.Y. Nguyen, T. Noll, B. Postma and M. Roveri. Spacecraft Early Design Validation using Formal Methods. Reliability Engineering & System Safety 132:20-35. December 2014.

Related standards: SS-ISO_26262_2018, IEC 62061, IEC TR 63074, SOTIF, IEC 61508, ISO/IEC 24028 (under dev.), ISO/IEC WD 23053 (under dev.), SAE ARP4754, SAE ARP4761. These standards represent and guide on many aspects of safety and Artificial Intelligence (AI) such as, potential interaction between functional safety and cybersecurity, fault handling (injection, detection and tolerance), systematic faults during the dev phase, and framework for AI Systems Using Machine Learning (ML).

Keywords: Safety, Fault injection, Model extension, Model-based safety analysis.

3.1.2.4 Model-Implemented Fault Injection

Name: **Model-Implemented Fault Injection**

Purpose: The purpose of Model-Implemented Fault Injection (MIFI) is to evaluate the safety aspects of the system's design by injecting fault models directly into simulated system models in early product development phases.

Description: Model-based development is often used to develop the systems with high safety requirements [MIF2]. Model-based development refers to the modelling of the intended SW based on the initial requirements and assumptions of the system [MIF3]. That model can then be used to test and verify the initial assumptions. In the model-implemented fault injection method the System Under Test (SUT) and faults that are to be injected are modelled. The model-implemented fault injection is a subcategory of simulation-based fault injection.
Later in the model development process, the same model is used to generate the SW code, executable on the target hardware. In MIFI different types of faults models are injected into the model [MIF3], allowing the dependability requirements to be tested in early development phase.

Relationship with other methods: Model-Implemented Fault injection is related to almost all Model-based verification methods present in this deliverable in one way or another. Other methods which also have a relationship to this method are *Simulation-based verification*, *Behaviour-Driven Formal Model Development*, *Kalman filter-based fault detector* and other fault injection techniques such as *Model-Based Fault injection for Safety Analysis*, etc.

Tool support: MODIFI (https://www.ri.se/en/what-we-do/expertises/fault-injection-and-attack-injection)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental - Simulation
- Type of component under evaluation: Model
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional – Safety
- Evaluation performance indicator: V&V Process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events

Strengths

- MIFI is aligned with the shift-left approach where the focus of the test and verification activities are shifted towards the early design and development process to find and improve the weaknesses of the software as much as possible and as early as possible with less effort and resources [MIF1].
- MIFI is used for testing and verification of the robustness of the simulated model of the intended software. This gives an early evaluation of the software behaviour under the presence of faults.
- MIFI gives valuable input to the design allowing the development engineers to get a holistic view of the dependability bottlenecks.
- MIFI can be used to evaluate the error and fault detection and handling mechanisms as well as system behaviour under the presence of faults.
- Measurements from MIFI may be useful in later V&V.

Limitations

- The MIFI is limited to the fault injection on the simulation level (simulation-based fault injection). It is not possible to evaluate the actual physical system. There are other techniques used to inject faults on physical level such as SWIFI (Software Implemented Fault Injection), fault injection on pin-level, EMI (electromagnetic interference) and PSD (power supply distribution) etc
- Accuracy of the fault models w.r.t the actual faults in the physical system may not be adequate.
- To execute huge amounts of faults, a lot of computer resources are required.
- Any change in the system design in the later stages of the product development cycle might decrease the usefulness of the measurements from the model and cannot be used for the comparison of the results between verification and validation stages.

References

- [MIF1] Bjerke-Gulstuen K., Larsen E.W., Stålhane T., Dingsøyr T. (2015) High Level Test Driven Development – Shift Left. In: Lassenius C., Dingsøyr T., Paasivaara M. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2015. Lecture Notes in Business Information Processing, vol 212. Springer, Cham. https://doi.org/10.1007/978-3-319-18612-2_23
- [MIF2] R. Svenningsson, J. Vinter, H. Eriksson, and M. T¨orngren, “Modifi: A model-implemented fault injection tool,” in Proc. of the 29th Int. Conf. on Computer Safety, Reliability, and Security, ser. SAFECOMP’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 210–222.

• [MIF3] P. Folkesson, F. Ayatolahi, B. Sangchoolie, J. Vinter, M. Islam, and J. Karlsson, "Back-to-back fault injection testing in model-based development," in Computer Safety, Reliability, and Security, 2015.
Related standards: ISO 2626, IEC 62061, IEC TR 63074, ISO PAS 21448, ISO 13849, IEC 61508, ISO/IEC TR 24028:2020, ISO/IEC WD 23053
Keywords: Safety, Fault injection, Fault insertion, Error insertion, Error injection, Failure injection, Fuzzing, Model-based development, Safety verification, Fault modelling.

3.1.2.5 Simulation-Based Fault Injection at System-level

Name: **Simulation-based fault injection at system-level**
Purpose: The purpose of simulation-based fault injection at system-level is to evaluate system's dependability by injecting faults, e.g., using simulator control commands during target system simulations.
Description: System-level simulation is comprised of hardware and software models of a cyber physical system (CPS). The fault injection could be performed on different abstraction layers such as logical, functional, hardware, software or system level etc. In this case, we focus on the simulation-based attack injection on system level. In our case the simulation-based attack injection at system level is done through injection of attacks on systems modelled in traffic simulators. Simulators of interest are SUMO and CARLA [SFI3] [SFI4]. The simulation-based fault injection at system level using simulators may be used for security testing of automated systems such as autonomous vehicles [SFI2]. Simulation-based fault injection is a V&V method where faults are injected into system software in a simulation environment. This type of fault injection is applicable when: • A software is available to run in a simulation environment. This type of testing is called Software in the Loop (SiL) testing and the software under evaluation is called SiL component*1 [SFI5] • The hardware is not available. • The software needs to be verified and validated in a simulation environment. Simulation-based fault injection is useful for both development and deployment stages to identify and resolve different types of vulnerabilities relevant for each stage. *1 A SiL component is an executable code written for a specific system, adjusted to run only in a simulation environment for software testing. This type of testing is useful especially when the hardware is either not existing or when it is in the development phase or when the verification results are required in short span of time. The latter could be facilitated by parallel execution of the tests in a cluster. Hardware requirements are taken away (e.g., end-to-end protection) from the SiL component so that it can run in a completely simulated or model-based environment. Note that the SiL testing is complimented by hardware in the loop (HiL) testing, when the hardware is available, in order to also evaluate the system when the software resides in the intended hardware, such as a particular mechatronic system.
Relationship with other methods: *Simulation-based fault injection at system-level* method is related to almost all simulation-based methods present in this chapter in one way or another such as *Simulation-based robot verification* and *Simulation-based attack injection at system-level,* to name a few.

Tool support: SUMO (Simulation of Urban MObility; https://www.eclipse.org/sumo/), CARLA (Open-source simulator for autonomous driving research; https://carla.org/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental - Simulation
- Type of component under evaluation: Software
- Evaluation tool: Open source
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional – Safety
- Evaluation performance indicator: V&V Process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Automotive_4 - Transmission line under different performance conditions
- VALU3S_WP1_Automotive_7 - Safety of vehicle during switch between routers

Strengths

The simulation-based fault injection at the system level can be useful for:

- End-to-end resilience assessment of a complete system specially in edge case scenarios*[1]
- Introducing faults in different parts of a system such as sensors, functions, and actuators to evaluate that specific part or even a complete system behaviour.
- Introducing faults in automated systems, which may be hard to do through other verification methods.
- It is possible to introduce multiple faults by using this method.
- Measurements from simulation-based faults injection may be useful in later V&V activities.

*[1]The edge cases are realised by injecting attacks in the system to create a test scenario which is otherwise rarely tested or testable in the real-world.

Limitations

- The simulation-based fault injection at system level is limited to the injection of faults in simulations only, so it may not be possible to accurately evaluate the actual physical system.
- The use of simulation-based fault injection techniques for Machine Learning (ML) based systems has showed promising results in previous experiments [SFI2]. However, there is a need to further explore this test technique for ML or deep learning-based systems.

References

- [SFI1] M.-C. Hsueh, T.K. Tsai, and R.K. Iyer, “Fault Injection Techniques and Tools,” Computer, vol. 40, no. 4, pp. 75-82, Apr. 1997.
- [SFI2] S. Jha et al., “AVFI: Fault Injection for Autonomous Vehicles,” in Proc. 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 55–56.
- [SFI3] Michael Behrisch, Laura Bieker et al., “SUMO – Simulation of Urban Mobility, An Overview”, Institute of Transportation Systems, German Aerospace Center, Rutherfordstr. 2, 12489 Berlin, Germany.
- [SFI4] Alexey Dosovitskiy, German Ros et al., “CARLA: An Open Urnban Driving Simulator”.

• [SFI5]https://www.add2.co.uk/applications/sil/#:~:text=The%20term%20'software%2Din%2D,prove%20or%20test%20the%20software.
Related standards: ISO 26262, IEC 62061, IEC TR 63074, ISO PAS 21448, ISO 13849, IEC 61508, ISO/IEC TR 24028:2020, ISO/IEC WD 23053
Keywords: Safety, Fault injection, Fault insertion, Failure injection, Fuzzing, Safety verification, Fault modelling, Fault handling, data corruption, Communication, Tele-operation .

3.1.2.6 Software-Implemented Fault Injection

Name: **Software-implemented fault injection**
Purpose: The purpose of software implemented fault injection is the deliberate insertion of upsets (faults or errors) in computer systems and/or components to evaluate its behaviour in the presence of faults or validate specific fault tolerance mechanisms in the target system.
Description: Software-implemented fault injection, abbreviated as SWIFI, uses a variety of software-based techniques for inserting faults or errors in a system-under-study. These techniques may range from pre-runtime approaches, such as the modification of the binary executable or source code to introduce the fault [FIN1], to runtime approaches, such as using software traps to halt execution and inject a fault in specific locations of the processor and/or the program under execution [FIN2]. SWIFI is adaptable to various hardware architectures and can implement most fault models, including models that represent hardware faults (e.g., single and multiple bit-flips in CPU registers and memory) [FIN2] and software faults (e.g., code mutations that mimic the most common real software faults) [FIN1], [FIN3], [FIN4]. ucXception is an example of a modern framework composed of SWIFI tools that can emulate both transient hardware faults (soft errors) and software faults, using a mix of pre-runtime and runtime approaches. It can be employed to evaluate the dependability of a system, identify failure modes and estimate their probability of occurrence, validate fault tolerance mechanisms and measure various fault and error coverages, as well as latencies (see, for example, [FIN5]).
Relationship with other methods: Software implemented fault injection is a fault injection method per se, specifically useful to emulate hardware transient faults and software faults (bugs) in components and systems (prototypes and real systems). Nevertheless, the idea of using software to inject (i.e., emulate) faults is actually the approach used by all the modern *Fault injection* techniques, including *Model-based fault injection* and *Interface fault injection*.
Tool support: ucXception (https://github.com/ucx-code/ucXception)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Experimental - Testing • Type of component under evaluation: Software, Hardware • Evaluation tool: Proprietary • Evaluation stage: Verification and Validation • Logic of the component under evaluation: Thinking • Type of requirements under evaluation: Non-functional—Safety, Non-functional—Others (reliability, availability). • Evaluation performance indicator: V&V Process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Automotive_9 - Failure detection of Software and Hardware subsystem components
- VALU3S_WP1_Railway_3 - Systematic and random failures verification

Strengths

- Capable of evaluating the dependability of real systems and validating fault tolerance mechanisms
- Can significantly accelerate the generation of failure data
- Can emulate realistic hardware and software faults
- Portable to different hardware architectures, operating systems, etc.

Limitations

- Requires the existence of a prototype or a real system for evaluation
- The used fault models should be realistic and represent faults that the system may experience
- Intrusiveness in the system because of the fault injection tool may skew the results
- Extensive fault injection campaigns may take a considerable amount of time, depending on the workload and number of experiment runs

References

- [FIN1] João A. Durães and Henrique S. Madeira "Emulation of Software Faults: A Field Data Study and a Practical Approach", IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 849-867, November 2006.
- [FIN2] João Carreira, Henrique Madeira e João Gabriel Silva, "Xception: Software Fault Injection and Monitoring in Processor Functional Units", IEEE Transactions on Software Engineering, Vol.24, No.2, February de 1998.
- [FIN3] R. Natella, D. Cotroneo, and H. Madeira, "Assessing Dependability with Software Fault Injection: A Survey", ACM Computing Surveys, Volume 48 Issue 3, February 2016.
- [FIN4] R. Natella, D. Cotroneo, J. Durães, H. Madeira, "On Fault Representativeness of Software Fault Injection", IEEE Transactions on Software Engineering, vol.39, no.1, pp. 80-96, January 2013.
- [FIN5] F. Cerveira, R. Barbosa, H. Madeira and F. Araújo, "The Effects of Soft Errors and Mitigation Strategies for Virtualization Servers," in IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2020.2973146.

Related standards: ISO 26262, IEC 62061, IEC TR 63074, ISO PAS 21448, ISO 13849, IEC 61508, ISO/IEC TR 24028:2020, ISO/IEC WD 23053

Keywords: Fault injection, Error insertion, Fault models, Hardware fault injection, Software fault injection, SWIFI, Safety evaluation.

3.2 Simulation

This sub-group of methods describes the simulation-based verification and validation of selected properties. By the systematic development and exploitation of models, simulation-based approaches enable the automated execution of validation scenarios at early design phases in the project. Simulation focusses on the use of models that behave or operate like a given system to predict how the system would respond to defined inputs. By introducing simulation-based virtual validation in the development processes of technical software-intensive systems, parts of the late system integration tests with expensive hardware test equipment can be replaced by appropriate verification and validation activities at design time prior to the component implementation and integration steps. The change of the development and test processes is also called Shift Left or Front Loading. Additionally, simulation-based validation approaches have good support to parallelize execution and evaluation activities.

Simulation-based solutions deal with different challenges to efficiently enable early verification and validation. On the one hand, the approaches address specific quality properties, which involve the development of dedicated simulation components. For example, for the early validation of autonomous systems that interact with humans, appropriate models of humans and their interactions with autonomous robots have to be developed. On the other hand, the system complexity and the efficiency of simulation-based verification and validation have to be considered. The cost-efficient development of simulation components and scenarios is a pre-requisite to fully exploit the advantages of early verification and validation at system design time.

This section covers six different approaches that tackle the challenges stated above. The solution *Virtual Architecture Development and Simulated Evaluation of Software Concepts* describes an approach for the simulation-based validation and the systematic assessment of design decisions at early stages by coupling simulation models and simulators, existing code, and virtual hardware platforms. *Simulation-Based Robot Verification* focusses on the verification of safety-related properties such as trajectory optimization and anomaly detection for autonomous robot systems by simulation-based fault injection. *Simulation-Based Testing for Human-Robot Collaboration* investigates how the interactions between humans and robots can be verified by simulation-based approaches. The approach covers several quality properties such as performance and feasibility of solution concepts as well as safety and efficiency of human-robot collaborations. *Test Optimization for Simulation-Based Testing of Automated Systems* deals with the cost-driven improvement of the development and design activities for simulation scenarios. It comprises the selection, minimization, and prioritization of test cases and simulation scenarios. *V&V of Machine Learning-Based Systems Using Simulators* focusses on providing and using simulated environments for the verification and validation of perception, planning, and decision-making components of autonomous systems. The approach *Virtual & Augmented Reality-Based User Interaction V&V and Technology Acceptance* uses virtual and augmented reality technologies to validate the safety-critical aspects of interactions between humans and collaborative robots in order to create trust and acceptance.

3.2.1 Simulation-Based Robot Verification

Name: **Simulation-based robot verification**
Purpose: The purpose of simulation-based verification is assuring robot's trajectory safety for body-in-white inspection systems. With simulation-based fault injection, system performance and system behaviour will be observed. Through observations, safety of the robot trajectory is verified.
Description: In industrial operations, failure of an autonomous robot system can cause a significant hazard on their operation. To prevent these kinds of accidents and avoid possible loss of life and properties, safety of the autonomous systems should be verified. As of now, most systems are tested through field testing, which a method that is costly, time-consuming, limited in the reproducible

scenarios, and risky in case of non-acceptable behaviours. To mitigate these issues, their software can be pre-validated using simulation-based testing [SBV1].

Simulation-based testing enables simulations of autonomous system operation in a virtual environment, which resembles the real environment where the system will operate. In physical simulation environment, behaviours of an autonomous system can be monitored and safety of robot can be verified without damaging any environmental hazard [SBV1].

In autonomous system V&V operations, fault injection is one of the testing methods for the observing status and behaviour of the virtual/real system while feeding the system with specific faults. Fault injection is also considered as a validation technique for system's robustness and fault tolerance. Five different fault injection techniques can be given, namely hardware-based fault injection, software-based fault injection, simulation-based fault injection, emulation-based fault injection, and hybrid fault injection. Fault injection mechanisms have their own elements for system test and verification. These mechanisms provide different system corruption functions at compilation and run time of a system [SBV2].

In this method, simulation and observation of robot behaviour in automotive body inspection system will be focused for the purpose of safety trajectory optimization and anomaly detection at component and system level. With given requirements, a robot's safety status will be determined through simulation-based testing.

Relationship with other methods: Simulation-based robot verification is related to some simulation-based and fault injection V&V methods in this deliverable. These related methods are *Simulation based testing for Human-Robot collaboration* and *Simulation-based Fault Injection at System-level*.

Tool support: Gazebo (http://gazebosim.org/), ROS (https://www.ros.org/), IMTGD FI tool, MoveIt (https://moveit.ros.org/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental - Simulation, Experimental - Monitoring
- Type of component under evaluation: Model, Software
- Evaluation tool: Open source, Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Acting, Sensing, Thinking
- Type of requirements under evaluation: Functional, Non-functional - Safety
- Evaluation performance indicator: V&V process criteria, SCP Criteria

Use case scenarios

- VALU3S_WP1_Industrial_1 - Manipulation of sensor data
- VALU3S_WP1_Industrial_3 - Safety trajectory optimization
- VALU3S_WP1_Industrial_4 - Anomaly detection at component and system level
- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths

- Simulation-based verification can be scalable if models, scenarios and simulations are created properly. In simulation-based testing, verification activities can be done for different robots

without changing other models or tools. Also, system testing can be done without producing any physical item and adding risk to the environment. • The fault injection function for system and monitoring in simulation environment can provide powerful monitoring.
Limitations • Most of the applications do not consider physical models in verification, as simulation-based applications mostly run on hierarchical models. This narrows availability of both academic and industrial resources in development. • Simulation tools for simulation-based fault injection can require much computational power and limit real-time applications.
References • [SBV1] Timperley, C. S., Afzal, A., Katz, D. S., Hernandez, J. M., & Le Goues, C. (2018, April). Crashing simulated planes is cheap: Can simulation detect robotics bugs early?. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) (pp. 331-342). IEEE. • [SBV2] Benso, A., & Prinetto, P. (Eds.). (2003). Fault injection techniques and tools for embedded systems reliability evaluation (Vol. 23). Springer Science & Business Media.
Related standards: IEC61508
Keywords: Simulation-based testing, Fault injection, Industrial robot.

3.2.2 Simulation-Based Testing for Human-Robot Collaboration

Name: **Simulation-based testing for human-robot collaboration**
Purpose: Constraint model-based testing oracles implementation in simulation-based testing for Human-Robot collaboration systems
Description: The evolution towards the Industry 4.0 paradigm aims to increase flexibility and robustness by maintaining the level of productivity. To meet these requirements, collaboration between humans and robots is considered a basic framework within the future intelligent manufacturing cells. However, this interaction between humans and robots is a complex process that raises challenges around compatibility and operational safety. Test-based simulation for human-robot collaboration provides the opportunity to evaluate the feasibility and performance of the system, particularly the layout or workplace planning, production reliability and, especially, the safety and efficiency of human-robot collaboration. Validating the safety of an HRI (Human Robot Interaction) application is not a trivial task. The simulation is key to the validation in this type of systems [SBT1, SBT5, SBT6]. Through simulation, a high percentage of the safety requirements can be validated without putting any human at risk. In the domain of HRI applications, the relevant value space of input variables in tests and simulations can approach infinity (ill-defined domains), even more when dealing with people with different disabilities. In this context, specifying the oracles and assertions of the different tests remains a complex task. Previous works [SBT2, SBT3, SBT4] have utilized constraint-based modelling techniques to interpret and diagnose procedural task carried out by users, including humans with disabilities. The objective of this work is to integrate a framework that will act as an oracle in a simulation-based testing

environment. This simulation will provide real time diagnosis for the interaction between disabled humans and robots in a manufacturing and disassembly domain.

Relationship with other methods: This method is related to *Simulation-based robot verification*, as it is also for robots and using Gazebo as tool. It is also related to *V&V of Machine Learning-Based Systems Using Simulators* as they can be complementary. The results of the method *Test optimization for simulation-based testing of automated systems* could be used as input in this method.

Tool support: Gazebo [SBT1] (http://gazebosim.org/), ULISES framework [SBT2]

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental – Simulation, Experimental - Testing
- Type of component under evaluation: Software, Model
- Evaluation tool: Open source, Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Thinking
- Type of requirements under evaluation: Non-functional - Safety
- Evaluation performance indicator: V&V process criteria, SCP Criteria

Use case scenarios

- VALU3S_WP1_Industrial_15 – Worker position/action monitoring
- VALU3S_WP1_Industrial_16 – Recognition of workers' voice commands
- VALU3S_WP1_Industrial_17 – Responses to external control devices
- VALU3S_WP1_Industrial_18 – AI capabilities to work in the system

Strengths

- In simulation-based testing, verification activities can be done for different robots without changing other models or tools. Also, system-testing can be done without producing any physical item and adding risk to environment.
- Constraint-based modelling of oracles can provide powerful asserts in complex simulation testing.

Limitations

- The generation of Constraint-based knowledge model is a complex and time-consuming task.
- Simulation tools that used constraint-based modelling for assertion can require much computational power and limits real-time applications.

References

- [SBT1] Koenig, N., & Howard, A. (2004, September). Design and use paradigms for gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566) (Vol. 3, pp. 2149-2154). IEEE.
- [SBT2] Aguirre, A., Lozano-Rodero, A., Matey, L. M., Villamañe, M., & Ferrero, B. (2014). A novel approach to diagnosing motor skills. IEEE Transactions on Learning Technologies, 7(4), 304-318.
- [SBT3] Aguirre, A., Lozano-Rodero, A., Villamañe, M., Ferrero, B., & Matey, L. M. (2012). OLYMPUS: An Intelligent Interactive Learning Platform for Procedural Tasks. In GRAPP/IVAPP (pp. 543-550).

• [SBT4] Ostiategui, F., Amundarain, A., Lozano, A., & Matey, L. (2010). Gardening Work Simulation Tool in Virtual Reality for Disabled People Tutorial. Proceedings of Integrated Design and Manufacturing-Virtual Concept (IDMME'10). • [SBT5] Webster, M., Western, D., Araiza-Illan, D., Dixon, C., Eder, K., Fisher, M., & Pipe, A. G. (2020). A corroborative approach to verification and validation of human–robot teams. The International Journal of Robotics Research, 39(1), 73-99. • [SBT6] Takaya, K., Asai, T., Kroumov, V., & Smarandache, F. (2016, October). Simulation environment for mobile robots testing using ROS and Gazebo. In 2016 20th International Conference on System Theory, Control and Computing (ICSTCC) (pp. 96-101). IEEE.
Related standards: ISO 10218-1: 2011, ISO 10218-2: 2011, ISO/TS 15066: 2016
Keywords: Simulation-based testing, Human-robot collaboration, Constraint-based modelling, Knowledge modelling, Test oracles, Artificial intelligence.

3.2.3 Test Optimization for Simulation-Based Testing of Automated Systems

Name: **Test optimization for simulation-based testing of automated systems**
Purpose: To minimize the resources invested while maximizing the number of scenarios and situations that are tested.
Description: Simulation-based testing has been envisioned as an efficient means to test Automated Systems. Employing simulation models permits 1) the execution of more and larger test cases, 2) selection of critical scenarios, 3) specification of test oracles for the automated validation of the system, or 4) replication of safety-critical functions of a real system where it is expensive to execute test cases [TOS2]. However, although the use of simulation methods provides several advantages, testing Automated Systems is still expensive and time-consuming. Simulation models of Automated System could be very complex, and executing the simulations becomes computationally expensive, which often make it infeasible to execute all the test cases. For this reason, test optimization plays a crucial role when testing automated systems. The objective of test optimization is to cost-effectively test a system, i.e., reduce the cost of testing a system while the overall test quality is maintained. Test optimization could include test case selection, test case minimization, test case prioritization, etc. • Test case selection focuses on selecting a set of test cases from the test suite that tests a specific system version. • Test minimization aims to eliminate redundant test cases from the existing test suite in order to reduce cost (i.e., reduce the test execution time). • Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal. Test optimization could be also obtained using automatic test case generation: the process of generating test suites for a particular system. Since these approaches are typically non-trivial, often search-based algorithms are employed. *Test case generation*: [TOS4] and [TOS2] present an approach for the generation of the optimal set of reactive test cases for simulation-based testing of Cyber-Physical Systems using search-based

algorithms. The optimization is made taking into account the following cost-effectiveness measures: requirements coverage, test case similarity and test execution time.

Test case selection: [TOS3] and [TOS5] present a cost-effective approach for test case selection that relies on black-box data related to inputs and outputs of the system.

Test case prioritization: [TOS1] presents a search-based approach that aims to cost-effectively optimize the test process by prioritizing the test cases that are executed at different test levels (i.e., MiL, SiL and HiL).

Relationship with other methods: This method could be used to optimize the test cases and test suites used in other simulation-based testing methods such as *Simulation-based robot verification*, *Simulation-based testing for human-robot collaboration*, *V&V of Machine Learning-Based Systems Using Simulators*, etc.

Tool support: Matlab and Simulink (https://www.mathworks.com/products/matlab.html). Current implementation of the algorithms is done in Matlab and simulation-based testing applied in Simulink. Adaptation is required to adapt to other simulators.

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation Type: Experimental - Simulation
- Type of component under evaluation: Software, Model
- Evaluation tool: Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Acting, Sensing and Thinking
- Type of requirements under evaluation: Functional, Non-Functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP Criteria

Use case scenarios

- VALU3S_WP1_Industrial_15 – Worker position/action monitoring
- VALU3S_WP1_Industrial_16 – Recognition of workers' voice commands
- VALU3S_WP1_Industrial_17 – Responses to external control devices
- VALU3S_WP1_Industrial_18 – AI capabilities to work in the system

Strengths

- Test optimization can help to reduce the time and cost needed for testing
- This method could be used in combination with simulation-based testing methods

Limitations

- More empirical evaluation is required for some of the approaches
- Some approaches require historical data
- Matlab and Simulink used for simulation

References

- [TOS1] A. Arrieta, S. Wang, G. Sagardui, L. Etxeberria. "Search-Based Test Case Prioritization for Simulation-Based Testing of Cyber-Physical System Product Lines" in Journal of Systems and Software. Volume 149, 2019, Pages 1-34, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2018.09.055.
- [TOS2] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, L. Etxeberria. "Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-

Physical Systems" in IEEE Transactions on Industrial Informatics, vol. 14, no. 3, pp. 1055-1066, March 2018, doi: 10.1109/TII.2017.2788019. • [TOS3] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, Goiuria Sagardui, Pareto efficient multi-objective black-box test case selection for simulation-based testing, Information and Software Technology, Volume 114, 2019, Pages 137-154, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2019.06.009. • [TOS4] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui and L. Etxeberria, "Search-based test case generation for Cyber-Physical Systems," 2017 IEEE Congress on Evolutionary Computation (CEC), San Sebastian, 2017, pp. 688-697, doi: 10.1109/CEC.2017.7969377. • [TOS5] Arrieta, A., Shuai Wang, Ainhoa Arruabarrena, Urtzi Markiegi, G. Mendieta and L. E. Elorza. "Multi-objective black-box test case selection for cost-effectively testing simulation models." Proceedings of the Genetic and Evolutionary Computation Conference (2018).
Related standards: -
Keywords: Test-optimization, Simulation-based testing, Automated systems.

3.2.4 Virtual Architecture Development and Simulated Evaluation of Software Concepts

Name: **Virtual architecture development and simulated evaluation of software concepts**
Purpose: Efficient and reliable prototyping of complex systems involving cross-domain aspects by integrating heterogeneous components within holistic testing scenarios subject to goal-specific model fidelity and by systematically evaluating properties of interest in self-contained virtual runtime environments.
Description: With cyber-physical systems – communicating embedded systems that can autonomously adapt to their environment and learn new tasks – the challenges for system developers are growing because the CPS must evolve during runtime. This leads to completely new challenges regarding the architecture of these systems. As there is a lack of experience, the only way to evaluate new architecture concepts is the development of real prototypes. To select the best concepts, however, a great number of prototypes have to be realized. This causes very high costs and long development cycles. The simulation and virtual validation framework FERAL (Framework for fast Evaluation on Requirements and Architecture Level) is a solution for virtual architecture development and simulated evaluation of software concepts. FERAL allows the design of virtual prototypes, which replace real prototypes by means of simulation and evaluate the impact of new architecture concepts in a cost-efficient manner. The solution enables simulation-based validation and the systematic assessment of design decisions at an early stage by coupling simulation models and simulators, existing code, and virtual hardware platforms. Simulation may be based purely on models; this is called Model-in-the-Loop (MiL) simulation. For that purpose, FERAL supports e.g. UML state machines, activity diagrams, and coupling with other simulators such as Matlab Simulink. FERAL supports this step by providing virtual hardware platforms, i.e., processor and network models to which the software components can be deployed in a virtual Hardware-in-the Loop (vHiL) simulation. All these simulations (MiL and vHiL) serve to detect defects in early development phases, which allows reducing the number of expensive

Hardware-in-the-Loop-(HiL) simulations and integration tests. FERAL is very suitable for the creation of complex, holistic test scenarios with inputs coming from different levels. It can, for example, process information from Simulink, Amalthea models, source code, architectural specifications, etc. With this technology, the planned contribution is twofold. One the one hand, IESE aims to use FERAL and support the generation of complex test scenarios for dedicated quality properties such as robustness and performance. On the other hand, by driving this simulation, it is possible to execute generated tests and follow the propagation of data of interest through the system.

Relationship with other methods: *Model-based testing, Model-based robustness testing, Simulation-based verification.*

Tool support: FERAL (proprietary license; https://www.iese.fraunhofer.de/en/services/virtual-architecture-development-and-evaluation.html)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental – Simulation
- Type of component under evaluation: Software, Model
- Evaluation tool: Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-functional - Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_13 - Corruption of input/output signal at robot gripper.
- VALU3S_WP1_Industrial_14 - Data manipulation in human-robot-interaction
- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths

- Enables early verification of the appropriateness of design decisions by providing executable simulation scenarios
- Provides technical solutions for coupling heterogeneous system parts (i.e. different implementation formats and maturity levels) and communication protocols
- Reuses and connects existing simulation tools

Limitations

- Initial abstraction level-dependent efforts for creating simulation scenarios and simulation components
- Trade-off between accuracy and effort for finding an appropriate simulation model quality

References

- [VAD1] T. Kuhn, T. Forster, T. Braun, R. Gotzhein: Feral - framework for simulator coupling on requirements and architecture level. ACM/IEEE MEMOCODE, pp. 11–22 (2013)
- [VAD2] P. O. Antonino, J. Jahic, B. Kallweit, A. Morgenstern, and T. Kuhn: Bridging the Gap between Architecture Specifications and Simulation Models. IEEE International Conference on

Software Architecture Companion, Seattle, WA, USA, pp. 77-80 (2018), DOI: 10.1109/ICSA-C.2018.00029.

- [VAD3] A. Bachorek, F. Schulte-Langforth, A. Witton, T. Kuhn, P. Oliveira Antonino: Towards a Virtual Continuous Integration Platform for Advanced Driving Assistance Systems. IEEE International Conference on Software Architecture Companion, pp. 61-64 (2019), DOI: 10.1109/ICSA-C.2019.00018
- [VAD4] T. Kuhn, P. O. Antonino, A. Bachorek: A Simulator Coupling Architecture for the Creation of Digital Twins. IEEE International Conference on Software Architecture Companion, pp. 326-339 (2020), DOI: 10.1007/978-3-030-59155-7_25

Related standards: ISO 26262, IEC 61508

Keywords: Virtual validation, Design verification, System simulation.

3.2.5 Virtual & Augmented Reality-Based User Interaction V&V and Technology Acceptance

Name: **Virtual & augmented reality-based user interaction V&V and technology acceptance**

Purpose: Human factors analysis and technology acceptance by end users using virtual and/or augmented reality technologies before the system is fully deployed in its industrial/home environment.

Description: Over the last decade, there has been an ever-growing interest in human-robot interaction (HRI), not only in traditional industrial fields but also in emerging areas such as homes. Therefore, it seems vital to understand how robots are perceived and understood by humans to be fully accepted. The manner people accept collaborative robots in their life is still unknown, and it is an essential aspect to overcome the resistance towards them.

The concept of trust is very important in the adoption of technologies to assist people. Trust can be defined as an attitudinal judgement of the degree to which a user can rely on an agent (the collaborative robot) to achieve his/her goals under conditions of uncertainty. Moreover, safety and efficiency of HRI collaboration often depend on appropriately calibrating trust towards the robot and using a user-centred approach to realise what impacts the development of trust. The evaluation of whether the person feels safe and comfortable with the proposed solution requires advanced physical prototyping or, as an alternative, virtual reality (VR) as a simulation tool allows for fast, flexible, and iterative testing processes.

In order to evaluate the users' sense of safety and comfort, surveys can be conducted so that the participants evaluate different parameters. This has been applied, e.g., for an unmanned aerial vehicle's (UAV) trajectory during the monitoring process in a VR environment [VUR1]. The study focusses on analysing three key parameters of the monitoring process; (i) the relative flight height, (ii) the speed of the UAV during the lap to the person, and finally, (iii) the shape of the trajectory that the UAV follows around him/her, considering two main options, namely, a circular path, which leads to maintain a constant distance between the person and the UAV, and an ellipsoidal one where the distance changes along the way.

The technological approach followed to achieve the HRI interaction involving end-users is described in [VUR1] and relies on a distributed architecture with two main modules (see Figure 3.3): the UAV Simulator, in charge of reproducing the flight of the UAV considering its dynamics, and generating the trajectories; and the VR Visualiser, in charge of rendering the virtual UAV and its behaviour, as well as the virtual environment in which the UAV flight takes place. These two modules communicate with each other using the MQTT protocol.

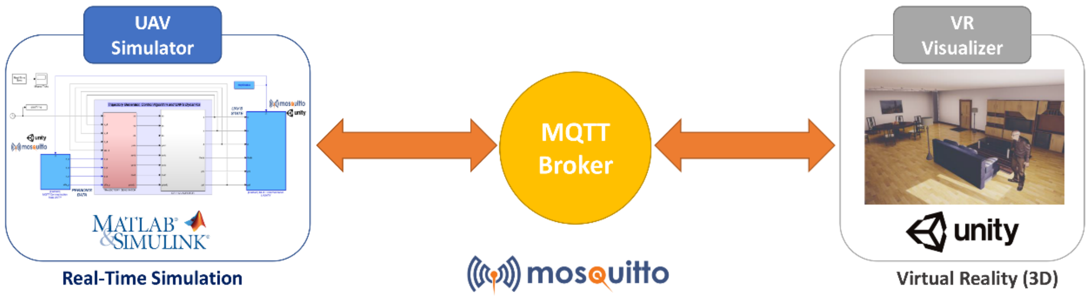


Figure 3.3 Architecture of the distributed platform [VUR1]

Relationship with other methods: This method relates to all the other methods that include simulated interaction/collaboration of an end-user with an automated system.

Tool support: In-house development using Unity3D (https://unity3d.com/), MATLAB (www.mathworks.com), and MQTT (https://mqtt.org/).

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental - Simulation
- Type of component under evaluation: Software, Model
- Evaluation tool: Open source, Proprietary
- Evaluation stage: Validation
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional – Safety, Non-functional - Others
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_3 - Safety trajectory optimization
- VALU3S_WP1_Industrial_10 - Localization of human
- VALU3S_WP1_Industrial_14 - Data manipulation in human-robot interaction

Strengths

- Early testing before the physical robot solution is deployed
- Allows obtaining collaboration experience without using the real robot in a safe environment and thus reducing the risks

Limitations

- The differences in the interaction/collaboration between simulation and reality
- The cost of building the VR/AR simulator

References

- [VUR1] Belmonte, L.; Garcia, A.S.; Segura, E.; Novais, P.J.; Morales, R.; Fernandez-Caballero, A. Virtual Reality Simulation of a Quadrotor to Monitor Dependent People at Home. IEEE Transactions on Emerging Topics in Computing, 2020. doi:10.1109/TETC.2020.30003

Related standards: ISO/TS 15066:2016

Keywords: HRI V&V, Technology acceptance, Early testing, Feeling of safety and comfort.

3.2.6 V&V of Machine Learning-Based Systems Using Simulators

Name: **V&V of machine learning-based systems using simulators**
Purpose: Efficient and effective V&V of SCP requirements in simulated environments without endangering human safety.
Description: Machine learning, in particular deep learning, is a critical enabling technology for many of the highly automated applications today. Typical examples include intelligent transport systems (ITS) where ML solutions are used to extract a digital representation of the traffic context from the highly dimensional sensor inputs. Unfortunately, the ML models are opaque in nature (stochastic and data driven with limited output interpretability), while functional safety requirements are strict and require a corresponding safety case [VVM1]. Furthermore, development of systems that rely on deep learning introduces new types of faults [VVM2]. To meet the increasing needs of trusted ML-based solutions [VVM3], numerous V&V approaches have been proposed. Simulators can be used to support system testing as part of V&V of SCP requirements. An ideal simulator to test perception, planning and decision-making components of an autonomous system must realistically simulate the environment, sensors and their interaction with the environment through actuators. Simulated environments bring several benefits to V&V of ML-based systems, particularly when: • Collection of live and interactive data is not possible • Data collection or data annotation is difficult, costly or time consuming • Real-world testing is endangering human safety • Coverage of collected data is limited • Reproducibility and scalability are important The major bulk of system-level testing of autonomous features in the automotive industry is carried out through on-road testing or using naturalistic field operational tests. These activities, however, are expensive, dangerous, and ineffective [VVM4]. A feasible and efficient alternative is to conduct system-level testing through computer simulations that can capture the entire self-driving vehicle and its operational environment using effective and high-fidelity physics-based simulators. There is a growing number of public-domain and commercial simulators that have been developed over the past few years to support realistic simulation of self-driving systems, e.g., TASS/Siemens PreScan, ESI Pro-SiVIC, CARLA, LGSVL, SUMO, AirSim, and BeamNG. Simulators will play an important role in the future of automotive V&V, as simulation is recognized as one of the main techniques in ISO/PAS 21448. As the possible input space when testing automotive systems is practically infinite, attempts to design test cases for comprehensive testing over the space of all possible simulation scenarios are futile. Hence, search-based software testing has been advocated as an effective and efficient strategy to generate test scenarios in simulators [VVM5, VVM6]. Another line of research proposes techniques to generate test oracles, i.e., mechanisms for determining whether a test case has passed or failed [VVM7]. Related to the oracle problem, several authors proposed using metamorphic testing of ML-based perception systems [VVM8, VVM9], i.e., executing transformed test cases while expecting the same output. Such transformations are suitable to test in simulated environments, e.g., applying filters on camera input or modifying images using generative adversarial networks.

Relationship with other methods: The approach to perform V&V, and especially testing, of ML-based systems is related to the other simulation methods. For example, robotic systems can rely onboth supervised and reinforcement learning (*Simulation-based robot verification*) and the test optimization method (*Test optimization for simulation-based testing of automated systems*) can be used to prune the set of test cases generated by search-based and metamorphic testing. Furthermore, the proposed simulation method can be combined with the methods listed under *Testing* and *Injection-based V&V*. Particularly, *Machine learning model validation* and *Simulation-based fault injection at system-level* are closely related. Moreover, many aspects that depend on an executing SUT can be conducted also on a simulated counterpart. Thus, methods such as *Software-implemented fault injection*, *Model-based robustness testing*, *Model-based testing*, and *Risk-based testing* are also related to this method.

Tool support: Simulators such as TASS/Siemens PreScan (https://tass.plm.automation.siemens.com/prescan), ESI Pro-SiVIC (https://www.esi-group.com), LGSVL (https://www.lgsvlsimulator.com), SUMO (https://www.eclipse.org/sumo/), CARLA (http://www.carla.org)https://carla.org/, AirSim (https://github.com/microsoft/AirSim), and BeamNG (https://www.beamng.com) are available. In house developed simulator based on Unreal Engine (https://www.unrealengine.com) can also be used for test case generation in form of simulated scenes and scenarios.

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental – Simulation, Analytical - Formal
- Type of component under evaluation: Software, Model
- Evaluation tool: Open source, Proprietary
- Evaluation stage: Validation, Verification
- Logic of the component under evaluation: Thinking
- Type of requirements under evaluation: Non-Functional - Safety, Non-Functional - Cybersecurity
- Evaluation performance indicator: -

Use case scenarios

- VALU3S_WP1_Automotive_1 - Radar/camera advanced detection and tracking
- VALU3S_WP1_Automotive_2 - Radar + camera cooperation
- VALU3S_WP1_Automotive_3 - Node connection to cloud

Strengths

- Cost efficient: Using simulation for V&V of automotive systems reduces the cost of using a real track and actual vehicles and instruments that could risk damage during the testing process.
- Time: Having an immediate response from a simulator shortens the software development cycle, i.e., it enables quicker feedback.
- Safety: Currently, testing many vehicle collisions and accident scenarios are done using safe dedicated test and assessment protocols, however, testing an incomplete system always exposes the testers to unpredictable dangers. Using simulators, the risks of test driving of an autonomous vehicle in urban areas will be substantially reduced.
- Edge cases: Many low probability safety critical situations and hazards that would not be encountered on a test track can be generated in simulated environments.

Limitations

- The gap between simulation and reality

• The cost of building the digital twin for sensors
References • [VVM1] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Levandowski, S. Gao, Y. Tan, H. Kaijser, H. Lönn, and J. Törnqvist. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. Journal of Automotive Software Engineering, 1(1), pp. 1-19, 2018. • [VVM2] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, A., and P. Tonella, P. Taxonomy of real faults in deep learning systems. In Proc. of the ACM/IEEE 42nd Int'l. Conference on Software Engineering, pp. 1110-1121, 2020. • [VVM3] Assessment List for Trustworthy AI, High-Level Expert Group on AI (AI HLEG), European Commission, https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=68342 • [VVM4] Koopman, P. and Wagner, M., 2016. Challenges in autonomous vehicle testing and • validation. *SAE International Journal of Transportation Safety*, *4*(1), pp.15-24. • [VVM5] Abdessalem, R.B., Nejati, S., Briand, L.C. and Stifter, T., 2018, May. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (pp. 1016-1026). IEEE. • [VVM6] Gambi, A., Mueller, M. and Fraser, G., 2019, July. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 318-328). • [VVM7] Stocco, A., Weiss, M., Calzana, M. and Tonella, P., 2020, June. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (pp. 359-371). • [VVM8] Tian, Y., Pei, K., Jana, S. and Ray, B., 2018, May. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering* (pp. 303-314). • [VVM9] Zhang, M., Zhang, Y., Zhang, L., Liu, C., & Khurshid, S. (2018). DeepRoad: Gan-based metamorphic autonomous driving system testing. *arXiv preprint arXiv:1802.02295*.
Related standards: ISO 26262, ISO/PAS 21448, UL4600, ASAM OpenDrive, ASAM OpenScenario
Keywords: Simulation-based testing, Artificial intelligence, Machine learning, Digital twin, Synthetic data generation.

3.3 Testing

This group of methods focuses on validating a system by its execution in the frame of so-called test cases. At least, a test case contains two fundamental sets of information: input data to be provided to the SUT, and a description of the expected output or behaviour. In order to perform a test case, an environment is used that allows to feed the SUT with the input data in a controlled manner, as well as to monitor its reactions. This environment is sometimes called test harness. Further, usually a means is needed to judge whether the SUT's reactions conform to expectations. Such means is sometimes referred to as test oracle. For testing, the SUT can be the final system as well as any artefact used in its development; i.e. models or specific hardware or software components.

It is distinguished between black box testing – where only the interfaces of the SUT are considered, and its interior considered as black box; white box testing – where also the SUT's interior, e.g. inner states, is monitored; and combinations of both, i.e. grey box testing. The scope of testing can be functional, i.e. assessing whether the SUT behaves as expected (fulfils its functions), and non-functional, i.e. assessing its performance, robustness, security etc. Therefore, testing can contribute significantly to establish SCP. However, it should be considered that testing is usually incomplete, i.e. even successful passing a large set of test cases (a test suite) is no guarantee for the SUT's correctness. A test suite's quality is correlated with two aspects: how good it covers the addressed issues (functionality, robustness, …), and how efficiently it achieves this.

A technique to get high quality test cases is (automated) test case generation, which is used by most of the methods described in this clause. Further, various coverage criteria are addressed, e.g. scenarios, potential implementation faults, or potential impact of cybersecurity attacks on safety. Many of the V&V-methods address testing of non-functional issues such as safety, robustness, cyber-security, but also with novel properties of automated systems, e.g. machine learning. Some reviewed methods can also be used for functional testing. All types of components are considered, with a slight emphasis on models, in order to detect conceptual flaws as early as possible. Finally, black-box and white-box testing are supported.

3.3.1 Behaviour-Driven Model Development and Test-Driven Model Review

Name: **Behaviour-driven model development and test-driven model review**
Purpose: Develop high-quality behaviour models in UML that are fit for test and code generation. High-quality functional tests are a side-product. The model can also be used to generate non-functional tests.
Description: Behaviour-Driven Model Development and Test-Driven Model Review combines Behaviour-Driven Development [BHM1] concepts, Model-Driven Development and Model-Based Testing in the following steps: 1. Define the test interface as methods and signal receptions in UML class diagrams 2. Express scenarios from the requirements as UML sequence diagrams 3. Model the required behaviour as UML state machine diagrams • While doing so, continually run the scenarios from the sequence diagrams against the state machines. The modeler gets feedback which scenarios do not run through yet and where they get stuck. 4. Validate, that the scenarios work on the finished state machine models 5. Evaluate coverage of the scenarios on the state machines 6. Generate a minimal set of tests to complete the coverage 7. Review the generated tests – they represent all behaviour that might have been unintentionally added while modelling the state machines. • Go back and change the model and the initial scenarios, if necessary and repeat. 8. Use the model to generate code 9. Run the original and the generated scenarios against • the generated code (to exclude problems introduced by the code generator)

- the target compiled code (to exclude problems introduced by the compiler or the platform)
- the deployed binary in the target environment (to exclude problems introduced by the environment and production periphery)

The approach has been described in more detail in [BHM2], it is similar to Behaviour-Driven Formal Model Development [BHM3].

Relationship with other methods: The approach is similar to *Behaviour-driven formal model development* and makes use of *Model-based testing* and can use *Model-based mutation testing*.

Tool support: Toolchain of Enterprise Architect (https://www.sparxsystems.com/products/ea/index.html), Method-specific plug-in to Enterprise Architect (pre-release), Embedded Engineer (https://www.lieberlieber.com/embedded-engineer/) , and MoMuT::UML(https://momut.org/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab, Closed (use of resulting scenarios)
- Evaluation type: Experimental – Testing, Experimental – Monitoring, Analytical – Semi-Formal
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Thinking, Sensing, Acting
- Type of requirements under evaluation: Functional
- Evaluation performance indicator: V&V process criteria

Use case scenarios

- VALU3S_WP1_Automotive_3 - Node connection to cloud
- VALU3S_WP1_Industrial_5 - Motor speed control
- VALU3S_WP1_Industrial_6 - Fault tolerance for motor position sensor data
- VALU3S_WP1_Industrial_7 - Safety behaviour for missing motor position sensor data
- VALU3S_WP1_Industrial_8 - Safety behaviour for remote control terminal connection failure
- VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal
- VALU3S_WP1_Railway_1 - Inject, detect and recover
- VALU3S_WP1_Railway_2 - Controlled vs random injection
- VALU3S_WP1_Railway_3 - Systematic and random failures verification

Strengths

- Produces high-quality behaviour models
- Enables domain experts to review the model indirectly by reviewing scenarios

Limitations

- Only limited tool support available yet

References

- [BHM1] C. Solis and X. Wang, "A study of the characteristics of behaviour-driven development," in2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 383–387.

• [BHM2] Schlick R., Krenn W. (2019) Tackling the Challenges of Internet-of-Things-Development using Models. 2ndInternational Workshop on Embedded Software for the Industrial IOT, DATE, Florence, 2019. http://adt.cs.upb.de/ESIIT2019-tproceedings.pdf • [BHM3] Snook C. et al. (2018) Behaviour-Driven Formal Model Development. In: Sun J., Sun M. (eds) Formal Methods and Software Engineering. ICFEM 2018. Lecture Notes in Computer Science, vol 11232. Springer, Cham. https://doi.org/10.1007/978-3-030-02450-5_2
Related standards: -
Keywords: Behaviour-driven development, Model-based testing, Functional testing, Modelling support.

3.3.2 Assessment of Cybersecurity-Informed Safety

Name: **Assessment of cybersecurity-informed safety**
Purpose: Black-box testing for security-informed safety of automated driving systems. To support black box testing as part of an independent evaluation, with the aim of producing an understanding of the interplay between safety and security, enabling a comparison of how well different ADSs can withstand safety-relevant security threats.
Description: Automated vehicles will soon be a reality. But to be safely released to the market it must be shown that cybersecurity threats do not jeopardize safety. As it is virtually impossible to validate an automated vehicle against all possible scenarios it will face in the real world, least of all in combination with security threats, there is a need to have a balance between the representativeness of the tests and the reliable performance indicators. One way is to define testing and validation procedures of ADS features that can combine tests both in simulation and in real environments, such as test tracks, into a complete assessment. This method could be used to establish independent tests with the purpose of evaluating and comparing the ability of ADSs to withstand security threats that can affect their safe operation. The method (Figure 3.4) consists of: A. Extracting enough information from the feature description to facilitate a categorization of feature classes for an ADS under test. With feature class, we mean generic descriptions of ADS features that match functionality offered by several vendors. B. Development of an appropriate test suite for test facilities to assess cybersecurity, matching the feature class and sensor setup. C. Co-simulation of post-attack behaviour with critical traffic scenarios to evaluate safety criteria. D. Evaluating coverage and update the test suite if needed and assess if the safety criterion holds. The methodology generates tests that has been shown to be relevant to the establishment of a baseline for cybersecurity of an ADS. The proposed process is somewhat different from a more traditional approach where the combinatorial explosion renders test coverage unfeasible when considering all types of attacks and traffic scenarios. In contrast this approach is not to provide complete coverage, but rather aims at building confidence. The Co-simulation approach identifies the critical scenarios that needs to be tested and makes the risks in the validation testing predictable, thus enabling the use of a proactive strategy in addressing the hazards. The identified critical scenarios, comprised of relevant attacks in representative traffic conditions, may well be orchestrated and evaluated to form a comparable independent assessment of cybersecurity.

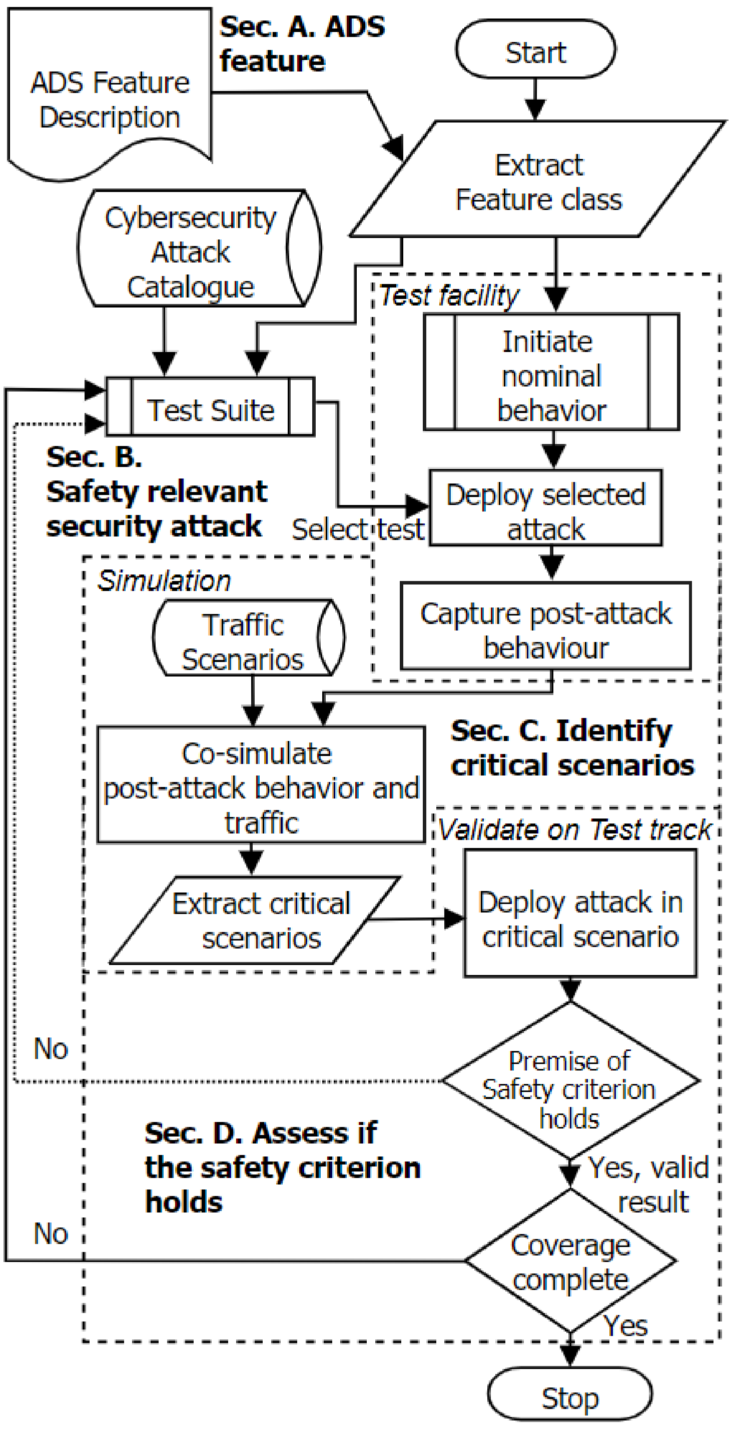

ADS Feature Description
Sec. A. ADS feature
Start
Extract Feature class
Cybersecurity Attack Catalogue
Test facility
Initiate nominal behavior
Test Suite
Sec. B. Safety relevant security attack
Select test
Deploy selected attack
Simulation
Traffic Scenarios
Capture post-attack behaviour
Co-simulate post-attack behavior and traffic
Sec. C. Identify critical scenarios
Validate on Test track
Extract critical scenarios
Deploy attack in critical scenario
No
Premise of Safety criterion holds
Sec. D. Assess if the safety criterion holds
Yes, valid result
No
Coverage complete
Yes
Stop

Figure 3.4 Methodology of tests that could support establishing cybersecurity-informed safety in an ADS

Relationship with other methods: Relationship with all system-related methods including *Simulation*.

Tool support: Simulation scenario tool and investigate the use of Carla as scenario simulator (https://carla.org)

Layers of the multi-dimensional framework

- Evaluation environment: Closed, In the lab
- Evaluation type: Experimental - Simulation, Experimental - Testing
- Type of component under evaluation: Software, Hardware

• Evaluation tool: Open source • Evaluation stage: Validation • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Non-functional - Safety, Non-functional - Cybersecurity • Evaluation performance indicator: V&V process criteria, SCP criteria
Use case scenarios • VALU3S_WP1_Automotive_4 - Transmission line under different performance conditions • VALU3S_WP1_Automotive_7 - Safety of vehicle during switch between routers • VALU3S_WP1_Agriculture_3 - Transmission line disturbances
Strengths • Will provide evidence to establish increased confidence that the ADS will not violate the safe operating conditions even during or after a cybersecurity attack • A safe and controlled testing process for ADS cybersecurity where dangers involved in performing real environment security attacks are mitigated • The co-simulation of post-attack behaviour and traffic to identify critical scenario will reduce the assessment test volume
Limitations • Simulation validity requires further investigation • Achieve a representativeness of the tests as reliable performance indicators • Investigate capture post-attack behaviour in a way so that it is useful for co-simulation
Reference • [ACS1] HEADSTART project: https://www.headstart-project.eu/
Related standards: ISO-TC22-SC32-WG11_N0613_ISO_SAE_DIS_21434_(E), SS-ISO_26262_2018, SAE J3061, UL4600
Keywords: Assessment, Cybersecurity-informed safety, Exploit vulnerabilities.

3.3.3 Machine Learning Model Validation

Name: **Machine learning model validation**
Purpose: Model validation in machine learning automated systems serves to evaluate how a system performs and how safe it is when applied to input data other than the data used to train it.
Description: Machine learning (ML) algorithms have been becoming pervasive in many technological fields, and they are both surpassing some traditional techniques as well as offering breaking solutions to previously intractable problems. A key element in ML is that the models learn from training data rather than being explicitly designed and tuned by engineers. This same key element poses high challenges in verifying and validating a trained model with regards to its effectiveness and safety. On one hand, models have generic structures which represent no physical processes/laws, and on the other hand, the data used to train and validate the models must rely on human effort to manually and informally curate the input data and define the expected output (supervised learning), rendering many of the classical V&V methods unsuitable. The usual supervised training process relies on large amounts of data that includes system inputs and respective output and optimizes the parameters in order to fit the model and minimize a “loss”

metric. But fitting the model to training data does not validate that the model will perform as desired outside that training domain. For that purpose, "testing data" that is never used to train the model is used to validate the model generalization to expected scenarios. Besides that, "validation data" can also be used to guide the model high-level architecture design. Hence three main categories of datasets are used [MLV1, MLV2]:

- Training datasets: data used to train and optimize the ML model parameters;
- Validation dataset: sample of data used to evaluate the trained model and help the design of the ML model structure (e.g., number of neural network layers, training procedures, etc.), the so-called hyper parameters. These datasets also introduce bias to the model as the model is tailored to favour them;
  Note: the term "validation" used in here is distinct to the term used in the context of V&V
- Test dataset: the sample of data used to estimate the real, unbiased performance of the model when applied to novel input.

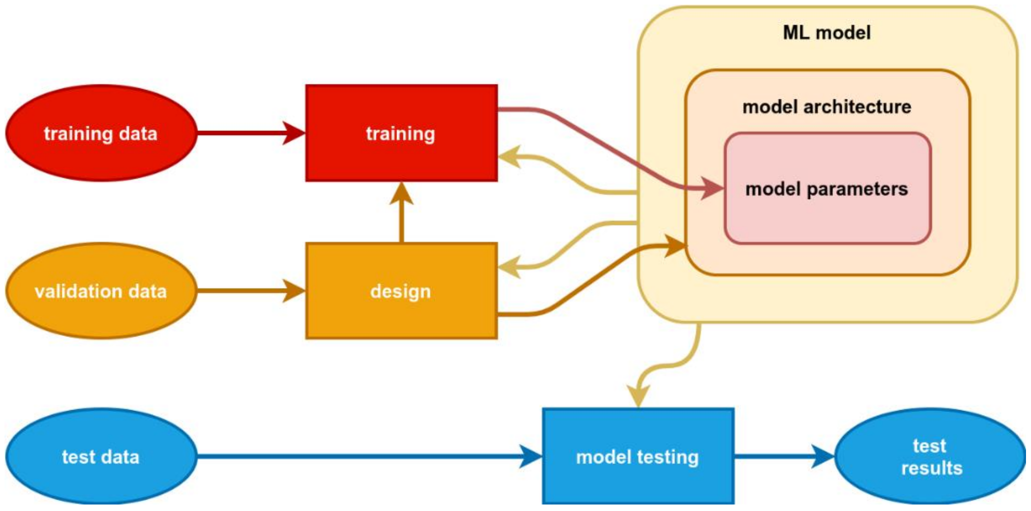


Figure 3.5 Datasets usage in ML model design, training and testing phases

Model validation relying on the provision of multiple datasets can then be tailored with multiple variants of the validation technique, including holdout, cross-validation, random subsampling, and bootstrapping.

Due to being a novel technology, the regulatory requirements for the V&V of ML systems are still in the phase of drafting and designing by the standards organizations and the regulatory authorities, for example, in the healthcare domain [MLV3, MLV4].

Relationship with other methods: Some variations on the verification and validation methods for ML have relations to other methods:

- Automatic test-case generation, i.e., the use of simulated/synthetic data in the validation, and adaptive stress testing, i.e., the adaptation of simulated data to find failure events. This relates to the method *Verification and validation of machine learning based systems using simulators*
- Formal testing for ML models, i.e., the use of those classical methods by formally defining specifications and demonstration of correctness guarantees. This relates to *Model Checking*, and *Formal Methods*.

- Analysis of adversarial robustness, i.e., analysis of how stable or sensitive is the model output with regards to small adversarial changes in the input. This relates to the method *Verification and validation of machine learning based systems using simulators* and to *Model Checking*.
- Runtime monitoring of deployed ML systems.

Tool support: ML frameworks such as Tensorflow (tensorflow.org), Pytorch (pytorch.org), and Flux (fluxml.ai), and Programming languages such as Python (python.org), Julia (julialang.org), and R (r-project.org).

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental – Testing
- Type of component under evaluation: Model
- Evaluation tool: Open Source
- Evaluation stage: Validation
- Logic of the component under evaluation: Thinking
- Type of requirements under evaluation: Functional, Non-Functional - Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Healthcare_1 - Bone segmentation baseline performance
- VALU3S_WP1_Healthcare_8 - Generalization of bone segmentation to typical conditions
- VALU3S_WP1_Healthcare_9 - Robustness of bone segmentation to challenging conditions

Strengths

- Does not require interpretability of the models, i.e., it allows black-box models (although it can be used in interpretable models too)
- The data and the procedures used to create the model have the same nature of the data and procedures used to validate it
- Enables assimilation of validation data into the model in order to further enhance it, being especially important when finding input data where the model is failing

Limitations

- Usually requires manual effort to define expected output (especially in supervised machine learning)
- Does not offer formal guarantees of effectiveness or safety
- Cannot evaluate the model behaviour on challenging input not yet existent

References

- [MLV1] Buduma, N., and N., Locascio. Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms. O'Reilly Media, 2017
- [MLV2] Goodfellow, I., Y., Bengio, and A., Courville. Deep Learning. MIT Press, 2016
- [MLV3] Proposed Regulatory Framework for Modifications to Artificial Intelligence/Machine Learning (AI/ML)-Based Software as a Medical Device (SaMD). FDA, 2019
- [MLV4] Regulatory Guidelines for Software Medical Devices – A Life Cycle Approach. Health Sciences Authority - Government of Singapore, 2020 (section 8. Artificial Intelligence Medical Devices (AI-MD))

Related standards: ISO 13485 Medical devices — Quality management systems — Requirements for regulatory purposes, IEC 62304 Medical device software — Software life cycle processes, IEC 62366 Medical devices — Part 1: Application of usability engineering to medical devices, ISO 14971 Medical devices — Application of risk management to medical devices, IEC 82304 Health software — Part 1: General requirements for product safety, ISO/IEC TR 24028 Information technology — Artificial intelligence — Overview of trustworthiness in artificial intelligence
Under development/drafting: ISO/IEC DTR 24029 Artificial Intelligence (AI) — Assessment of the robustness of neural networks — Part 1: Overview, ISO/IEC CD 23894 Information Technology — Artificial Intelligence — Risk Management, ISO/IEC CD 23053 Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML), ISO/IEC WD TS 4213 Information technology — Artificial Intelligence — Assessment of machine learning classification performance, (proposal) ISO/IEC AWI 24029-2 Artificial Intelligence (AI) — Assessment of the robustness of neural networks — Part 2: Methodology for the use of formal methods
Other potentially related proposals and drafts listed in https://www.iso.org/committee/6794475/x/catalogue/

Keywords: Artificial intelligence, Machine learning, Model validation, Overfitting, Underfitting, Training dataset, Validation dataset, Test dataset.

3.3.4 Model-Based Mutation Testing

Name: Model-based mutation testing

Purpose: Derive high quality tests from behaviour models.

Description: Model-based mutation testing is a fault-based variant of Model-Based Testing, where the generated test cases are guaranteed to detect implementations of certain faulty versions of the specification. The idea here is to show that in implementing the system, the requirements were correctly understood and that the SUT is free of the faults that were injected into the specification. Faulty specifications are called mutants, hence the term model-based mutation testing.
The final output of the method is a high-quality test suite. Quality of test suites can be measured by different means. Mutation score [MMT1] is a test suite metric, quantifying the fault detection capability of a test suite. Formally, mutation score is defined as the number of killed mutants divided by the number of created mutants. Its benefit
is that it quantifies semantic quality features of implementations, i.e. absence of implementation faults, as opposed to syntactic quality features, such as transition, state, or branch coverage metrics. The idea of mutations for test quality analysis goes back to at least 1978 [MMT2].

Relationship with other methods: *Model-based testing* as a general concept is specialized in Model-based mutation testing

Tool support: MoMuT (https://momut.org/), Conformiq Designer (https://www.conformiq.com/2019/08/conformiq-designer-5-0-for-mutation-testing/)

Layers of the multi-dimensional framework

- Evaluation environment: In the Lab, Closed, Openl
- Evaluation type: Experimental - Testing
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Proprietary

- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional
- Evaluation performance indicator: V&V process criteria

Use case scenarios

- VALU3S_WP1_Automotive_3 - Node connection to cloud
- VALU3S_WP1_Industrial_5 - Motor speed control
- VALU3S_WP1_Industrial_6 - Fault tolerance for motor position sensor data
- VALU3S_WP1_Industrial_7 - Safety behaviour for missing motor position sensor data
- VALU3S_WP1_Industrial_8 - Safety behaviour for remote control terminal connection failure
- VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal
- VALU3S_WP1_Railway_1 - Inject, detect and recover
- VALU3S_WP1_Railway_2 - Controlled vs random injection
- VALU3S_WP1_Railway_3 - Systematic and random failures verification

Strengths

- The method can produce compact test suites (in relation to the complexity of the system under test) with high functional coverage.
- The use of mutation coverage to drive test case generation orients itself on faults and guarantees that tests are generated where faults propagate into an observable deviation of the system behaviour. When using control-flow coverage to drive test-case generation, this is usually not achieved.

Limitations

- Too large symbolic or concrete state spaces of the system under test can make the approach computationally infeasible. Usually, good partial results can be achieved nonetheless, since missing coverage can be manually analysed, and testing is by conception an incomplete approach.
- As a black box testing approach, the implementation might structure the behaviour differently than the specification model does. This can lead to additional potential faults in the implementation, that are not covered by the test suite. Model-Defactoring [MMT3] is a potential solution for this problem.

References

- [MMT1] Y. Jia and M. Harman, 'An Analysis and Survey of the Development of Mutation Testing', *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, Sep. 2011, doi: 10/dd8s2k.
- [MMT2] DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. IEEE Computer 11(4), 34–41 (1978)
- [MMT3] Schlick R., Herzner W., Jöbstl E. (2011) Fault-Based Generation of Test Cases from UML-Models – Approach and Some Experiences. In: Flammini F., Bologna S., Vittorini V. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2011. Lecture Notes in Computer Science, vol 6894. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24270-0_20

Related standards: -

Keywords: Fault-driven testing, Model-based testing, Mutation testing, Test-case generation.

3.3.5 Model-Based Robustness Testing

Name: Model-based robustness testing
Purpose: Use an abstracted behaviour model of a component or system to derive unexpected or slightly out of specification stimuli in order to check the robustness of the artefact under test.
Description: Robustness is defined by ANSI/IEEE as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions." [MRT1]. Some authors include the graceful handling of internal error conditions in their definition of robustness. For our purposes, we stay with the ANSI/IEEE definition with challenges to robustness originating outside of the system. The definition of what makes up valid inputs and normal environmental conditions can be considered as the precondition in a contract between the system and its environment. If the precondition is fulfilled, the system shall fulfil its obligations and operate as specified. In most settings, this contract is not or at least not completely formalised. Robustness testing thereby is the experimental evaluation in how far the system operates as specified or acceptably close to specification (i.e. gracefully degrades [MRT2]), if this precondition is violated. If the system implementation takes more and incorrect assumptions than included in the contract, this should be instead addressed by sufficiently complete functional tests and stress tests that stay within the explicitly defined operation conditions. Since there are usually infinite possibilities to violate the contract preconditions with unexpected inputs, it is a) impossible to be complete and a definition of test adequacy is needed to decide which tests to select and b) tests need to be generated automatically since manual test design is infeasible. The intentional and automated use of unspecified inputs is also called fuzzing [MRT3], especially when applied to security testing. Classic fuzzing originally uses randomized inputs in large test suites. To generate inputs outside of the nominal inputs, a machine-readable specification of the allowed inputs is useful. Depending on the application area and test goals, it might be possible to reduce the test suite size by limiting the inputs to be only slightly out of specification (SooS) and close to some valid input, or not. E.g. in context of secure implementation of communication protocols, an approach based on detailed input specifications called grammar-based fuzzing and several approaches to derive inputs are taken, including mutation, machine learning and evolutionary computing [MRT4]. Model-Based Robustness Testing takes a (semi-)formal description of the expected system behaviour (i.e. the contract, although not necessarily in form of pre-condition/assumption and obligation/ guarantee but implicitly in a behaviour description) and a fault model how the precondition part of the contract could be violated. From this, both inputs and test oracles that decide if robustness properties and functional requirements hold, can be derived, see e.g. [MRT5] In many systems, but especially in cyber-physical systems, the inner state of the system might affect how an unexpected input is treated. If a representation of the system behaviour is given as a (semi-) executable behaviour model, this can be used to drive the system under test into different states and fuzz the inputs there [MRT6]. A different approach of Model-Based Robustness Testing for image processing applications is described in [MRT7], where based on a definition of the input situations and possible image processing problems test images and possibly the related ground truth is generated.

Relationship with other methods: *Model-based testing* can be used as basis to achieve coverage of the model. *Virtual Architecture Development and Simulated Evaluation of Software Concepts* can be used to apply the robustness tests before the complete system is available and assembled. *Interface fault injection* is a similar approach but uses only the interface definition.
Tool support: MoMuT (https://momut.org), Vitro (https://vitro-testing.com/vitro2/technology/)
Layers of the multi-dimensional framework • Evaluation environment: In the lab, Closed, Open • Evaluation type: Experimental - Testing • Type of component under evaluation: Model, Software, Hardware • Evaluation tool: Proprietary • Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Functional • Evaluation performance indicator: V&V process criteria
Use case scenarios • VALU3S_WP1_Automotive_3 - Node connection to cloud • VALU3S_WP1_Industrial_6 - Fault tolerance for motor position sensor data • VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal • VALU3S_WP1_Railway_1 - Inject, detect and recover • VALU3S_WP1_Railway_2 - Controlled vs random injection • VALU3S_WP1_Railway_3 - Systematic and random failures verification
Strengths • Using a behaviour model, the method can use coverage driven testing to reach interesting system states and provide out-of-specification inputs there.
Limitations • Depends on a behaviour model • For very complex systems, it is often just not feasible to run a robustness test suite of the size objectively needed.
References • [MRT1] "Standard Glossary of Software Engineering Terminology (ANSI)". The Institute of Electrical and Electronics Engineers Inc. 1991 • [MRT2] R. Bloem, K. Chatterjee, K. Greimel, T. A. Henzinger, and B. Jobstmann, 'Specification-centered robustness', in *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, Vasteras, Sweden, Jun. 2011, pp. 176–185, doi: 10/cwfvw7. • [MRT3] A. Takanen, J. DeMott, C. Miller, and A. Kettunen, *Fuzzing for software security testing and quality assurance*. 2018. • [MRT4] H. A. Salem and J. Song, 'A Review on Grammar-Based Fuzzing Techniques', p. 10, 2019. • [MRT5] A. Savary, M. Frappier, M. Leuschel, and J.-L. Lanet, 'Model-Based Robustness Testing in Event-B Using Mutation', in *Software Engineering and Formal Methods*, vol. 9276, R. Calinescu and B. Rumpe, Eds. Cham: Springer International Publishing, 2015, pp. 132–147.

• [MRT6] J.-C. Fernandez, L. Mounier, and C. Pachon, 'A Model-Based Approach for Robustness Testing', in *Testing of Communicating Systems*, vol. 3502, F. Khendek and R. Dssouli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 333–348. • [MRT7] O. Zendel, W. Herzner, and M. Murschitz, 'VITRO - Model based vision testing for robustness', in *IEEE ISR 2013*, Seoul, Korea (South), Oct. 2013, pp. 1–6, doi: 10/ghj3vj.
Related standards: -
Keywords: Model-based testing, Robustness testing, Smart fuzzing, Contract violation.

3.3.6 Model-Based Testing

Name: Model-based testing
Purpose: Derive tests from (semi-)formal behaviour models or test models. In general, these tests are functional tests but solutions to test non-functional properties like performance or robustness are available.
Description: Model-Based Testing is an approach for testcase generation for black-box testing. From a model specifying the behaviour of the system under test (or a sub-set thereof), test cases are generated, often driven by some notion of coverage of the model. There are several approaches described in literature, using several modelling formalisms and several commercial tools are available as well. [MBT1] gives an overview and a taxonomy of approaches. The web page at [MBT2] collects literature and tool references on the topics. Many applications of the approach are in the safety critical systems domain (see [MBT3] for details), probably because there the additional effort of creating a sufficiently complete model for testing is easier to argue.
Relationship with other methods: Model-based testing as a general concept is specialized in *Model-based mutation testing*.
Tool support: MoMuT (https://momut.org) as an in-project tool. A list of 18 maintained and 23 not anymore maintained tools can be found here: http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html
Layers of the multi-dimensional framework • Evaluation environment: In the lab, Closed, Open • Evaluation type: Experimental - Testing • Type of component under evaluation: Model, Software, Hardware • Evaluation tool: Proprietary • Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Functional • Evaluation performance indicator: V&V process criteria
Use case scenarios • VALU3S_WP1_Automotive_3 - Node connection to cloud • VALU3S_WP1_Industrial_5 - Motor speed control • VALU3S_WP1_Industrial_6 - Fault tolerance for motor position sensor data • VALU3S_WP1_Industrial_7 - Safety behaviour for missing motor position sensor data • VALU3S_WP1_Industrial_8 - Safety behaviour for remote control terminal connection failure

• VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal • VALU3S_WP1_Railway_1 - Inject, detect and recover • VALU3S_WP1_Railway_2 - Controlled vs random injection • VALU3S_WP1_Railway_3 - Systematic and random failures verification
Strengths • Systematic generation of test cases, ensuring a consistent degree of test quality • Model updates for changed or new requirements are in most cases easier and faster to do than updating several hundreds or thousands of tests that might be affected. From the updated model, corrected tests can be generated. Many test case generators strive to leave unrelated tests unaffected. • Optimised test suites can achieve the same test quality with less test execution efforts.
Limitations • Testing is an inherently incomplete approach. Generated, high quality tests cannot change this. • The effort of creating a test model is often seen as an otherwise unnecessary effort. It can be balanced with reduced test design efforts. • The quality of the generated tests depends not only on the model and the tool, but also on the coverage criterion used to drive the generation of the tests.
References • [MBT1] Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability 22(5), 297–312 (2012) • [MBT2] http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html • [MBT3] H. G. Gurbuz and B. Tekinerdogan, 'Model-based testing for software safety: a systematic mapping study', Software Qual J, vol. 26, no. 4, pp. 1327–1372, Dec. 2018, doi: 10/ghj4xp. • [MBT4] Model-Based Testing Essentials - Guide to the ISTQB Certified Model-Based Tester: Foundation Level | Wiley, Wiley.com. https://www.wiley.com/en-gb/Model+Based+Testing+Essentials+Guide+to+the+ISTQB+Certified+Model+Based+Tester%3A+Foundation+Level-p-9781119130017 (accessed Nov. 15, 2020).
Related standards: Model-Based testing is highly recommended for SIL3 and SIL4 in IEC61508. ISTQB offers a certification for model-based testers [MBT4].
Keywords: Behaviour models, Test case generation.

3.3.7 Risk-Based Testing

Name: **Risk-based testing**
Purpose: Risk-based testing aims to reduce risks of software-based systems and to increase efficiency and effectiveness of software testing through integration of risk assessment in the testing process.
Description: Risk-based testing uses risk assessment to guide testing. In this context, risk is understood as a factor that may have negative consequences and is typically expressed in terms of likelihood (i.e., probability of failure) and impact (e.g., cost or severity of failure). Risk assessment is integrated into the entire test process, i.e., test planning, design and implementation, execution and

evaluation. The intuition behind the approach is to focus testing on scenarios that trigger critical situations. Risk-based testing approaches can be integrated in industrial test processes [RBT1].

It is worth noting that risk-based testing is not only about improving testing but testing also may support and improve risk assessment by providing details about known risks or detection of new ones.

Several risk-based testing approaches have been developed during the last years.

- The PRISMA (Product Risk Management) approach [RBT7] first identifies risks (business and technical) and then categorizes them into four different risks levels, represented in a risk matrix. Different test approaches are used for each risk level, e.g., tests for high-risk areas involve more reviews or have stricter exit criteria.
- Risk-Based Test Case Prioritization Using Fuzzy Expert Systems [RBT4] supports the prioritization of requirements-based tests and consists of the 4 steps
    1. Risk estimation by correlating with requirements (determine risk indicators that effectively indicate defects, including requirement complexity, requirement size, requirement modification status, and potential security threats)
    2. Risk exposure calculation for requirements (as a weighted mean of risk indicator values)
    3. Risk exposure calculation for risk items (based on risk exposure values of risk items)
    4. Prioritization of requirements and test cases (based on risk exposure values linked to the requirements)

    Quantities such as requirements modification status or potential security threats are subjective and thus the approach applies fuzzy expert systems to reduce that subjectivity.
- SmartTesting is a process for risk-based test strategy development consisting of 7 core steps [RBT5]:
    1. Definition of risk items (functional and non-functional aspects)
    2. Probability estimation for each risk item (e.g., from historical data)
    3. Impact estimation for each risk item
    4. Computation of risk values
    5. Determination of risk levels
    6. Definition of test strategy based on the different risk levels
    7. Refinement of test strategy to match characteristics of the components
- RACOMAT is a risk management tool [RBT6] following the ISO 31000 standard developed during the EU project RASEN. It uses formal risk modelling and assessment (based on CORAS) and existing libraries such as Common Attack Pattern Enumeration and Classification (CAPEC) to enable automated security testing. Since it can be reused it allows to focus on elements that have not yet been tested but influence the impact or elements where likelihood estimation is difficult.

    RACOMAT also supports test-based risk assessment. Observations of the system under test can be valuable input to the risk model and helps estimating likelihood and impact for given scenarios.

In order to compare and categorize various risk-based approaches, a taxonomy has been developed [RBT2]. It distinguishes three top-level classes:

- Context: The overall context is characterized through identification of risk drivers that determine the direction of the processes, ranging from safety and security to business and compliance. The quality properties to be considered are determined, typically including functionality, security and reliability, and the relevant elements (called risk items) are identified.

- Risk Assessment: The approaches are compared in terms of the risk factors they consider to be influential (like risk exposure), the technique to estimate and evaluate risks (list-based or formal) as well as the scale to determine the risk level (quantitative or qualitative). Further, the degree of automation of the methods used can be measured.
- Risk-Based Testing Strategy: This class differentiates methods depending on which parts of the testing process are based on risk assessment. It focuses on the phases risk-based planning (including test objectives and techniques, completion criterion as well as resource planning), risk-based test design & implementation (including preparation of test data, test selection or test automation) and risk-based text execution & evaluation (including monitoring and reporting).

This taxonomy has recently been refined to meet the needs of current standards [RBT3].

Relationship with other methods: Risk-based testing can be used to prioritise tests in all *Testing* methods.

Tool support: RACOMAT (https://www.fokus.fraunhofer.de/en/sqc/racomat)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab, Closed, Open
- Evaluation type: Experimental - Testing
- Type of component under evaluation: Software, Hardware
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional
- Evaluation performance indicator: V&V process criteria

Use case scenarios

- VALU3S_WP1_Automotive_3 - Node connection to cloud

Strengths

- Improved efficiency (reduced testing time and budget)
- Improved teste effectiveness (detection of defects, increased detection rate of tests)

Limitations

- Testing is inherently incomplete. Prioritization by risk can reduce the remaining risk, but cannot address the inherent incompleteness.

References

- [RBT1] M. Felderer, R. Ramler, Integrating risk-based testing in industrial test processes, Software Quality Journal 22, p 543-575, 2014.
- [RBT2] M. Felderer, I. Schieferdecker, A taxonomy of risk-based testing, International Journal on Software Tools for Technology Transfer 16, p 559-568, 2014.
- [RBT3] J. Großmann, M. Felderer, J. Viehmann, I. Schieferdecker, A Taxonomy to Assess and Tailor Risk-Based Testing in Recent Testing Standards, IEEE Software 37, p 40-49, 2020.
- [RBT4] C. Hettiarachchi, H. Do, and B. Choi, Risk-based test case prioritization using a fuzzy expert system. Information and Software Technology, 69, p 1-5, 2016

• [RBT5] R. Ramler, M. Felderer, A process for risk-based test strategy development and its industrial evolution, International Conference on Product-Focused Software Process Improvement, Springer, p355-371, 2015. • [RBT6] J. Viehmann and F. Werner, Risk assessment and security testing of large sclae network systems with RACOMAT, Proceedings of Risk Assessment and Risk-Driven Testing, p 3-17, 2015. • [RBT7] E. van Veendaal, The PRISMA Approach: Practical Risk-Based Testing, 2012.
Related standards: ISO/IEC/IEEE 29119, ETSI EG 203251 (testing), ISO 31000 (risk management)
Keywords: Test optimization, Risk analysis.

3.3.8 Signal Analysis and Probing

Name: **Signal analysis and probing**
Purpose: Speeding up System-on-Chip validation.
Description: A method to validate signals on an IC based on using a tester setup that probes the IC to measure the signals on the chip. These signals are post-processed on the tester by means of complex signal analysis in order to assess the SoC’s performance.
Relationship with other methods: No relevant specific relationships at this moment. They will nonetheless be studied in detail during the project.
Tool support: Semiconductor test equipment.
Layers of the multi-dimensional framework • Evaluation environment: In the lab, Closed, Open • Evaluation type: Experimental - Testing • Type of component under evaluation: Software, Hardware • Evaluation tool: Proprietary • Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Functional • Evaluation performance indicator: V&V process criteria
Use case scenarios • VALU3S_WP1_Automotive_8 - Automatic Emergency Braking (AEB) • VALU3S_WP1_Automotive_12 - ADAS system has to reliable and has to comply with Safety standards • VALU3S_WP1_Automotive_13 - SoC validation with intensive use of SoC internal self-tests
Strengths • Proven method in IC/SOC validation
Limitations • Complex • Needs expensive equipment • Difficult to parallelize • Takes time to set up • Testing requires probing of the IC
References:

- [SAP1] Probe: Signal Analysis Measurement Fundamentals. Online, https://probe.co.il/signal-analysis-measurement-fundamentals/

Related standards: This method can be a part of the *Testing* activities that standards prescribe, e.g., the functional safety ones.

Keywords: Signal testing, ATE, IC probing.

3.3.9 Software Component Testing

Name: **Software component testing**

Purpose: Software component testing is performed on each individual component of the system separately without integrating with other components. Static analysis of code as well as data flow analysis shall be used to detect poor and potentially incorrect program structures or specifications failures.

Description: Boundary and structure-based tests are to be executed to detect specific software errors. Depending on the Hardware/Software design, unit testing might be taken into account, too, with several different classes of testing, such as equivalence class and input partition, model-based test case generation and so on. The latter testing class that will be performed is the performance testing (stress tests, response timings, etc.), to ensure that the working capacity of the system is sufficient to meet the specified requirements

Relationship with other methods: All *Testing* and *Runtime Verification* methods are or may be related to this

Tool support: Lint, Vector Cast (https://www.vector.com/), SonarQube (https://www.sonarqube.org/), Jenkins (https://www.jenkins.io/), C-Unit (http://cunit.sourceforge.net/), Tessy (https://www.razorcat.com/en/product-tessy.html), JUnit (https://junit.org/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental – Testing
- Type of component under evaluation: Software
- Evaluation tool: Proprietary, Open Source
- Evaluation stage: Validation
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional - Safety
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_Agriculture_1 - Vehicle switching from parallel guidance to manual mode
- VALU3S_WP1_Agriculture_2 - Vehicle switching from manual mode to parallel guidance
- VALU3S_WP1_Agriculture_3 - Transmission line disturbances
- VALU3S_WP1_Agriculture_4 - Disturbances in IMU communication
- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths • Enabler of Continuous Integration • Enable Regression Tests • Design and implementation faults detection • Avoids poor coding related unexpected behaviours
Limitations • Code overhead • Time consuming • Test personnel must be different from developers
References: • [SCT1] Kim HK., Kwon OH. (2005) SCTE: Software Component Testing Environments. In: Gervasi O. et al. (eds) Computational Science and Its Applications – ICCSA 2005. ICCSA 2005. Lecture Notes in Computer Science, vol 3481. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11424826_15 • [SCT2] Franz F. (2008) Experiences with Evolutionary Timing Test of Automotive Software Components. In: Margaria T., Steffen B. (eds) Leveraging Applications of Formal Methods, Verification and Validation. ISoLA 2008. Communications in Computer and Information Science, vol 17. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88479-8_29 • [SCT3] Gao J.Z., Wu Y. (2004) Testing Component-Based Software – Issues, Challenges, and Solutions. In: Kazman R., Port D. (eds) COTS-Based Software Systems. ICCBSS 2004. Lecture Notes in Computer Science, vol 2959. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24645-9_2
Related standards: All functional safety standards
Keywords: Component, Testing, V-cycle.

3.3.10 Test Parallelization and Automation

Name: **Test parallelization and automation**
Purpose: Cost- and time-effective system test environment.
Description: Complex systems require testing of a huge number of use cases and varieties of parameters. Expensive test equipment and time to market requirements demand efficient use of test resources and reliable result tracking. To accomplish this goal, several steps must be considered: 1. Test resource management: A typical system test setup consists of a device under test (DUT) and a device that generates input vectors to stimulate the DUT. Both are connected with a flexible (hardware or software) interface. An effective test environment consists of multiple test setups with similar or different properties for parallel testing. The resource management tool must assign a test task, defined by the user, automatically to the correct test setup for the respective test scenario. If the interface between the test vector generator and the DUT is flexible, the resource manager must set the required connections automatically, depending on the test case. The user must not care about which test setup is used for the test, and no manual interaction should be required. 2. Test scheduler: For a complex system test, thousands of tests need to be run. The scheduler must take in the list of tests and schedule them to run efficiently on the available test setups. In

combination with the resource manager, all test setups must be used in parallel for full efficiency of the test environment. To allow one to use a manual test setup (e.g. for debugging), the scheduler must offer an interface for the user to block a test setup from automatic tests. If there is no test scheduled and no manual test is done, the scheduler must do tests with random parameters to increase the robustness of the test item.

3. Test result evaluation: Logging and presentation of test results is a key factor for effective testing. The test environment shall log all parameters of a test, including timing and used resources and store it in a database. An overview report must be available at any time as well as detailed analysis of single test runs including a filter function to search for specific parameters. The database must offer a comparison function to compare multiple test runs and find correlations of test results.

The interface for the complete test environment must offer a web interface for remote control and remote result evaluation.

This approach allows a 24/7 usage of the test equipment, reduces errors caused by human interaction but still allows seamless integration of manual tests if required. The logging and result evaluation methods allow for full reproducibility of tests and help to find correlations between test parameters and test results.

Relationship with other methods: It can be combined with other *Testing* methods.

Tool support: Couchbase (https://www.couchbase.com/), TestLink (http://testlink.org/), Jenkins (https://www.jenkins.io/)

Layers of the multi-dimensional framework:

- Evaluation environment: Lab (web-based remote access)
- Evaluation type: Experimental – Testing
- Type of component under evaluation: Hardware, Software
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Validation, Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios:

- VALU3S_WP1_Automotive_8 - Automatic Emergency Braking (AEB)
- VALU3S_WP1_Automotive_12 - ADAS system has to reliable and has to comply with Safety standards
- VALU3S_WP1_Automotive_13 - SoC validation with intensive use of SoC internal self-tests

Strengths:

- Efficient use of test resources (time and hardware) 24/7
- Minimal human interaction, failproof
- Efficient result evaluation
- Full reproducibility

Limitations:

- Complex initial setup
- Proprietary customization to new test scenarios required

• Test creation and portability of all potential real-life situations impossible, relevant few test cases to be developed • On-board resources for validation data analysis and assessment need themselves to be validated
References: • [TPL1] Automation Panda: To Infinity and Beyond: A Guide to Parallel Testing. Online, https://automationpanda.com/2018/01/21/to-infinity-and-beyond-a-guide-to-parallel-testing/
Related standards: Testing recommended in all functional safety standards.
Keywords: Automation, Robustness testing, System testing.

3.4 Runtime Verification

This group of methods focuses on verifying a system during execution.

Today's automated systems are continuously growing in complexity, notably in what respects to the nature of their distributed architectures, the size and number of software components, and the amount of concurrency associated with these components. This makes most state-of-the art static verification techniques unscalable and impracticable. Runtime verification techniques are lightweight alternatives that make use of monitors, build based on formal specifications, that observe the target system and verify at execution time whether a set of specifications are met. A broad overview can be found, e.g., in the survey by Leucker and Schallhart [12].

The runtime verification methods proposed in this section target mainly embedded systems and attempt to address some of their limitations. *Dynamic Analysis of Concurrent Programs* focuses on concurrency, and exploits the use of test-runs, possibly enriched with injected noise, to be able to produce monitors that predict beforehand when errors might occur. *Runtime Verification Based on Formal Specification* investigates the expressivity of different temporal logics suitable for runtime verification of CPS and explores the usage of Domain Specific Languages to facilitate their adoption by different stakeholders. *Test Oracle Observation at Runtime* further pursues the goal of predicting errors beforehand, shared with Section 3.4.1, using quantitative methods, i.e., using precise metrics to estimate how close is a system from reaching an invalid state.

3.4.1 Dynamic Analysis of Concurrent Programs

Name: **Dynamic analysis of concurrent programs**
Purpose: To find errors in synchronisation of concurrently executing threads, processes, or any other concurrently executing tasks.
Description: Computing platforms with many CPUs and/or CPU cores are omnipresent nowadays. This holds even for embedded systems. In order to fully benefit from such architectures, concurrent programming is necessary. However, programming concurrent applications is significantly harder than producing sequential code since one additionally needs to properly synchronise all the concurrently running tasks. Improper synchronisation often leads to errors (such as data races, deadlocks, atomicity violations, order violations, etc.) that can have disastrous consequences but that

are extremely difficult to detect. The reason is that such errors can manifest extremely rarely, and it is thus very difficult to catch them by traditional testing.
At the same time, concurrency is a very significant obstacle for current analysis and verification methods too. Methods of lightweight static analysis often suffer from quite significant imprecision on concurrent code despite a lot of effort has been invested and is still being invested into their improvement in this area. Model checking can in theory provide a guarantee that all such errors will be discovered in a precise way, but it suffers from scalability issues on large systems, and it may also have problems with various features of real-life code (user inputs, network communication, dealing with file systems and databases, etc.). Various forms of bounded model checking or systematic testing of concurrent code [DAC1, DAC2] improve the scalability but still suffer from similar issues.
An alternative to the above approaches is *dynamic analysis* (performed at runtime during testing runs or even during production runs) that not only monitors the activity of the system under test but tries to extrapolate the behaviour seen in the testing runs and to produce warnings about possible errors in the system even though such errors have not been witnessed in the performed runs. Various specialised checkers, e.g., [DAC5, DAC6, DAC7, DAC8, DAC9, DAC10], capable to produce such warnings have been developed for different kinds of concurrency errors, such as data races, deadlocks, atomicity violations, or breakage of so-called contracts for concurrency, which can encode numerous more specific kinds of concurrency errors (atomicity violation, order violation, etc.).
These approaches cannot guarantee overall soundness: they can at most guarantee soundness wrt the witnessed testing runs – this is, they can guarantee that if symptoms of an error appear in these runs, a warning will be produced. On the other hand, they can be more scalable and can cope better with various features of real-life code than model checking and can avoid various sources of false alarms that are problematic for static analysis (such as which memory objects are available to which threads at which moment).
Moreover, the effectiveness of extrapolating dynamic analyses can be further improved by combining them with so-called *noise injection* methods [DAC3, DAC4]. Such methods try to disturb the scheduling of concurrently executing threads (e.g., by injecting context switches, delays, etc. into the execution) such that thread interleavings that are otherwise rare manifest more often. (Note that the produced interleavings are still valid – unless real-time features are in play.) With noise injection, one can sometimes even use precise checkers (i.e., without any extrapolation and hence with no false alarms) and still can catch very rare behaviours even in industrial applications [DAC11].

Relationship with other methods: an alternative approach to *Model checking* and *Source code static analysis* (formal methods in general); of course, combinations of these approaches are possible too.

Tool support: ANaConDA (http://www.fit.vutbr.cz/research/groups/verifit/tools/anaconda/), ConTest (https://www.research.ibm.com/haifa/conferences/hvc2010/present/Testing_and_Debugging_Concurrent_Software.pdf), Road Runner (https://github.com/stephenfreund/RoadRunner), Helgrind (https://www.valgrind.org/docs/manual/hg-manual.html), LLVM Thread Sanitizer (https://clang.llvm.org/docs/ThreadSanitizer.html).

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Experimental - Monitoring
- Type of component under evaluation: Software

- Evaluation tool: Open source
- Evaluation stage: Validation
- Logic of the component under evaluation: Thinking, Acting
- Type of requirements under evaluation: Functional, Non—functional - Safety
- Evaluation performance indicator: SCP criteria, V&V process criteria

Use case scenarios

- VALU3S_WP1_Automotive_1 - Radar/camera advanced detection and tracking
- VALU3S_WP1_Automotive_2 - Radar + camera cooperation
- VALU3S_WP1_Automotive_3 - Node connection to cloud
- VALU3S_WP1_Automotive_6 - Transmission line switching
- VALU3S_WP1_Automotive_7 - Safety of vehicle during switch between routers
- VALU3S_WP1_Automotive_8 - Automatic Emergency Braking (AEB)

Strengths

- Ability to detect even rarely manifesting concurrency-related errors.
- Better scalability than various heavier-weight formal methods or their bounded variants.
- Better handling of various not purely computational features of real-life systems (inputs, network connection, database connections, etc.).
- Can better avoid many false alarms hard to avoid in light-weight static analysis approaches.

Limitations

- Does not provide formal soundness guarantees.
- When using extrapolation, false alarms may arise.
- Some of the efficient extrapolating analyses are highly specialised in the considered class of errors.
- Monitoring of the running system and noise injection can slow-down the application significantly.
- Needs the system to be verified in a runnable form, including a test harness (unless the system is operated and monitored in production).
- Noise injection can invalidate real-time features of a system.

References

- [DAC1] Musuvathi, M., Qadeer, S., Ball, T., Basler, G., Nainar, P., Neamtiu, I.: Finding and Reproducing Heisenbugs in Concurrent Programs. Proc. of OSDI'08, USENIX, 2008.
- [DAC2] Wu, J., Tang, Y., Hu, H., Cui, H., Yang, J.: Sound and Precise Analysis of Parallel Programs through Schedule Specialization. Proc. of PLDI'12, ACM, 2012.
- [DAC3] Edelstein, O., Farchi, E., Goldin, E., Nir, Y., Ratsaby, G., Ur, S.: Framework for Testing Multi-threaded JavaPrograms. Concurrency and Computation: Practice and Experience 15(3-5), 2003.
- [DAC4] Fiedor, J., Hruba, V., Krena, B., Letko, Z., Ur, S., Vojnar, T.: Advances in Noise-based Testing of Concurrent Software. Software Testing, Verification, and Reliability 25(3), 2015.
- [DAC5] Hammer, C., Dolby, J., Vaziri, M., Tip, F.: Dynamic Detection of Atomic-Set-Serializability Violations. Proc. of ICSE'08, ACM, 2008.
- [DAC6] Flanagan, C., Freund, S., Yi, J.: Velodrome: A Sound and Complete Dynamic Atomicity Checker for Multithreaded Programs. Proc. of PLDI'08, ACM, 2008.

• [DAC7] Joshi, P., Park, C.S., Sen, K., Naik, M.: A Randomized Dynamic Program Analysis Technique for Detecting Real Deadlocks. Proc. of PLDI'09, ACM, 2009. • [DAC8] Flanagan, C., Freund, S.: FastTrack: Efficient and Precise Dynamic Race Detection. Proc. of PLDI'09, ACM, 2009. • [DAC9] Rhodes, D., Flanagan, C., Freund, S.N.: BigFoot: Static Check Placement for Dynamic Race Detection. Proc. of PLDI'17, ACM, 2017. • [DAC10] Dias, R., Ferreira, C., Fiedor, J., Lourenco, J., Smrcka, A., Sousa, D., Vojnar, T.: Verifying Concurrent Programs Using Contracts. Proc. of ICST'17, IEEE, 2017. • [DAC11] Fiedor, J., Muzikovska, M., Smrcka, A., Vasicek, O., Vojnar, T.: Advances in the ANaConDA Framework for Dynamic Analysis and Testing of Concurrent C/C++ Programs. Proc. of ISSTA'18, ACM, 2018.
Related standards: -
Keywords: Dynamic analysis, Concurrency, Deadlock, Data race, Atomicity, Order violation, Extrapolation, Noise injection.

3.4.2 Runtime Verification Based on Formal Specification

Name: **Runtime verification based on formal specification**
Purpose: Formally specify properties of runtime observations and verify them using automatically generated monitors. Focus on properties that are impractical when using static techniques (e.g., model checking and theorem proving, due to the size of complexity of the analysis, and also when precise data – typically related to non-functional properties – is only available upon execution time).
Description: With the complexity of today's automated systems continuously increasing, the current state-of-the-art of exhaustive verification techniques, in general, takes impracticable time to compute. Runtime Monitoring (RM) and Runtime Verification (RV) techniques are lightweight alternatives that make use of monitors which are small executable components that observe the behaviours of the target system to determine if those behaviours satisfy the associated specifications. RM has been around for more than 30 years and has been used in several scopes of analysis of traces produced by a running system, including profiling, performance analysis, fault-detection, among other relevant aspects that ensure the well-being of the system. The concept of monitor is central in RM, as it is the (software or hardware) computational entity that collects the data traces from the system and proceeds with the aforementioned analysis. RV is an evolution of RM, where monitors are synthesised from formal specifications in an automatic way, via specialized synthesis algorithms. Hence, the monitors adopted in RV provide further guarantees over the monitors employed in RM, since the former will observe the precise meaning of the specifications. A broad overview can be found, e.g., in [RVF1-RVF4]. RV techniques typically consider formalisms such as temporal logics, state machines, and regular expressions, focusing mainly on monitor synthesis. Different flavours of temporal logic are considered in RV to verify components viewed as black-boxes, such as LTL [RVF5], interpreted over finite traces [RVF6], and its extensions such as MITL [RVF7], TLTL [RVF6], STL [RVF8], and RMTLD [RVF13, RVF14]. RV for LTL has been recently extended also to take into account assumptions on the system behaviours, specified as a formal model describing the execution traces that the system is expected

to follow [RVF9]. Assumptions allow the monitors to be more precise, emitting a conclusive verdict before the corresponding monitor without assumption, and predictive, outputting conclusive verdicts before it actually sees it from the input trace or events. An important challenge in this context is related to the expressiveness of the assumptions and how to use model checking techniques for infinite-state or timed systems at runtime to explore the belief state space.

When making use of monitors in critical systems, one has to ensure that they neither negatively influence the security aspects of the original system nor affect the functional and the safety non-functional requirements of the system (e.g., task scheduling [RVF12]). Guaranteeing that the deployment of such solutions does not negatively influence the dependability properties of systems can be overly complicated and time-consuming when no proper integration methods are used.

The generation of monitors from formal specifications is potentially supported by Domain-Specific Languages (DSL), which provide support to abstract the formalities of correctly integrating monitoring architectures in the target system and letting developers focus on what needs to be monitored [RVF10]. Typically, these techniques must be combined with orchestration of the target system's software architecture so that monitors can be coupled, observe its execution, and identify, during runtime, aspects that were not foreseen during the design-phase or errors that could not be proved to be absent via static verification methods [RVF11].

Relationship with other methods: *Model checking.*

Tool support: LOLA (https://www.react.uni-saarland.de/tools/lola/), LARVA (http://www.cs.um.edu.mt/svrg/Tools/LARVA/), JAVA-MOP (https://github.com/runtimeverification/javamop), AMT, NuRV, RMTLD3Synth (https://github.com/anmaped/rmtld3synth), MARS, Spectra (https://pajda.fit.vutbr.cz/testos/spectra)

Layers of the multi-dimensional framework

- Evaluation environment: In the Lab, Closed, Open
- Evaluation type: Experimental – Monitoring, Analytical – Formal
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Thinking, Sensing, Acting
- Type of requirements under evaluation: Functional, Non-Functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery
- VALU3S_WP1_Healthcare_1 - Bone segmentation
- VALU3S_WP1_Railway_2 - Controlled vs random injection

Strengths

- Verification, during execution time, of properties of the target system that are hard or impossible to capture during design time
- Lightweight scalable application of formal methods

• Rigorous specification of monitoring properties • Possibility of increasing of the overall system reliability without compromising security and safety aspects, via a correct generation of monitors
Limitations • The use of runtime monitoring solutions inevitably implies extra overheads in the target system • It is not always easy to define formal specifications (need specialized expertise from users) • A black box specification requires to know how an internal condition (e.g., a fault) manifests itself on the observable interface of the system/component. Assumption-based RV requires a formal specification of the assumption.
References • [RVF1] Falcone, Y., Havelund, K., Reger, G. "A tutorial on runtime verification." In Engineering Dependable Software Systems, 34:141–175, 2013. doi: 10.3233/978-1-61499-207-3-141 • [RVF2] Havelund, K., Goldberg, A. "Verify your runs." In Proceedings of the first IFIP TC 2/WG 2.3 Conference on verified software: theories, tools, experiments (VSTTE'05), volume 4171 of LNCS. Springer, pp 374–383 • [RVF3] Leucker, M., Schallhart, C. "A brief account of runtime verification." JLAMP, 78(5):293–303, 2009 • [RVF4] Bartocci, E., Falcone, Y. (eds) "Lectures on runtime verification—introductory and advanced topics.", volume 10457 of lecture notes in computer science. Springer, 2018 • [RVF5] Pnueli, A. "The Temporal Logic of Programs". In FOCS, pages 46–57, 1977 • [RVF6] Bauer, A., Leucker, M., Schallhart, C. "Runtime Verification for LTL and TLTL." In ACM Transactions on Software Engineering and Methodology, 20(4):14–64, September 2011. doi: 10.1145/2000799.2000800 • [RVF7] Maler, O., Nickovic, D., Pnueli, A. "From MITL to timed automata." In Proceedings of the 4th International Conference on Formal Modeling and Analysisof Timed Systems (FORMATS'06), 2006, pp. 274–289 • [RVF8] Maler, O., Nickovic, D. "Monitoring temporal properties of continuous signals." In Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS'04), 2004, pp. 152–166 • [RVF9] Cimatti, A., Tian, C., Tonetta, S.: "Assumption-Based Runtime Verification with Partial Observability and Resets." In Runtime Verification (RV 2019), pp. 165–184. Springer. doi: 10.1007/978-3-030-32079-9_10 • [RVF10] Medhat, R. et al. "Runtime Monitoring of Cyber-Physical Systems Under Timing and Memory Constraints." In ACM Trans. Embed. Comput. Syst. Vol. 14, no. 4 (Oct. 2015), 79:1–79:29. • [RVF11] Cassar, I. et al. "A Survey of Runtime Monitoring Instrumentation Techniques." In Electronic Proceedings in Theoretical Computer Science vol. 254 (Aug. 2017), pp. 15–28. • [RVF12] Khan, M. T., Serpanos, D., Shrobe, H. "A rigorous and efficient run-time security monitor for real-time critical embedded system applications." In 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), 100–105. December 2016. • [RV13] Pedro, A., Leucker, M., Pereira, D., Pinto, J.S. "Real-time MTL with durations as SMT with applications to schedulability analysis." In TASE 2020 14th Internaltional Symposium on Theoretical Aspects of Software Engineering, December 11-13, 2020, Hangzhou, China

- [RV14] Pedro, A., Pereira, D., Pinto, J.S., Pinho, L.M. "Monitoring for a decidable fragment of MTLD" In 15th International Conference on Runtime Verification (RV'15). 22 to 25, Sep, 2015. Vienna, Austria.

Related standards: -

Keywords: Runtime monitoring, Runtime verification, Safety, Formal methods.

3.4.3 Test Oracle Observation at Runtime

Name: **Test oracle observation at runtime**

Purpose: Dynamically assess the robustness of a system behaviour during its runtime by measuring how far the system is from satisfying or violating a property expressed in a formal specification language.

Description: Runtime verification is a pragmatic, yet rigorous, technique to reason about software and systems in a systematic manner. It borrows the concept of *formal specifications* and combines it with the analysis of *individual system behaviours*. Thus, this technique does not encounter the scalability issues that are commonly associated to the formal verification. Runtime verification is a black-box method that does not require an abstract model. As a consequence, it can be used both at the design and the deployment time of a system. Runtime verification has found its way in many application areas, including the field of monitoring CPS.

Signal Temporal Logic (STL) [TOO1] is a specification formalism for expressing real-time temporal properties of CPS. In its original form, STL was designed as a declarative specification language for runtime monitoring. We can see STL specifications as binary classifiers that partition behaviours into good and bad ones (Figure 3.6). The following requirement is a typical example of an informal English specification that describes the rising of a continuous signal and can be naturally formalized in STL:

"The rise time of the voltage V from V_{min} to V_{max} must be smaller or equal than T_{rise} is".

The STL specification that describes the above requirement is expressed as follows.

$v_l = (V \leq V_{low});$

$v_m = (V > V_{low} \text{ and } V \leq V_{high});$

$v_h = (V > V_{high});$

$\text{always } (v_l \rightarrow (v_m \text{ until}[0, T_{rise}] \ v_h));$

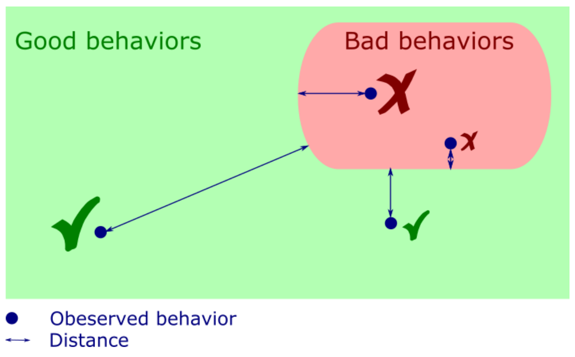


Figure 3.6 Good vs. Bad Behaviours

Quantitative semantics of specification languages: Consider the predicate $x \geq 5$ and the valuation in which x equals to 2. The qualitative evaluation of this simple specification tells us that the valuation violates the property. Given this numerical predicate over a real-valued variable and a variable valuation, we can also answer the question on how far the valuation is from satisfying or violating the predicate. This rich feedback contrasts the classical yes/no answer that we typically get from reasoning about Boolean formulas. Fainekos and Papas [TOO2] lifted this quantitative property of numerical predicates to the temporal case, giving rise to the quantitative semantics for STL. This extension replaces the binary satisfaction relation with a real valued *robustness degree* function while preserving the syntax of the specification language. The robustness degree indicates how far an observed signal from satisfying or violating the specification is. The original quantitative STL semantics have been defined in terms of the infinite form. This spatial notion of robustness was consequently extended in several directions, including time robustness and average robustness. The quantitative semantics is a powerful extension that allows ordering specifications according to their goodness or badness and opens up the possibility of building new applications.

Applications: Runtime verification is a versatile technology that has been successfully combined with many other analysis and synthesis methods. In particular, STL qualitative and quantitative monitors have been used as the basis for many new techniques for analysing CPS, such as falsification-based testing [TOO3], parameter synthesis [TOO4], specification mining [TOO5], and fault explanation [TOO6, TOO7].

Relationship with other methods: Can be used with coverage driven tests generated by *Model-based testing*. The described approach is a variant of *Runtime verification based on formal specifications*.

Tool support: RTAMT, RTAMT-CPP (https://github.com/nickovic/rtamt)

Layers of the multi-dimensional framework

- Evaluation environment: In the Lab, Closed and Open
- Evaluation type: Experimental - Monitoring
- Type of component under evaluation: Model, Hardware, Software
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Thinking, Acting
- Type of requirements under evaluation: Functional, Non-functional – Safety
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_5 - Motor speed control
- VALU3S_WP1_Industrial_6 - Fault tolerance for motor position sensor data
- VALU3S_WP1_Industrial_7 - Safety behaviour for missing motor position sensor data
- VALU3S_WP1_Industrial_8 - Safety behaviour for remote control terminal connection failure
- VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal

Strengths

- Quantitative indicator on the degree of behaviour robustness with respect to the specification
- Computationally inexpensive
- Can be used both during the design of the system and during its operation
- Black-box technique that does not require the system model
- Can be used as the basis for developing other V&V methods

Limitations

- Not exhaustive – analysis of an individual behaviour

• Passive observation of a behaviour (unless combined with test generation or runtime assurance) • Specification languages have certain expressiveness limits
References • [TOO1] O. Maler and D. Ničković, "Monitoring properties of analog and mixed-signal circuits," STTT, vol. 15, no. 3, pp. 247–268, 2013. • [TOO2] G. E. Fainekos and G. J. Pappas, "Robustness of Temporal Logic Specifications," in Proc. of {FATES} 2006: First Combined International Workshops on Formal Approaches to Testing and Runtime Verification, 2006, vol. 4262, pp. 178–192. • [TOO3] T. Nghiem, S. Sankaranarayanan, G. E. Fainekos, F. Ivancic, A. Gupta, and G. J. Pappas, "Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems," in Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, 2010, pp. 211–220. • [TOO4] E. Asarin, A. Donzé, O. Maler, and D. Ničković, "Parametric Identification of Temporal Properties," in Proc. of RV 2011: the Second International Conference on Runtime Verification, 2012, vol. 7186, pp. 147–160. • [TOO5] Z. Kong, A. Jones, and C. Belta, "Temporal Logics for Learning and Detection of Anomalous Behavior," IEEE Trans. Autom. Contr., vol. 62, no. 3, pp. 1210–1222, 2017. • [TOO6] Ezio Bartocci, Thomas Ferrère, Niveditha Manjunath, Dejan Nickovic: Localizing Faults in Simulink/Stateflow Models with STL. HSCC 2018: 197-206 • [TOO7] Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, Dejan Nickovic: Automatic Failure Explanation in CPS Models. SEFM 2019: 69-86
Related standards: -
Keywords: Runtime verification, Monitoring, CPS, Signal Temporal Logic.

3.5 Formal Verification

This group of methods aims to mathematically prove properties of a system or of information about it. We distinguish between formal verification for source code and formal verification in general.

3.5.1 Formal Source Code Verification

This sub-group of methods focuses on formal verification of source code as target system artefact type.

Formal verification of source code exploits a collection of techniques aimed at (automatically) verifying that a computer program satisfies a given set of properties. Software verifiers typically operate on a source-code representation of programs given in some high-level language (e.g. C, Java), in which properties are either user-defined (e.g. using assertions) or automatically-generated (e.g. memory safety, bounds checks, absence of runtime errors). Often software verifiers employ a number of different techniques operating at different levels of abstraction and ensuring different levels of formal guarantees on soundness and/or completeness.

Prominent examples of V&V methods for formal source code verification include software model checking, static analysis and abstract interpretation, symbolic execution, type systems, and theorem proving/SMT solving. Some methods can be applied also on other artefact types, e.g. model checking.

3.5.1.1 Deductive Verification

Name: **Deductive verification**

Purpose: Formal source code verification ensuring that the source code conforms to its formal specification

Description: Deductive software verification expresses the correctness of the source code as a set of mathematical statements, called verification conditions. A human user typically contributes in two ways:

1. formalising an informally stated specification for the source code, and
2. providing (if necessary) guidance to a verification system to show formally the conformance of the source code to the specification.

Properties to be verified are typically expressed in specification languages that combine first-order-logic and some theories (arithmetic, sets, arrays, bit vectors) to allow modelling the programming language while still allowing proof automation. These properties are verified in a modular way (e.g. method contracts using the Design by Contract paradigm, loop invariants, class invariants, lemmas, assumptions, assertions) and include both behavioural properties, e.g., calculates the longest repeated substring, and safety properties, e.g., null de-reference, out-of-bounds errors, termination, arithmetic overflow.

Weakest precondition/strongest postcondition calculus proof obligations are then discharged using either automated or interactive theorem provers such as those listed below:

- Interactive theorem provers (such as HOL, ACL2, Isabelle, or Coq) where the user is responsible for finding a proof, and in particular for providing values for quantifier instantiations.
- Automatic theorem provers where the proof is extracted from the specification, annotations, and the program
- Satisfiability Modulo Theories (SMT) solvers (Z3, Yices, Alt-Ergo, CVC3)

Relationship with other methods: *Theorem Proving and SMT Solving, Static Code Analysis, Model Checking.*

Tool support: There are many different tools in the verification system landscape with tools such as KeY (https://www.key-project.org/), OpenJML (http://www.openjml.org/), VerCors (https://vercors.ewi.utwente.nl/) and Why3 (http://why3.lri.fr) used to verify Java source code; tools such as VCC (https://github.com/microsoft/vcc), Frama-C (https://frama-c.com/) and Verifast (https://github.com/verifast/verifast) for verifying C source code; tools such as Code Contracts (https://github.com/microsoft/CodeContracts) and Spec# (https://rise4fun.com/SpecSharp) used to verify C# code; SPARKPro (https://www.adacore.com/sparkpro) and GNATprove (http://www.open-do.org/projects/hi-lite/gnatprove/) for verifying Ada code, AutoProof (http://comcom.csail.mit.edu/autoproof/) for verifying Eiffel code and purpose build languages and corresponding verifiers such as Dafny (https://github.com/dafny-lang/dafny) and KIV (https://www.uni-augsburg.de/de/fakultaet/fai/isse/software/kiv/).

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical - Formal

- Type of component under evaluation: Model, Software
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-Functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths

- Used to prove safety and behavioural properties of source code.
- Static verification detects runtime errors.
- High degree of automation using SMT solvers.
- When tools silently infer a particular fact (a termination measure, for instance), it reduces the burden on the user (but may appear as a gap in reasoning to an outsider).

Limitations

- Knowledge about the background theory implemented in a tool is often needed to understand a solution.
- Some tools require interactive theorem proving such as ghost code or inductive proofs via lemma functions.
- Some systems expose an explicit proof object (i.e., a derivation in a certain calculus) and this is typically too fine-grained to communicate easily.
- In some systems the user only works with the annotated source code which often does not make the argument structure explicit.

References

- [DEV1] Filliâtre, J. Deductive software verification. Int J Software Tools Technology Transfer 13, 397 (2011). https://doi.org/10.1007/s10009-011-0211-0
- [DEV2] Hähnle R., Huisman M. (2019) Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. In: Computing and Software Science. LNCS, vol 10000. Springer, Cham. https://doi.org/10.1007/978-3-319-91908-9_1
- [DEV3] Towards deductive verification of control algorithms for autonomous marine vehicles S Foster, M Gleirscher, R Calinescu- arXiv preprint arXiv:2006.09233, 2020 - arxiv.org
- [DEV4] Luo Z., Siegel S.F. (2018) Symbolic Execution and Deductive Verification Approaches to VerifyThis 2017 Challenges. In: ISoLA 2018. LNCS, vol 11245. Springer, Cham. https://doi.org/10.1007/978-3-030-03421-4_12
- [DEV5] Oortwijn W., Huisman M. (2019) Formal Verification of an Industrial Safety-Critical Traffic Tunnel Control System. In: Integrated Formal Methods. IFM 2019. Lecture Notes in Computer Science, vol 11918. Springer, Cham. https://doi.org/10.1007/978-3-030-34968-4_23
- [DEV6] Marieke Huisman, Rosemary Monahan, Peter Müller, Andrei Paskevich, Gidon Ernst. VerifyThis 2018: A Program Verification Competition. [Research Report] Université Paris-Saclay. 2019

Related standards: Related to standards that require verification of safety conditions. Examples include those in the avionic domain where standards DO-178C or DO-278A require evidence that

software implements its intended functions and does not perform unintended functions. It Includes also standards such as ISO 26262, which covers functional safety in the event of system failure, ISO 21448, which covers safety hazards that are inherent in the implemented functionality and can occur without parts of the system failing, and the forth-coming IEEE P2846, which covers assumptions for Models in Safety-Related Automated Vehicle Behaviour.
Keywords: Static verification, Specification, correctness, Safety properties, Behavioural properties, Functional correctness, Design by contract, Deductive verification.

3.5.1.2 Source Code Static Analysis

Name: **Source code static analysis**
Purpose: Deriving various runtime properties and finding various kinds of errors in programs without executing them at all or at least not under their original semantics.
Description: The domain of *static analysis* is rather broad and even the understanding of the term "static analysis" varies in different sources. For the purpose of this document, we will understand it as an approach to analysing properties of code (primarily source code, but one can use static analysis over executable binary code too) without executing it at all or at least without executing it under its original semantics. Note that static analysis may produce results applicable for verification but also many other purposes – e.g., optimisation, generation of target code, automated parallelization, code understanding, worst-case execution time analysis, performance (or, more generally, resource consumption) analysis, etc. Given the broad meaning of static analysis, it covers approaches that range from purely syntactic ones (e.g., even grep can be used as a static analyser in an extreme case) to much more semantic approaches that execute the given system under some alternative (typically more abstract) semantics. The approaches of model checking and deductive verification can be viewed as forms of static analysis too though they are often considered standalone approaches (which is adopted in this document too). If we do not count possible extreme interpretations of static analysis, the probably most common approaches to static analysis include data-flow analysis, abstract interpretation, symbolic execution, type and effect analysis, constraint-based static analysis, error-pattern-driven analysis, as well as various pointer analyses. Describing all these approaches in detail is of course beyond the scope of this document, and so we provide their very basic characterization only (inspired by the overview paper [SAN1]). Given a pre-defined set of properties of program states, which are of interest for some particular reason and which can be denoted as the so-called data-flow facts, a *data-flow analysis* tracks how the data-flow facts propagate through a program, usually encoded as a control-flow graph (CFG). If the CFG of a program is not explicit in the program code, it may be derived by some preceding static control-flow analysis. The propagation of data-flow facts is tracked in a way consistent with all feasible paths through the CFG, but without directly executing the program. The most common approach to data-flow analysis is the lattice-theoretic iterative data-flow analysis, which essentially solves a system of data-flow equations derived from the CFG of the program being analysed [SAN2, SAN3, SAN4, SAN5, SAN6]. These equations describe how data-flow facts propagate forward or backward through program statements and how they meet at program junctions.

Abstract interpretation [SAN7, SAN8, SAN9] is a theory of sound approximation of the semantics of computer programs that consists in giving a class of programs a concrete and abstract semantics defined on suitable concrete and abstract lattice-based domains. These domains are usually linked by a pair of monotone functions – the so-called abstraction and concretisation that form a Galois connection (though this requirement can be lifted, and the analysis defined in terms of one of these functions only). Program statements are modelled as monotone functions, often called as concrete and abstract transformers, on the concrete and abstract domains, respectively.
Symbolic execution [SAN10, SAN11, SAN12, SAN13, SAN14, SAN15] encodes runs through a program over systematically enumerated program paths as formulae over a suitable logic theory. When doing so, program inputs are represented symbolically as variables whose value is gradually restricted according to the conditions that the chosen program path contains. Automatic decision procedures, implemented, e.g., in SMT solvers, are used to check satisfiability of the obtained formulae.
Type and effect systems [SAN16, SAN17] extend the basic type systems of programming languages to take into account various semantic effects of the supported programming constructions. One can, for instance, track how the memory is accessed (reading, writing, allocation, de-allocation), whether and how synchronization is used (locking and unlocking of mutexes), whether and how various other resources are used (such as reading and writing of files), whether some exceptions can be generated, etc.
In *constraint-based static analysis*, a set of constraints is derived from the analysed program such that when the constraints are solved, the solution provides the needed information about the program [SAN18, SAN19]. Various kinds of constraints can be used such as conditional set-constraints, linear arithmetic constraints, polynomial arithmetic constraints, etc.
Static analyses based on *error patterns* use some syntactic characterisation of various classes of errors. First tools based on this approach appeared already in the late 70's in the Lint tool. Modern versions of "linters" are still in use and development for numerous languages. However, in more advanced tools, usage of error patterns is often combined with various other static analyses: An error pattern provides some basic characterisation of an undesirable control flow in a program, but then results of, e.g., data-flow analysis, abstract interpretation, various pointer analyses, and/or symbolic execution are used to rule out infeasible control-flow paths in order to reduce the number of false alarms [SAN20, SAN21]. Moreover, all these techniques can also be complemented by using further techniques, such as code slicing to remove parts of code irrelevant when verifying a certain property of the code [SAN22].
Pointer analyses may have various forms such as alias analysis or shape analysis. These analyses are often used as auxiliary analyses for other kinds of analyses. They may be implemented as specific forms of data-flow analysis or abstract interpretation, but specialised algorithms for them have been proposed too [SAN23, SAN24, SAN25].
As one can see, the field of static analysis is indeed broad. Moreover, as already indicated above, it is often the case that several approaches to static analysis are combined together. Since static analyses often over-approximate the behaviour of programs (though they can also under-approximate it or combine over- and under-approximation), they can produce a number of false alarms. To keep this number low, the described techniques or their combinations are often combined with further heuristics (such as statistical reasoning about which warnings are likely to be real errors, baselining

– i.e., ignoring results of the first analysis run and reporting warnings that appear in later analysis runs only, or combinations of static and dynamic approaches).
Relationship with other methods: Static analysis is an alternative to dynamic analysis, *Model checking*, or *Deductive verification*; of course, combinations of such approaches are possible too.
Tool support: Numerous free as well as commercial static analysers exist – the following is just a small sample: Coverity Static Analysis (https://scan.coverity.com/), SpotBugs (https://spotbugs.github.io), KlockWork Static Code Analysis (https://www.perforce.com/products/klocwork), GrammaTech CodeSonar (https://www.grammatech.com/codesonar-cc), Frama-C (https://frama-c.com/), Facebook Infer (https://fbinfer.com/), AbsInt (https://www.absint.com/), PolySpace (https://www.mathworks.com/products/polyspace.html), cppcheck (http://cppcheck.sourceforge.net/), cppclean (https://pypi.org/project/cppclean/), Sparse (https://sparse.docs.kernel.org/en/latest/), Klee (https://klee.github.io/), Symbiotic (https://github.com/staticafi/symbiotic), etc. Some of the tools (such as Frama-C or Facebook Infer) provide open frameworks for defining specific analysis plugins. Moreover, various compilers or IDEs (gcc, Clang, Visual Studio, …) include tools for static analysis too. Project partners are involved in the development of some these analysers or their plugins (e.g., BUT is developing plugins or subsystems of Facebook Infer, Frama-C, and Symbiotic).
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Analytical – Formal, Analytical – Semi-formal • Type of component under evaluation: Software • Evaluation tool: Open source, Proprietary • Evaluation stage: Verification, Validation • Logic of the component under evaluation: Thinking, Acting • Type of requirements under evaluation: Functional, Non-functional - Safety, Non-functional - Cybersecurity, Non-functional - Privacy, Non-functional - Others • Evaluation performance indicator: SCP criteria, V&V process criteria
Use case scenarios • VALU3S_WP1_Automotive_1 - Radar/camera advanced detection and tracking • VALU3S_WP1_Automotive_2 - Radar + camera cooperation • VALU3S_WP1_Automotive_3 - Node connection to cloud • VALU3S_WP1_Automotive_6 - Transmission line switching • VALU3S_WP1_Automotive_7 - Safety of vehicle during switch between routers • VALU3S_WP1_Automotive_8 - Automatic Emergency Braking (AEB)
Strengths • Lightweight (more syntactic) static analysis is highly scalable. Heavier-weight (more semantic) static analysis may scale less but often still significantly better than other approaches. • Some forms of static analysis are applicable even on code fragments without any need for the fragment to even compile, not speaking about a need to run it. • Some forms of static analysis are sound (i.e., provide formal guarantees of the correctness of the provided answers) though soundness is often sacrificed to scalability and reducing the number of false alarms.

Limitations

- Highly scalable static analysis approaches often come with a number of false alarms (or the number of such alarms is artificially reduced by accepting unsoundness of the approach, i.e., allowing the approach to miss real errors).
- Some heavier-weight approaches may require the system under analysis to compile and even to come with a test harness.
- Efficient and precise static analysers are often fine-tuned for a specific class of programs and properties. Their event slight change may require the analysis to be reworked.
- Some forms of static analysis are not applicable for some classes of errors or they are applicable in a limited way only – e.g., it is difficult to cover errors such as data races in a satisfactorily exhaustive way by purely syntactic error patterns.

References

- [SAN1] Krena, B., Vojnar, T.: Automated Formal Analysis and Verification: An Overview. International Journal of General Systems, 42(4), 2013.
- [SAN2] Kildall, G.A.: A Unified Approach to Global Program Optimization. ACM Press, 1973.
- [SAN3] Kam, J.B., Ullman, J.D.: Global Data Flow Analysis and Iterative Algorithms. Journal of the ACM, 23, 1976.
- [SAN4] Kam, J.B., Ullman, J.D.: Monotone Data Flow Analysis Frameworks. Acta Informatica, 7, 1977.
- [SAN5] Khedker, U.P., Dhamdhere, D.M.: A Generalized Theory of Bit Vector Data Flow Analysis. ACM Transactions on Programming Languages and Systems (TOPLAS), 16(5), 1994.
- [SAN6] Khedker, U., Sanyal, A., Sathe, B.: Data Flow Analysis: Theory and Practice, CRC Press, 2009.
- [SAN7] Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. Proc. of POPL'77, ACM, 1977.
- [SAN8] Cousot, P., Cousot, R.: Abstract Interpretation Frameworks. Journal of Logic and Computation, 2(4), 1992.
- [SAN9] Rival, X., Yi, K.: Introduction to Static Analysis. An Abstract Interpretation Perspective. MIT Press, 2020.
- [SAN10] Boyer, R.S., Elspas, B., Levitt, K.N.: SELECT – A Formal System for Testing and Debugging Programs by Symbolic Execution. Proc. of Int. Conf. on Reliable Software, ACM, 1975.
- [SAN11] Howden, W.E.: Symbolic Testing and the DISSECT Symbolic Evaluation System. IEEE Trans. on Software Engineering, 3(4), 1977.
- [SAN12] King, J.C.: A New Approach to Program Testing. Proc. Int. Conf. on Reliable Software, ACM, 1975.
- [SAN13] King, J.C.: Symbolic Execution and Program Testing. CACM, 19(7), ACM, 1976.
- [SAN14] Cadar, C., Dunbar, D., Engler, D.R.: KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. Proc. of OSDI'08, USENIX, 2008.
- [SAN15] Baldoni, R., Coppa, E., Cono D'elia, D., Demetrescu, C., Finocchi, I.: A Survey of Symbolic Execution Techniques. ACM Computing Surveys, 50, ACM, 2018.
- [SAN16] Nielson, F., Nielson, H.R., 1999. Type and Effect Systems. Correct System Design, Recent Insight and Advances, LNCS 1710, Springer, 1999.

• [SAN17] Palsberg, J., Millstein, T.D.: Type Systems: Advances and Applications. The Compiler Design Handbook, 2007. • [SAN18] Aiken, A.: Introduction to Set Constraint-based Program Analysis. Science of Computer Programming, 35(2-3), 1999. • [SAN19] Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis, Springer-Verlag, 2005. • [SAN20] Hallem, S., Chelf, B., Xie, Y., Engler, D.R.: A System and Language for Building System-Specific, Static Analyses. Proc. of PLDI'02, ACM, 2002. • [SAN21] Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Gros, C.-H., Kamsky, A., McPeak, S., Engler, D.R.: A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. CACM 53(2), ACM, 2010. • [SAN22] Slaby, J., Strejcek, J., Trtik, M.: Symbiotic: Synergy of Instrumentation, Slicing, and Symbolic Execution. Proc. of TACAS'13, LNCS 7795, Springer, 2013. • [SAN23] Andersen, L.O.: Program Analysis and Specialization for the C Programming Language. PhD thesis, University of Copenhage, 1994. • [SAN24] Steensgaard, B.: Points-to Analysis in Almost Linear Time. Proc. of POPL'96, ACM, 1996. • [SAN25] Smaragdakis, Y., Balatsouras, G.: Pointer Analysis. Foundations and Trends in Programming Languages, 2(1), 2015.
Related standards: MISRA C/C++, ISO 26262, DO-178B/C, IEC 61508
Keywords: Static analysis, Data-flow analysis, Control-flow analysis, Abstract interpretation, Symbolic execution, Type and effect systems, constraint-based static analysis, Error patterns, Pointer analysis.

3.5.2 General Formal Verification

This sub-group of methods focuses on mathematically proving properties of different system artefact types. They use formal methods to specify and analyse the system behaviour. A core method in this group is model checking, which is used to algorithmically verify properties of the behavioural model. Section 3.5.2.3 gives an overview of the variety of model checking techniques that exist for the various combinations of formal languages that can be used to specify the system behaviour and its properties.

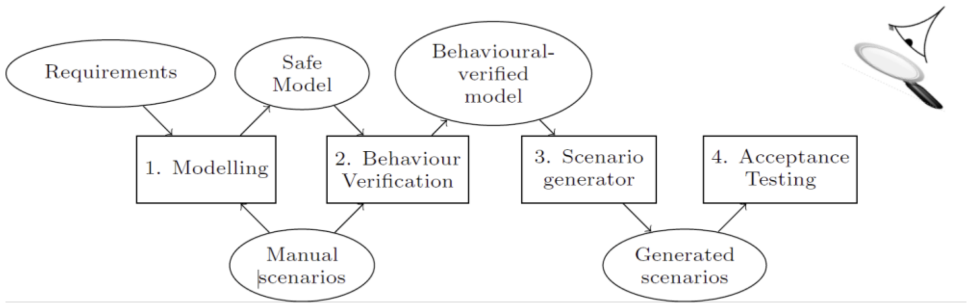
Developing formal models usually requires advanced skills in formal methods. Proving safety requirements with model checking may be not sufficient as the model may be too strict forbidding required operation behaviour. *Behaviour-driven formal model development* describes a development flow where scenarios are used to derive and validate the formal model. This is also verified with model checking and used to derive test cases for the final implementation.

Similar validation problems exist for the specification of properties that formalize the system requirements and that are used for model checking. Errors in the properties specification may be severe issues that hamper the subsequent verification process. *Formal requirements validation* and *Model checking* gives an overview of the related property-based analysis techniques.

Reachability-analysis-based verification for safety-critical hybrid systems goes more in depth into the model checking problem of reachability properties for hybrid systems, which is of particular relevance for the aerospace use case scenarios. Even if we focus only on reachability properties (that are simpler than more general temporal properties), the continuous dynamics typically described with ordinary differential equation or differential algebraic equations and their interplay with discrete mode changes make the exhaustive state space analysis quite challenging for state-of-the-art techniques.

Finally, *Theorem proving and SMT solving* focuses on mathematical reasoning on the correctness of system properties, typically safety and liveness properties

3.5.2.1 Behaviour-Driven Formal Model Development

Name: **Behaviour-driven formal model development**
Purpose: Develop formal models of safety-critical functionalities that provenly fulfil safety requirements without hampering required operational behaviour. Tests that are derived in the process can be used to verify that an implementation conforms to the formal model.
Description: Behaviour-Driven Formal Model Development aims to address the issue that domain expertise and formal modelling are often not in the skill set of a single person. The process is shown in Figure 3.7. The approach uses scenarios in the sense of Behaviour Driven Development [BFM1, BFM2] to allow the domain expert to specify examples (Manual Scenarios) for the expected operation of the system. Based on the Requirements and the Manual Scenarios, the formal methods expert builds a Safe Model that covers both safety and functional requirements. Using formal methods like refinement proofs and model checking, safety properties of the model are guaranteed. In the next step, Behaviour Verification, the Manual Scenarios are run against the model, thereby ensuring that the Behavioural Verified Model is not only safe but also does what it is expected to do. Next, the scenarios' coverage of the model is evaluated and additional scenarios to achieve higher coverage are generated. The Generated Scenarios are then reviewed by the domain expert in an Acceptance Testing step by indirectly reviewing any additional behaviour included in the model. The combined set of scenarios can later contribute to ensuring that a specific implementation complies with the model. The concept of Behaviour-Driven Formal Model Development has been introduced in [BFM3] and applied to a new standard for railway operations in [BFM4].  Requirements Safe Model Behavioural-verified model 1. Modelling 2. Behaviour Verification 3. Scenario generator 4. Acceptance Testing Manual scenarios Generated scenarios *Figure 3.7 Steps of Behaviour-Driven Formal Model Development*
Relationship with other methods: *Model-based testing, Model-based mutation testing.*

Tool support: Integration of Rodin (http://www.event-b.org/), Gherkin for Event-B (not released), Pro-B (https://www3.hhu.de/stups/prob/index.php/Main_Page), and MoMuT::Event-B (https://momut.org)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab, Closed (use of resulting scenarios)
- Evaluation type: Experimental – Testing, Analytical – Formal
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Thinking, Sensing, Acting
- Type of requirements under evaluation: Functional, Non-functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Railway_1 - Inject, detect and recover
- VALU3S_WP1_Railway_2 - Controlled vs random injection
- VALU3S_WP1_Railway_3 - Systematic and random failures verification

Strengths

- Allows for verified (functional) safety without neglecting operational soundness/expected functionality.
- The process allows to split work between different skill sets. Domain expert and formal methods expert use scenarios as one means of communication. The domain expert does not need to understand the formal method to validate the formal model.
- Manually derived and automatically generated scenarios can be used as test cases for the implementation, to ensure that the implementation adheres to the formal model.
- Problems found in the implementation through testing can be traced back to top-level and derived requirements.

Limitations

- Fits best to event-triggered behaviour, less to continuous or data-driven functionalities.

References

- [BFM1] North, D.: Introducing BDD. Better Software Magazine (Mar 2006)
- [BFM2] Smart, J.F.: BDD in Action: Behavior-Driven Development for the Whole Software Life cycle. Manning Publications Company (2014)
- [BFM3] Snook C. et al. (2018) Behaviour-Driven Formal Model Development. In: Sun J., Sun M. (eds) Formal Methods and Software Engineering. ICFEM 2018. Lecture Notes in Computer Science, vol 11232. Springer, Cham. https://doi.org/10.1007/978-3-030-02450-5_2
- [BFM4] M. Butler *et al.*, "Behaviour-Driven Formal Model Development of the ETCS Hybrid Level 3," *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, Guangzhou, China, 2019, pp. 97-106, doi: 10.1109/ICECCS.2019.00018.

Related standards: Addresses formal proof of safety requirements as recommended or required by several safety standards (IEC61508, 50128). Further uses model-based testing which is highly recommended for SIL3 and SIL4 in IEC 61508.

Keywords: Behaviour-Driven Development, Test-Driven Development, Formal methods.

3.5.2.2 Formal Requirements Validation

Name: **Formal requirements validation**
Purpose: Validate the specification of formal requirements in terms of consistency, compatibility with scenarios, vacuity, realizability, and other formal checks.
Description: Flaws in requirements may have severe impacts on the subsequent phases of the development flow. These also undermine the formal verification activity, which typically formalizes the requirements into formal properties that are considered as golden and used to check the correctness of the design or software/hardware implementation. Most of the efforts in formal methods have historically been devoted to comparing a design against a set of requirements. The validation of the requirements themselves, however, has often been disregarded and poses several challenges. First, requirements are often written in natural language, and may thus contain a high degree of ambiguity. Second, the informal requirements often express global constraints on the system-to-be (e.g., mutual exclusion), and, in order to retain a direct connection with the informal requirements, the formalization cannot follow standard model-based approaches, but must be complemented with more suitable formalisms such as temporal logics. Third, the formal validation of requirements suffers from the lack of a clear correctness criterion (which in the case of design verification is basically given by the availability of high-level properties). Finally, the expressiveness of the language used in the formalization may go beyond the typical level used for formal verification (e.g., propositional logic). The methodology for requirements analysis proposed in the PROSYD project [FRV1, FRV2] is based on a series of checks in terms of satisfiability of LTL to verify the consistency of properties, their compatibility with some scenario, and their entailment of some assertions. This work has been extended to more expressive logic to represent requirements of hybrid systems [FRV3, FRV4] and to temporal satisfiability modulo theories [FRV5]. Other formal checks that can be used to pinpoint issues in the requirements specification are based on the notions of realizability [FRV6], vacuity [FRV7], or unconstrained outputs [FRV8].
Relationship with other methods: *Model checking*.
Tool support: nuXmv (https://nuxmv.fbk.eu), OCRA (https://ocra.fbk.eu)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Analytical – Formal • Type of component under evaluation: Model • Evaluation tool: Open Source, Proprietary • Evaluation stage: Validation • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Functional, Non-Functional – Safety • Evaluation performance indicator: V&V process criteria, SCP criteria
Use case scenarios • VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults • VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation • VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events • VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths

- The formal validation helps in identifying issues in the requirements specification
- Detecting issues at the requirements level in the early phases of the design may yield a huge saving in terms of costs

Limitations

- The formalization requires effort and expertise in interpreting the results

References

- [FRV1] Ingo Pill, Simone Semprini, Roberto Cavada, Marco Roveri, Roderick Bloem, Alessandro Cimatti: Formal analysis of hardware requirements. DAC 2006: 821-826
- [FRV2] Roderick Bloem, Roberto Cavada, Ingo Pill, Marco Roveri, Andrei Tchaltsev: RAT: A Tool for the Formal Analysis of Requirements. CAV 2007: 263-267
- [FRV3] Alessandro Cimatti, Marco Roveri, Stefano Tonetta: Requirements Validation for Hybrid Systems. CAV 2009: 188-203 Requirements Validation for Hybrid Systems. CAV 2009: 188-203
- [FRV4] Alessandro Cimatti, Marco Roveri, Angelo Susi, Stefano Tonetta: Validation of requirements for hybrid systems: A formal approach. ACM Trans. Softw. Eng. Methodol. 21(4): 22:1-22:34 (2012)
- [FRV5] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, Stefano Tonetta: SMT-based satisfiability of first-order LTL with event freezing functions and metric operators. Inf. Comput. 272: 104502 (2020)
- [FRV6] Amir Pnueli, Roni Rosner: On the Synthesis of a Reactive Module. POPL 1989: 179-190
- [FRV7] Dana Fisman, Orna Kupferman, Sarai Sheinvald-Faragy, Moshe Y. Vardi: A Framework for Inherent Vacuity. Haifa Verification Conference 2008: 7-22
- [FRV8] Koen Claessen: A Coverage Analysis for Safety Property Lists. FMCAD 2007: 139-145

Related standards: -

Keywords: Formal requirements validation, Formal methods, Formal specification, Requirements engineering.

3.5.2.3 Model Checking

Name: **Model checking**

Purpose: To verify if a model of a system satisfies a property

Description: Model checking [MCH1, MCH2, MCH3] is a method to verify if a model of the system under verification satisfies its specification. Many languages to write the model exist and range from finite-state to infinite-state machines, from discrete-time to timed or hybrid systems, from non-deterministic automata to stochastic models, from synchronous to asynchronous communicating programs. Given a formal semantics of the input language, model checking can also be applied to models defined for other purposes (architectural description or simulation) or directly to software or hardware source code.

Also, for the property specification, there is a wide range of options, ranging from simple reachability or invariant properties, to temporal properties, from safety to liveness properties. Depending on the modelling language, temporal properties can be specified in various logics, either propositional or first-order, discrete or continuous or hybrid time, linear or branching or probabilistic or hyper-properties logic.

The model checking problem is solved algorithmically by a procedure that decides if the model satisfies the property or finds a counterexample that shows how the model violates it. When the problem is undecidable (as for example for software), the model checking procedure may be incomplete.

Explicit-state model checkers (such as Spin [MCH4]) prove the property by enumerating the reachable states of the model. Symbolic model checkers [MCH5] (such as nuXmv [MCH6]) represent states and transition symbolically and perform the search by means of logical operations. Modern model checkers combine various techniques, integrating explicit-state search with deductive and abstraction refinement techniques.

Automated abstraction [MCH7], in particular, *predicate abstraction* [MCH8, MCH9], refined according to the encountered false counterexamples to the property being verified using the so-called *CEGAR loop* [MCH10], is successful especially in software verification.

Approaches based on *bounded model checking* [MCH11, MCH12, MCH13] explore the state space up to a certain bound. Although they do not provide sound results in general to prove universal properties, they are very successful in practice in finding counterexamples. In case of symbolic model checking, they typically exploit the efficiency of the underlying SAT or SMT solvers.

Other effective and efficient SAT-based model checking techniques are based on generalization of induction such as *k-induction* [MCH14] and *IC3* [MCH15], and their integration with implicit predicate abstraction [MCH16, MCH17].

Various competitions exist to evaluate and compare model checkers on different kinds of benchmarks, with focus on hardware [MCH18], software [MCH19], or other logical formats such as horn clauses [MCH20].

Relationship with other methods: *Formal verification*, *Deductive verification*, *Source code static analysis*.

Tool support: Examples include
Spin (http://spinroot.com/), NuSMV (http://nusmv.fbk.eu/, nuXmv (https://nuxmv.fbk.eu/), Uppaal (http://www.uppaal.org/), mCRL2 (https://www.mcrl2.org/), TLA+ (https://lamport.azurewebsites.net/tla/tla.html), CADP (https://cadp.inria.fr/), DIVINE (http://divine.fi.muni.cz/), Dfinder (http://www-verimag.imag.fr/DFinder.html), ProB (https://www3.hhu.de/stups/prob/), SAL (http://sal.csl.sri.com/, Phaver (http://www-verimag.imag.fr/~frehse/phaver_web/), SpaceX (http://spaceex.imag.fr/), Model Checker for Multi-Agent Systems, CoCoSim (https://coco-team.github.io/cocosim/), FDR (https://cocotec.io/fdr/), CPAchecker (http://cpachecker.sosy-lab.org/), Ultimate Automizer (https://ultimate.informatik.uni-freiburg.de/automizer/), CBMC (www.cprover.org › cbmc), JBMC (www.cprover.org › jbmc) , ESBMC (http://www.esbmc.org/), 2L, VeriAbs and many more

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical - Formal
- Type of component under evaluation: Model, Software
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths

- Automatic - making model checkers relatively easy to use
- Can check for specific properties without verifying the full system
- Exhaustively explores the state space
- Well-established verification tools and techniques with industrial applications

Limitations

- Exhaustively explores the state space so careful consideration has to be taken regarding the inputs (models and properties to be checked)
- The size of the state space is often exponential in the number of variables and the number of components of the system which execute in parallel.

References

- [MCH1] Edmund M. Clarke, Orna Grumberg, Doron A. Peled: Model checking. MIT Press 1999, ISBN 978-0-262-03270-4
- [MCH2] Christel Baier, Joost-Pieter Katoen: Principles of model checking. MIT Press 2008, ISBN 978-0-262-02649-9, pp. I-XVII, 1-975
- [MCH3] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, Roderick Bloem: Handbook of Model Checking. Springer 2018, ISBN 978-3-319-10574-1
- [MCH4] G. J. Holzmann: The SPIN Model Checker - primer and reference manual. Addison-Wesley 2004, ISBN 978-0-321-22862-8, pp. I-XII, 1-596
- [MCH5] Kenneth L. McMillan: Symbolic model checking. Kluwer 1993, ISBN 978-0-7923-9380-1, pp. I-XV, 1-194
- [MCH6] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, Stefano Tonetta: The nuXmv Symbolic Model Checker. CAV 2014: 334-342
- [MCH7] E. Clarke, O. Grumberg, and D. Long. Model Checking and Abstraction. Trans. Program. Lang. Syst., 16(5):1512–1542, 1994.
- [MCH8] Graf, S., Saidi, H.: Construction of Abstract State Graphs with PVS. Proc. of CAV'97, LNCS 1254, Springer, 1997.
- [MCH9] Flanagan, C., Qadeer, S.: Predicate Abstraction for Software Verification. Proc. of POPL'02, ACM, 2002.
- [MCH10] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. J. ACM, 50(5):752–794, 2003.
- [MCH11] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu: Symbolic Model Checking without BDDs. TACAS 1999: 193-207
- [MCH12] Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded Model Checking Using Satisfiability Solving. Formal Methods Syst. Des. 19(1), 2001.
- [MCH13] Abhishek Udupa, Ankush Desai, Sriram K. Rajamani: Depth Bounded Explicit-State Model Checking. SPIN 2011: 57-74

• [MCH14] Sheeran, M., Singh, S., Stalmarck, G.: Checking Safety Properties Using Induction and a SAT-solver. Proc. of FMCAD'00, LNCS 1954, Springer, 2000. • [MCH15] Bradley, A.R.: SAT-Based Model Checking without Unrolling. Proc. of VMCAI'11, LNCS 6538, Springer, 2011. • [MCH16] Stefano Tonetta: Abstract Model Checking without Computing the Abstraction. FM 2009: 89-105 • [MCH17] A. Cimatti, A. Griggio, S. Mover, S. Tonetta: IC3 Modulo Theories via Implicit Predicate Abstraction. TACAS 2014: 46-61 • [MCH18] Hardware Model Checking Competition 2020: http://fmv.jku.at/hwmcc20/ • [MCH19] International Competition in Software Verification SV-COMP (https://sv-comp.sosy-lab.org/). • [MCH20] P. Rümmer. Competition report: CHC-COMP 2020. https://arxiv.org/abs/2008.02939
Related standards: Related to standards that require verification of safety conditions include DO-178C, DO-278A, ISO 26262, ISO 21448, and IEEE P2846.
Keywords: Model checking, Formal verification, Formal specification, Proof generation.

3.5.2.4 Reachability-Analysis-Based Verification for Safety-Critical Hybrid Systems

Name: **Reachability-analysis-based verification for safety-critical hybrid systems**
Purpose: Exhaustive exploration of system evolution over time, given an initial input range
Description: Hybrid system reachability analysis provides an exhaustive verification alternative to standard simulation-based testing (e.g. Monte-Carlo simulation). Given an appropriate description of the system and a set of inputs, the system evolution over a finite time horizon is computed as a set of reachable states in a specific representation. The representation of reachable states varies among algorithms that solve the problem, and the choices here typically offer various trade-offs between performance and accuracy – note that exact computation of reachable states is an undecidable problem in the general case, therefore what the algorithms typically compute here is an (over)approximation.
Relationship with other methods: *Model checking*, *Theorem Proving and SMT Solving*.
Tool support: HyComp, Flow*, SpaceEx
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Analytical-Formal • Type of component under evaluation: Software, Model • Evaluation tool: Open source, Proprietary • Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Functional, Non-functional - Safety • Evaluation performance indicator: V&V process criteria
Use case scenarios • VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults • VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation • VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events

• VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery
Strengths • Automation • Exhaustive coverage • Sound analysis
Limitations • Scalability • False negatives
References • [RAV1] V. A. Tsachouridis, G. Giantamidis, S. Basagiannis, K. Kouramas, "Formal analysis of the Schulz matrix inversion algorithm: A paradigm towards computer aided verification of general matrix flow solvers" in journal Numerical Algebra, Control & Optimization of American Institute of Mathematical Sciences, vol. 10, 2020, doi: 10.3934/naco.2019047. • [RAV2] V. A. Tsachouridis and G. Giantamidis, "Computer-aided verification of matrix Riccati algorithms*", 2019 IEEE 58th Conference on Decision and Control (CDC), Nice, France, 2019, pp. 8073-8078, doi: 10.1109/CDC40024.2019.9030135. • [RAV3] Cimatti A., Griggio A., Mover S., Tonetta S. (2015) HyComp: An SMT-Based Model Checker for Hybrid Systems. In: Baier C., Tinelli C. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2015. Lecture Notes in Computer Science, vol 9035. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-46681-0_4. • [RAV4] Chen X., Ábrahám E., Sankaranarayanan S. (2013) Flow*: An Analyzer for Non-linear Hybrid Systems. In: Sharygina N., Veith H. (eds) Computer Aided Verification. CAV 2013. Lecture Notes in Computer Science, vol 8044. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39799-8_18. • [RAV5] Frehse G. et al. (2011) SpaceEx: Scalable Verification of Hybrid Systems. In: Gopalakrishnan G., Qadeer S. (eds) Computer Aided Verification. CAV 2011. Lecture Notes in Computer Science, vol 6806. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22110-1_30.
Related standards: -
Keywords: Reachability analysis, Symbolic computation, Affine arithmetic, Interval analysis, SMT (satisfiability modulo theories), Zonotopes, Polytopes, Taylor models, Support functions, Star sets.

3.5.2.5 Theorem Proving and SMT Solving

Name: **Theorem proving and SMT solving**
Purpose: Provides mathematical reasoning on the correctness of system properties.
Description: Produces a formal proof of the correctness of a software system which can be used to provide robust evidence for certification of software systems. Properties verified are typically safety and liveness properties. Theorem provers may be automatic or interactive, with proof tactics and libraries provided to assist the users. Theorem Provers and SMT solvers are used for verification, proving the correctness of programs, software testing based on symbolic execution, and for synthesis, generating program fragments by searching over the space of possible programs. Theorem provers like Coq, PVS and

Isabelle/HOL are proof management systems that provide a formal language to write mathematical formulae and tools for proving these formulae in a logical calculus. Different theorem provers support proofs in different logics, e.g., Keymeara X supports proofs for differential dynamic logic, a logic for specifying and verifying properties of hybrid systems with mixed discrete and continuous dynamics whereas the STeP theorem prover is concerned with invariant conditions. SMT solvers like Z3 support proof strategies through providing theories of arithmetic, fixed-size bit-vectors, extensional arrays, datatypes, uninterpreted functions, and quantifiers which facilitate reasoning. Applications are primarily in extended static checking, test case generation, deductive verification, and predicate abstraction.

Relationship with other methods: Theorem provers and SMT solvers are typically used as a component of verification tools (e.g., for *Model checking* or *Deductive verification*). Many of these verififcation tools offer a portfolio of theorem provers and SMT solvers to work on the proofs.

Tool support: Tools include Coq (https://coq.inria.fr/), PVS (https://pvs.csl.sri.com/), Isabelle/HOL (http://isabelle.in.tum.de/), KeyMeara X (http://www.ls.cs.cmu.edu/KeYmaeraX/), Z3 (https://github.com/Z3Prover/z3), Lean (https://leanprover.github.io/), CVC4 (http://cvc4.github.io/), Alt-Ergo (http://alt-ergo.ocamlpro.com/) and many more

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical - Formal
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults
- VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation
- VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events
- VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery

Strengths

- Provides a formal proof of the system correctness which can be used for robust evidence for certification
- Tools provide many proof tactics provided to automate proofs
- Approaches available for discrete and continuous dynamics.

Limitations

- Often require interaction when proof tactics fail
- Specialist knowledge required
- Since provers operate on a formal model, there needs to be a separate step to show that the formal model properties actually represent the system properties

References

• [TPS1] KeYmaera: a hybrid theorem prover for hybrid systems (system description) A. Platzer and J.-D. Quesel, Automated Reasoning, Springer, Berlin (2008), pp. 171-178 • [TPS2] Z. Manna, N. Bjoerner, A. Browne, and E. Chang, STeP: The Stanford Temporal Prover, LNCS, Vol. 915, 1995, pp. 793–794. • [TPS3] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, and T. Nipkow. Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL. volume 10510 of LNCS, pages 50–66. Springer, 2017. • [TPS4] Deductive verification of hybrid systems using step, Z. Manna and H.B. Sipma Hybrid Systems: Computation and Control, Springer, Berlin (1998), pp. 305-318 • [TPS5] P. Bagade, A. Banerjee, S.K.S. Gupta, Chapter 12 - Validation, Verification, and Formal Methods for Cyber-Physical Systems, In Intelligent Data-Centric Systems, Cyber-Physical Systems, Academic Press, 2017 • [TPS6] A Survey on Theorem Provers in Formal Methods M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun and M. Ikram Ullah Lali, https://arxiv.org/pdf/1912.03028.pdf
Related standards: Related to standards that required verification of safety conditions e.g. DO-178C, DO-278A, ISO 26262 and ISO 21448
Keywords: Theorem proving, Automated, Interactive, Verification, Extended static checking, Test case generation, Predicate abstraction.

3.6 Semi-Formal Analysis

This group of methods deals with system evaluation by using structured means whose application does not result in a mathematical proof. The methods have been divided into two sub-groups: one for SCP-focused semi-formal analysis and another for general semi-formal analysis.

3.6.1 SCP-Focused Semi-Formal Analysis

This sub-group of methods focuses on semi-formally evaluating SCP-specific characteristics of a system, e.g. on developing confidence in system dependability in relation with SCP-specific characteristics of a system such as faults, vulnerabilities, threats, intrusion, and contribute to their avoidance, identification and recovery.

Semi-Formal Methods are formalisms and languages that are not considered fully “formal”. As a mathematically rigorous approach to the SCP V&V of complex systems is unfeasible in many cases, semi-formal techniques are developed to complement formal V&V. System decomposition, abstraction, and specific models reduce SCP V&V to sub-problems of limited scope that may be addressed using semi-formal methods and tools, which can rely on models, architectural principles, mathematical or probabilistic calculus, qualitative and quantitative analysis and simulation, while addressing engineering and assurance standards.

The review below of V&V methods for SCP semi-formal analysis are based on different approaches. Several model-based methods are reviewed, applied for safety and threat analysis, contributing to risk analysis, and applied for the overall system, or for specific sub-systems or problems, such as human interaction protocols, wireless sensor networks. Other are data driven methods that aim to complement mathematical models for fault detection. Certain methods focus on specific problems such as cryptographic modules and wireless interfaces.

3.6.1.1 Human Interaction Safety Analysis

Name: **Human interaction safety analysis (HISA)**
Purpose: Find safety issues in human-machine interaction (HMI) protocols. The aim of the method is to enable the HMI design to be improved to reduce safety risks, and the analysis results can also be used as part of a safety case.
Description: This is a safety analysis method which systematically identifies interaction failures between humans and machines. The focus so far has been on analysis of protocols for transition of control of the dynamic driving task between a human driver and an automated driving system. However, the aim is to extend it to other types of interaction as between humans and cyber-physical systems as well as make use of real-world data to improve the analysis results [HIS1]. The process consists of the steps: (1) propose a communication protocol; (2) create the interaction sequence between a HU (Human User) and a machine as two communicating entities through the HMI, considering the possible combinations of time intervals (Figure 3.8, top left where the machine is an ADS – automated driving system); (3) perform cause-consequence analysis (CCA) by constructing cause-consequence diagrams (CCD) based on the interaction sequences (Figure 3.8, top right), and for each failed event on the CCD perform a fault tree analysis (FTA) considering a model of human behaviour (Figure 3.8, bottom); and lastly (4) perform a risk assessment for identified potential faults and improve the HMI design if the residual risk is considered unacceptable. The results of the analysis should be useful as a part of the argument for safety of the ADS, and thus used in the ADS safety case.

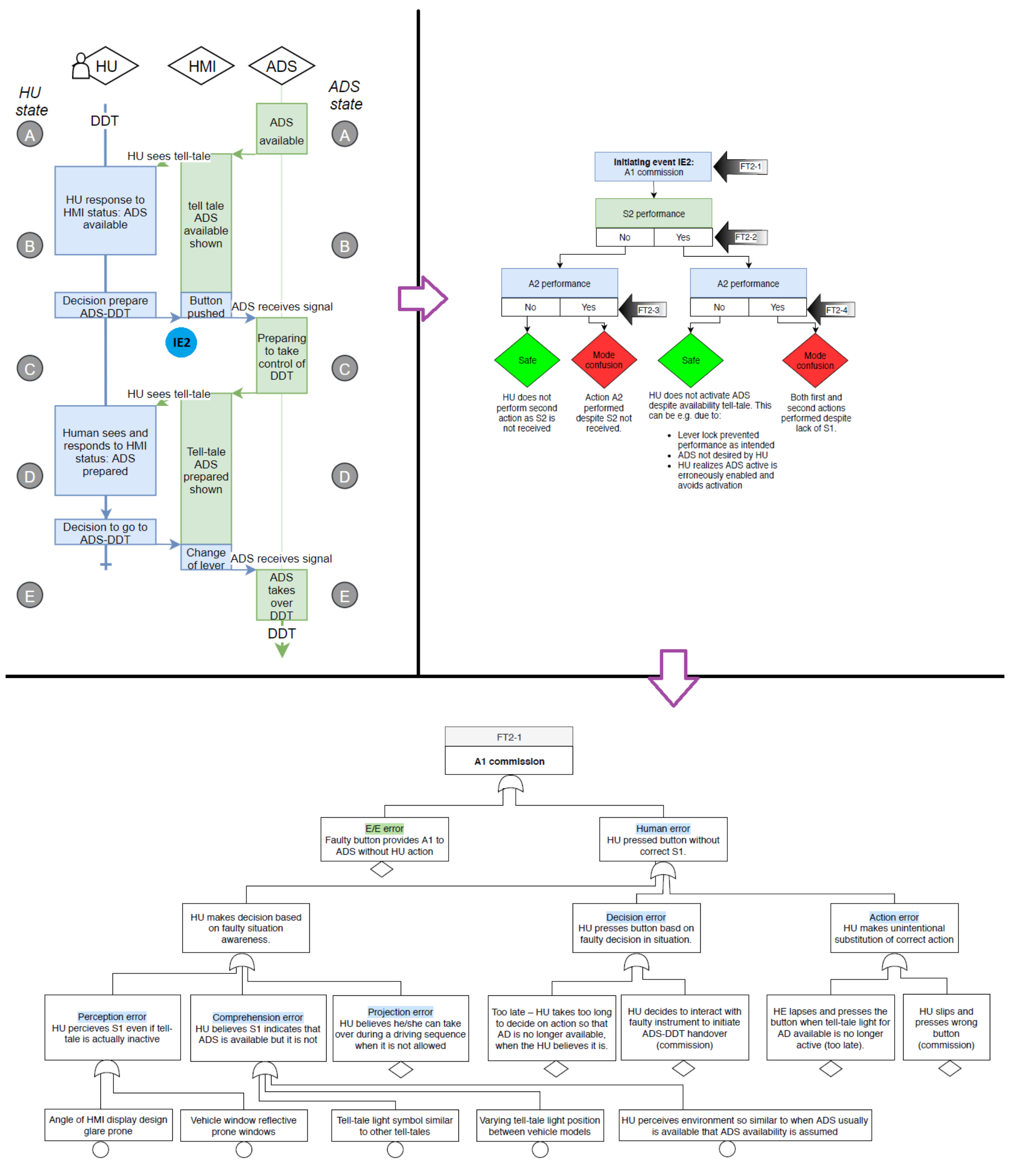

HU
HMI
ADS
HU state
ADS state
DDT
A
ADS available
HU sees tell-tale
HU response to HMI status: ADS available
tell tale ADS available shown
B
Decision prepare ADS-DDT
Button pushed
ADS receives signal
IE2
Preparing to take control of DDT
C
HU sees tell-tale
Human sees and responds to HMI status: ADS prepared
Tell-tale ADS prepared shown
D
Decision to go to ADS-DDT
Change of lever
ADS receives signal
ADS takes over DDT
E
DDT
Initiating event IE2: A1 commission
FT2-1
S2 performance
No
Yes
FT2-2
A2 performance
FT2-3
FT2-4
Safe
Mode confusion
HU does not perform second action as S2 is not received
Action A2 performed despite S2 not received.
HU does not activate ADS despite availability tell-tale. This can be e.g. due to:
• Lever lock prevented performance as intended
• ADS not desired by HU
• HU realizes ADS active is erroneously enabled and avoids activation
Both first and second actions performed despite lack of S1.
FT2-1
A1 commission
E/E error
Faulty button provides A1 to ADS without HU action
Human error
HU pressed button without correct S1.
HU makes decision based on faulty situation awareness.
Decision error
HU presses button basd on faulty decision in situation.
Action error
HU makes unintentional substitution of correct action
Perception error
HU percieves S1 even if tell-tale is actually inactive
Comprehension error
HU believes S1 indicates that ADS is available but it is not
Projection error
HU believes he/she can take over during a driving sequence when it is not allowed
Too late – HU takes too long to decide on action so that AD is no longer available, when the HU believes it is.
HU decides to interact with faulty instrument to initiate ADS-DDT handover (commission)
HE lapses and presses the button when tell-tale light for AD available is no longer active (too late).
HU slips and presses wrong button (commission)
Angle of HMI display design glare prone
Vehicle window reflective prone windows
Tell-tale light symbol similar to other tell-tales
Varying tell-tale light position between vehicle models
HU perceives environment so similar to when ADS usually is available that ADS availability is assumed

Figure 3.8 Illustration of the HISA method

Relationship with other methods: Related with human-machine interaction methods such as: *Simulation-based testing for human-robot collaboration*, and with model-based analysis methods such as: *Model-based safety analysis*. Suitable to be combined with model-based assurance, see: *Model-based assurance and certification*, and *Model-based design verification*.

Tool support: N/A (currently no dedicated tool available)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Semi-formal
- Type of component under evaluation: Software, Hardware, Model
- Evaluation tool: Open source

• Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Non-functional - Safety • Evaluation performance indicator: SCP criteria
Use case scenarios • VALU3S_WP1_Automotive_4 - Transmission line under different performance conditions • VALU3S_WP1_Agriculture_1 - Vehicle switching from parallel guidance to manual mode • VALU3S_WP1_Agriculture_2 - Vehicle switching from manual mode to parallel guidance
Strengths • Enables analysis of HMI frameworks with respect to both Electrical and Electronics (E/E) and human errors (functional safety and human factors expertise is combined). • Provides (analytical) evidence for an ADS safety case.
Limitations • Lack of dedicated tool support (existing tools for e.g. sequence diagrams and FTA can be used). • Low TRL, lack of application to real use cases.
References • [HIS1] Warg, F., Ursing, S., Kaalhus, M. and Wiik, R., 2020, January. Towards Safety Analysis of Interactions Between Human Users and Automated Driving Systems. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*.
Related standards: ISO 26262:2018 (functional safety for road vehicles), ISO PAS 21448:2019 (safety of the intended functionality – includes requirements on ability of the HMI to prevent reasonably foreseeable misuse, SAE J3016:2018 (taxonomy and definitions for ADS)
Keywords: Human-machine interaction, Safety analysis, Safety case.

3.6.1.2 Intrusion Detection for WSN based on WPM State Estimation

Name: **Intrusion detection for wireless sensor networks based on WPM state estimation**
Purpose: Detect and provide notifications for intruders and attacks targeting Wireless Sensor Networks.
Description: State-of-the-art solutions for intrusion and attack detection have major drawbacks in resource-constrained platforms such as WSNs. In fact, those solutions require large amount of memory and storage to provide effective and useful intrusion/attack detection. This issue led the design of WIDS *(WSN Intrusion Detection System*) [IDS1], an intrusion detection system specifically designed to overcome the limitations and gain advantages on the peculiarity of WSNs. WIDS models the known attacks into more general *Weak Model Processes (WPM)*, which are a simplification of Hidden Markov Processes where the probability on the edges can be only 1 or 0, i.e., a hidden state can be either *reachable* or *unreachable*. Each node in the WPM represents a possible state in which one WSN node happens to be. The reachability of a state depends on specific conditions on a chosen set of *observable* events which are detected and analysed by WIDS and the underlying networking drivers. WIDS uses such models to estimate the current state of WSN nodes and to detect when such states represent a danger for the WSN node or, for the whole WSN. For example, consider the observable, e.g., “Clear Channel Assessment Failure” (the node was unable to acquire the radio medium) which can lead to a dangerous state, e.g., “WSN node under Jamming”.

The WIDS design has been recently implemented as a component in the TinyOS framework with the name *TinyWIDS* [IDS2]. TinyWIDS embeds WIDS design and provide enhanced radio transceiver drivers to provide low-level observables, the WPM representation as JSON, the continuous state estimation by inspecting selected attacks WPMs and notification system based on TinyOS events.
Currently, TinyWIDS can detect incoming attacks and eventually the source of them, but it cannot provide any reactive behaviour which could be useful to avoid further damage to the WSN before the operators could take action.
In Figure 3.9, an example of WPM is shown. Apart from the initial assumed state (called "reset state") there are five possible states (1-5), each with the set of observables (inside the parenthesis) that enables the transition towards it. The two states with the greyed background are the "dangerous" states. The edges of the WPM show the possible transitions and the threat-score associated to them.
On the right side, the figure shows a timeline with a possible sequence of observables (top) and the state traces which WIDS/TinyWIDS store at different moments before the final state estimation (at time t5).

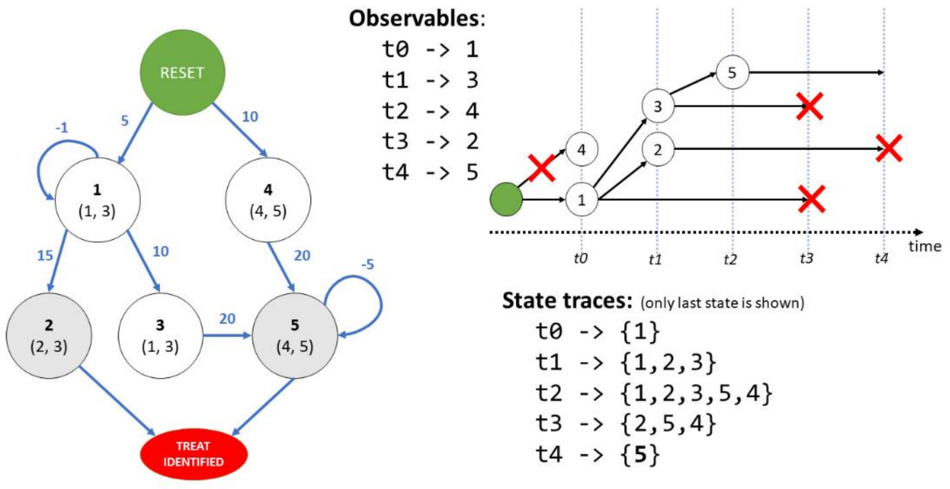


Figure 3.9 TinyWIDS State estimation

Relationship with other methods: -

Tool support: TinyOS (http://www.tinyos.net/), Agilla2 (https://github.com/luigi-pomante/Agilla2), TinyWIDS (https://github.com/luigi-pomante/TinyWIDS)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Semi-formal, Experimental - Monitoring
- Type of component under evaluation: Software, Model
- Evaluation tool: Open Source
- Evaluation stage: Validation
- Logic of the component under evaluation: Sensing, Thinking
- Type of requirements under evaluation: Non-functional - Cybersecurity
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_Agriculture_3 – Transmission line disturbances

Strengths

- Lightweight, low computational resource requirements
- Compatible with different WSN platforms
- Extensible

Limitations

- May require segmentation of the WSN to improve detection
- Higher software layers may need to adapt to the notification interfaces
- There is no way to react to an incoming attack. A WSN under attack remains vulnerable in the time between detection/notification and actual operator's action.

References

- [IDS1] Pugliese M., Giani A., Santucci F. (2010) Weak Process Models for Attack Detection in a Clustered Sensor Network Using Mobile Agents. In: Hailes S., Sicari S., Roussos G. (eds) Sensor Systems and Software. S-CUBE 2009. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 24. Springer, Berlin, Heidelberg
- [IDS2] Bozzi Luciano, Giuseppe Lorenzo, Pomante Luigi, Pugliese Marco, Santic Marco, Santucci Fortunato & Tiberti Walter. (2018). TinyWIDS: a WPM-based Intrusion Detection System for TinyOS2.x/802.15.4 Wireless Sensor Networks. 13-16. 10.1145/3178291.3178293.

Related standards: IEEE 802.15.4

Keywords: WSN, Intrusion Detection System, WPM, WIDS.

3.6.1.3 Kalman Filter-Based Fault Detector

Name: **Kalman filter-based fault detector**

Purpose: The main purpose of the method is to detect, using run-time data measured from a system via sensors, faults that can occur in the normal operation. In particular, when related to e.g. the agriculture use case, that consists of a robot that can be a target of faults and attacks in different design and system aspects related to CAN networks, radio link for remote teleoperation, GPS, etc., the aim is to monitor the robot and, in case of a deviation from the nominal operation, detect it and identify the corresponding fault.

Description: In control engineering fault detection is a process to detect, using run-time measurements, the occurrence of faults, which are unexpected or unwanted or intolerable behaviours of the system. This process in general generates alarm signals that indicate the early occurrence of faults or a trace that faults may have existed inside the system. Several research papers have been written on this topic, see e.g. [KFB1] and reference therein.

In particular, one of the pioneering works for Failure Detection and Identification is the one in [KBF2], where the problem of detecting and identifying control system component failures in linear time-invariant systems has been addressed. Design algorithms are given in the paper to detect and uniquely identify a component failure both in the case when system components can fail simultaneously, and in the case when they fail only one at a time. However, this approach, and the ones derived within this concept, suffer of a main issue: they assume that a mathematical model of the system is available.

Nowadays, it is well known that deriving a mathematical model for complex systems can be cost and time prohibitive. Thus, the research community started to investigate the so-called *data-driven methods*, where the information, and even a model, of the system are extracted from the collected historical data. In this respect, there are various methods which can be deployed to track faults. One among them is the approach based on the Principal Component Analysis (PCA) [KBF3], also enforced with the Kalman filter, presented in [KBF4] and [KBF5].

In [KFB4] the authors presented an output-only damage detection technique based on time-series models. They applied the PCA algorithm to the historical dataset in order to reduce the data size. Then, they fitted the data with an Auto Regressive (AR) model and calculated the Probability Density Function (PDF) of the damage feature, which was obtained by taking the ratio between the variances of prediction errors considering references and healthy dataset. A system was finally declared as *faulty* if the distance between the peak of the corresponding PDF and the healthy PDF was larger than the tolerance bound.

Such an approach has been then modified in [KFB5] by introducing a novel form of damage detection algorithm based on Recursive Principal Component Analysis (RPCA) together with the Time Varying Auto Regressive Model (TVAR), also leveraging the Kalman filter. In particular, they first used PCA on the dataset to project the dataset to its own orthogonal space, and then generated time series models based on the responses and fitted a TVAR model into the projected dataset. Finally, the Kalman filter was utilized to track the dynamics of the model effectively: the dynamics were tracked in real-time to extract the changes in the model coefficients, thereby revealing the faults in the system. The authors showed that the proposed approach resulted in a successful damage detection procedure, and that it worked well with both simulated and real data.

Although such data-driven methods address the modelling issues of the model-based approaches they work well when dealing with data that are generated by a system operating with linear dynamics. However, this is not usually the case of real complex systems. For this reason, new approaches are needed to take such issues into account.

From a practitioner point of view, the above methods can be implemented in two steps:

1. In the first step (training, off-line) a historical dataset is collected from the system of interest, and classical toolboxes (as detailed below) can be used to derive a mathematical model of the system;
2. In the second step (fault detection, run-time) on-the-fly data are collected from the system of interest, and a Kalman filter, which requires very light computational resources, as it is only based on linear operations on (generally small) matrices, is run over a real-time board. A fault detection is raised when the Kalman filter output significantly deviates from the nominal expected behaviour.

Relationship with other methods: Relation with methods for model-based fault detection, e.g. *Failure detection and diagnosis (FDD) in robotic systems*.

Tool support: System Identification toolbox (https://it.mathworks.com/products/sysid.html), Machine Learning toolbox (https://it.mathworks.com/products/statistics.html), Control Systems toolbox (https://it.mathworks.com/products/control.html)

Layers of the multi-dimensional framework

- Evaluation environment: in the lab
- Evaluation type: Experimental – Simulation, Analytical – Semi-formal
- Type of component under evaluation: Software, Model
- Evaluation tool: Open source
- Evaluation stage: Validation
- Logic of the component under evaluation: Sensing, Thinking

- Type of requirements under evaluation: Non-functional - Cybersecurity
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_Agriculture_3 – Transmission line disturbances

Strengths

- Fault detection performance

Limitations

- Difficulty in model identification
- PCA-based approach works well only with linear systems
- Lack of a bridge between the model-based approaches and Machine Learning to define data-driven fault detectors

References

- [KFB1] Ding, S. X. (2008). Model-based fault diagnosis techniques: design schemes, algorithms, and tools. Springer Science & Business Media.
- [KBF2] Massoumnia, M. A., Verghese, G. C., & Willsky A. S. (1989). Failure Detection and Identification. IEEE Transactions on Automatic Control, 34(3), 316 –321.
- [KFB3] Jolliffe, I. T. (1986). Principal component analysis. Springer, New York, NY.
- [KFB4] Lakshmi, K., & Rama Mohan Rao, A. (2014). A robust damage-detection technique with environmental variability combining time-series models with principal components. Nondestructive Testing and Evaluation, 29(4), 357-376.
- [KFB5] Bhowmik, B., Hazra, B., & Pakrashi, V. (2018). Real time damage detection using recursive principal components and time varying auto-regressive modeling. Mechanical Systems and Signal Processing, 101, 549-574.

Related standards: -

Keywords: Kalman filter, Fault detection.

3.6.1.4 Model-Based Safety Analysis

Name: **Model-based safety analysis**

Purpose: Define the requirements that a system has to fulfil, along with the procedures that have to be developed, in order to ensure a consistent safety level during the overall system lifecycle; analyse the failure propagation phenomena and evaluate their consequences in terms of safety and reliability, based on a formal model of the system of interest.

Description: Safety analysis techniques are well established and are used extensively during the design of safety-critical systems. Despite this, most of the techniques are highly subjective and dependent on the skill of the practitioner. Since analyses are based usually on an informal system model, it is unlikely that they will be complete, consistent, and error free. The lack of precise models of the system architecture and its failure modes often forces the safety analysts to devote much of their effort to gathering architectural details about the system behaviour from several sources and embedding this information in the safety artefacts such as the fault trees.
Model-Based Safety Analysis (MBSA) is an approach in which the system and safety engineers share a common system model created using a model-based development process. By extending the system

model with a fault model as well as relevant portions of the physical system to be controlled, automated support can be provided for much of the safety analysis.

MBSA enables deductive as well as inductive hazards analysis towards automated or semi-automatic generation of artefacts that are necessary for arguing about HARA (Hazards analysis and Risk Assessment) for certification-aware domains. Failure Mode Effects Analysis (FMEA) and the Fault Tree Analysis (FTA) are the two classical safety analyses considered in the proposed approach.

The FMEA is a “bottom-up” analysis based on a single-failure approach and executed on each system item or functional block, according to the following main steps:

1. Identification of credible failure modes
2. Evaluation of each single failure mode effects at various levels up to system level
3. Evaluation of severity of the failure effects consequences
4. Identification of failure detection methods
5. Assignment of the failure mode rate based on item reliability and apportionment criteria

An FTA is a model that graphically and logically represents the combinations of failures occurring in a system that leads to a hazardous condition. FTA uses a “top-down” approach, in order to identify all potential causes of a particular undesired top event. Starting from the Top Event (identified as a possible safety violation of interest), the analysis systematically determines all possible causes, both single fault and combination of faults, at the subsequent lower levels until a Basic Event is encountered. A Basic Event is defined as an event that is no further developed into a lower level of detail. If a basic event is attributed to items failures, it can be extracted from item failure modes analysed in FMEA.

Model-Based Safety Analysis builds upon methodologies to analyse the propagation of faults such as Failure Logic Analysis (FLA), with the intent to unify as well as (partially) automatize existing traditional dependability analysis approaches (e.g., Fault Tree Analysis, and Failure Modes and Effects Analysis). Similar to Failure Propagation Transform Logic (FPTC) [MSA1, MSA2], FLA [MSA3, MSA4, MSA5, MSA6] automatically calculates the failure behaviour of an entire system from the failure behaviour of its individual components. Failure behaviour of individual components, established by studying the components in isolation, is expressed by a set of logical expression rules that relate output failures (occurring on output ports) to combinations of input failures (occurring on input ports).

Model-Based Safety Analysis is based on the realization of a unique graphical model, defined using SysML, which defines both the functional and architectural characteristics and the aspects related to the system's behaviour in the presence of malfunctions. The use of the SysML model of the system and the FLA technique for the automatic calculation of the anomalous behaviour of the whole system starting from the anomalous behaviour of its individual functions and components, allows designers to perform automated safety analysis, with derivation of consistent safety analysis artefacts, such as FMEA and FTA, to support the safety assessment process, and have a quick response for the systems' decisions during the system design phase.

Using a common model for both system and safety engineering and automating parts of the safety analysis, we can both reduce the cost and improve the quality of the results.

Another embodiment of the Model-Based Safety Analysis methodology is based on the automatic injection of faults into the nominal model (i.e., without faults) of the system of interest, as depicted in Figure 3.10. In this approach, model extension is performed to enrich the nominal model with a library-based specification of the possible faults that may affect the behaviour of the system. The library of faults provides a specification of the most common failure patterns, and it is user-extensible. The extended model (including faults) of the system of interest can be analysed by means of exhaustive techniques based on model checking and produce artefacts such as FTs and FMEA tables. Both the nominal model and the extended model are specified using the SMV language. Translations into SMV are available from (variants of) the AADL architectural language and are available (or are targeted be implemented) from fragments of other languages, e.g., Simulink. In addition to safety assessment techniques such as FTA and FMEA, Model-Based Failure Safety Analysis also includes techniques to design and analyse the fault detection, isolation and recovery capabilities of a system of interest.

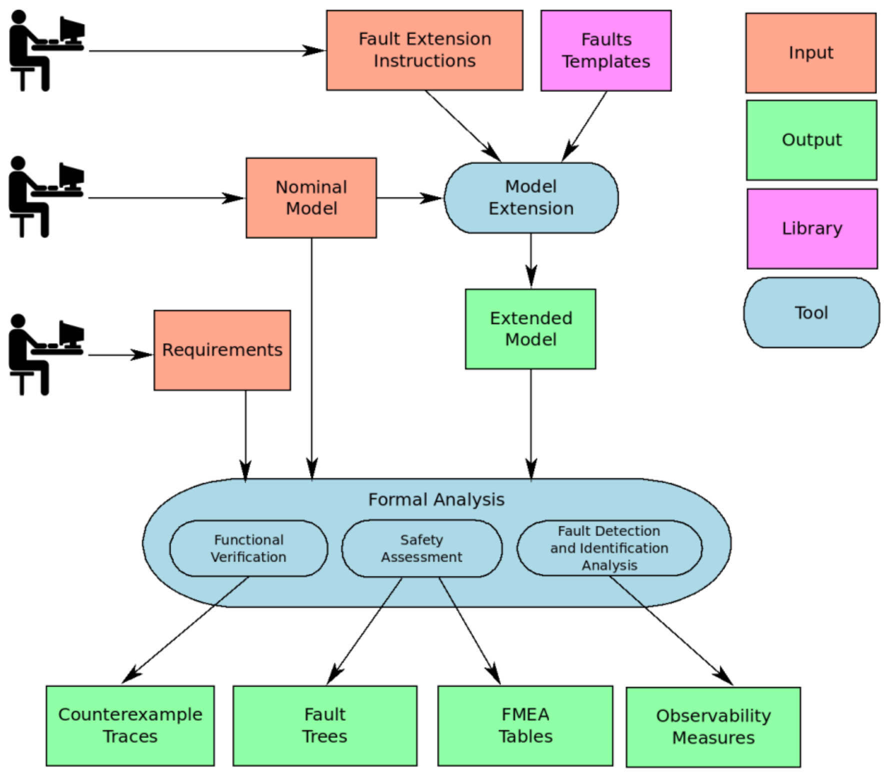


Figure 3.10 Fault injection for safety analysis

Relationship with other methods: *Model-Based fault injection for safety analysis* is a building block for Model-Based Safety Analysis. Other methods for model-based verification presented in this document are analogous but focused on functional and property verification. *Model checking* is an enabling technology.

Tool support: The CHESS (https://www.eclipse.org/chess/index.html) [MSA7] open-source toolset. The CHESS-FLA [MSA8] tool integrated in CHESS (developed through collaborations of INTECS with Malardalen University in the context of the CHESS, CONCERTO, AMASS European research projects). The xSAP https://xsap.fbk.eu/) Safety Analysis Platform [MSA9] (developed by FBK), built upon the nuXmv (https://nuxmv.fbk.eu/) model checker. The ocra (https://ocra.fbk.eu/ toolset for architectural and contract-based design (developed by FBK). The COMPASS (http://www.compass-toolset.org/) toolset [MSA10, MSA11], based on AADL (developed by FBK in several studies with funded by the European Space Agency).

Layers of the multi-dimensional framework

• Evaluation environment: In the lab • Evaluation type: Analytical – Formal, Analytical – Semi-formal • Type of component under evaluation: Model, Software, Hardware • Evaluation tool: Open source, Proprietary • Evaluation stage: Validation, Verification. • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Functional, Non-functional – Safety, Non-functional – Cybersecurity • Evaluation performance indicator: V&V process criteria, SCP criteria.
Use case scenarios • VALU3S_WP1_ Healthcare_2 - Safety analysis and certification • VALU3S_WP1_ Agriculture_1 - Vehicle switching from parallel guidance to manual mode • VALU3S_WP1_ Agriculture_2 - Vehicle switching from manual mode to parallel guidance • VALU3S_WP1_ Agriculture_3 - Transmission line disturbances
Strengths • Improves the communication between the system engineer and safety experts, by facilitating understanding of the logic and the eventual failures of the system • Achieves a systematic and comprehensive safety assessment that allows to early identify the greatest number of possible critical problems related to the impact of failures on the functionality of the system • Makes it easier to keep the system design aligned with the safety assessment • Carries out the main safety analyses (FMEA, FTA) in semi-automatic mode, thus receiving rapid feedback, with a consequent immediate impact on the design • Since the anomalous behaviour of the system is calculated from components, the impact of modifying a component is easier to define, and the derived incremental safety analysis is cheaper, therefore reducing the costs of system maintenance and reuse. • Developed from collaborative efforts in prior projects, e.g. CHESS (http://www.chess-project.org/), CONCERTO (http://www.concerto-project.org/), and AMASS (https://www.amass-ecsel.eu/). • Part of the CHESS Eclipse project
Limitations • Tool support usability can be improved. • FMEA and FTA support can be improved. • The automated analysis may be subject to the state-explosion problem, impacting the effectiveness of verification.
References • [MSA1] Wallace, Modular Architectural Representation and Analysis of Fault Propagation and Transformation, in proceedings of 2nd International Workshop on Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA 2005). • [MSA2] R. F. Paige, L. M. Rose, X. Ge, D. S. Kolovos, and P. J. Brooke. FPTC: automated safety analysis for domain-specific languages. In Models in Software Engineering, M. R. Chaudron (Ed.). Lecture Notes In Computer Science, Vol. 5421. Springer-Verlag, Berlin, Heidelberg, pp. 229-242, 2009Space Product Assurance: Software product assurance, id. ECSS-Q-ST-80 issue C, 06.03.2009.

- [MSA3] B. Gallina and S. Punnekkat, "FI4FA: A Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures Analysis," in Proc. of EUROMICRO, ser. SEAA '11. IEEE Computer Society, 2011, pp. 493–500.
- [MSA4] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat, "Model-driven dependability analysis method for component-based architectures," in Euromicro-SEAA Conference. IEEE Computer Society, 2012.
- [MSA5] Gallina, B., Sefer, E., and Refsdal, A. (2014). Towards safety risk assessment of socio-technical systems via failure logic analysis. In Proceedings of the 2014 IEEE International Symposium on Software Reliability Engineering Workshops, pages 287–292.
- [MSA6] B. Gallina and Z. Haider, A. Carlsson, S. Mazzini S. Puri, "Multi-concern Dependability-centered Assurance for Space Systems via ConcertoFLA", International Conference on Reliable Software Technologies- Ada-Europe 2018, Lisbon, June 2018
- [MSA7] Mazzini S., J. Favaro, S. Puri, L. Baracchi., "CHESS: an open source methodology and toolset for the development of critical systems", 2nd International Workshop on Open Source Software for Model Driven Engineering (OSS4MDE), Saint-Malo, October 2016.
- [MSA8] CHESS Dependability Guide – FLA (https://www.eclipse.org/chess/publis/CHESS_DependabilityGuide.pdf)
- [MSA9] B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli and G. Zampedri. The xSAP Safety Analysis Platform. In Proceedings of *TACAS 2016*. Eindhoven, The Netherlands, April 2-8, 2016.
- [MSA10] M.Bozzano, A.Cimatti, J.-P.Katoen, V. Y.Nguyen, T.Noll and M.Roveri. Safety, Dependability, and Performance Analysis of Extended AADL Models. *The Computer Journal, 54(5):754-775, 2011.*
- [MSA11] M. Bozzano, A. Cimatti, J.-P. Katoen, P. Katsaros, K. Mokos, V.Y. Nguyen , T. Noll, B. Postma and M. Roveri. Spacecraft Early Design Validation using Formal Methods. *Reliability Engineering & System Safety 132:20-35*. December 2014.

Related standards: IEC 61508, ISO 26262, EN 50129, SAE-ARP-4754, SAE-ARP-4761, CEI EN 62304, ISO 14971

Keywords: Hazards Analysis and Risk Assessment (HARA), Model-Based Safety Analysis (MBSA), Failure Logic Analysis (FLA), Failure Modes and Effects Analysis FMEA), Fault Tree Analysis (FTA), Fault Detection, Isolation and Recovery (FDIR).

3.6.1.5 Model-Based Threat Analysis

Name: **Model-based threat analysis**

Purpose: Derive threat information for a cyber-physical system based on a structural model and domain-specific threat databases. Enrich the system with mitigations that address the identified threats.

Description: Model-based threat analysis is a threat modelling [MTA1, MTA2, MTA3, MTA4] approach that utilizes STRIDE [MTA2] as a basis. It serves as means to analyse systems for threats as well as failures [MTA1], and consists of three major components [MTA3]:

1. A *system model* represents the system under consideration in its current status. This means that that the approach can be applied during the design phase where assumptions about the future system are driving development, as well as during the implementation phase which reveal shortcomings of the planned system and therefore results in an adaption of the system. Moreover,

model-based threat analysis can also be applied during the operational phase where the system is already running. A component may fail and, therefore, require replacement.
The system model is based on a data flow diagram [MTA4]. It holds all known security attributes on system components as well as the connections between them. [MTA3]

2. A *threat model* represents a digital twin of known threats. It is constituted of rules that allow for a later analysis of the system model. These rules are anti-patters, which are basically system configurations that are considered insecure and should therefore not be contained within the system under consideration. [MTA3]
3. A *threat analysis engine* that enables an automated analysis of the system. It compares each rule with the system model to detect potentially insecure configurations and consequently threats that the system under consideration may be affected by.

The whole threat modelling process results in a threat catalogue depicting threats that the system suffers from and, consequently, require treatment [MTA3]. This is supported by including the STRIDE [MTA2] category, as well as impact and attack feasibility of the threat [MTA5]. The current rule sets were derived from UNECE WP29, ETSI and the ITU.

The described approach is an iterative process which allows for consecutive analysis of the system with applied security measures that serve as mitigations.

Relationship with other methods: *Model-based safety-analysis, Risk analysis*.

Tool support: ThreatGet (https://threatget.com)

Layers of the multi-dimensional framework (Rupert)

- Evaluation environment: In the Lab
- Evaluation method: Analytical – Semi-Formal
- Type of component under evaluation: Model
- Evaluation tool: Proprietary
- Evaluation stage: Validation
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional - Cybersecurity
- Key performance indicators: SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_9 - Safety/security behaviour for corrupted data from remote control terminal

Strengths

- Potential threats can be automatically detected
- Can be applied during the whole system lifecycle
- Predefined impact and attack feasibility inform decisions for the risk treatment process

Limitations

- Needs a structural model of the system
- Requires a rule set for the target domain

References

- [MTA1] C. Schmittner, S. Chlup, A. Fellner, G. Macher, E. Brenner, ThreatGet: Threat modeling based approach for automated and connected vehicle systems, AmE 2020 - Automotive meets Electronics; 11th GMM-Symposium, March 2020

• [MTA2] A. Shostack, Threat modeling: designing for security, Wiley, 2014 • [MTA3] C. Schmittner, P. Tummeltshammer, D. Hofbauer, A. Shaaban, M. Meidlinger, M. Tauber, A. Bonitz, R. Hametner, M. Brandstetter, Threat Modeling in the Railway Domain, January 2019 • [MTA4] P. Torr, Demystifying the Threat-Modeling Process, IEEE Security and PRivacy Magazine, September 2005 • [MTA5] ISO/TC 22/SC 32, ISO/SAE DIS 21434 Road vehicles — Cybersecurity engineering, ISO – International Standardization Organization, May 2020
Related standards: ISO/SAE DIS 21434
Keywords: Threat modelling, Threat analysis, Security, STRIDE.

3.6.1.6 Risk Analysis

Name: **Risk analysis**
Purpose: To perform quantitative risk analysis of complex systems in different domains considering both safety and security issues
Description: In technical domains, risk is generally defined as the probability of damage including the severity of the considered damage, while Risk Assessment is a fundamental decision-making process in the development of information security, resulting in the selection of appropriate safeguards for an information system. Risk analysis is, indeed, the first step of the Risk Assessment process in which the system should be properly modelled as well as the potential threats. There are several methods for assessing risk, from checklists and questionnaires to algorithms and software tools. The choice of the Risk Analysis methodology should be taken on the basis of several factors, such as cost, complexity, completeness, feasibility, level of automation etc [RAS1]. According to [RAS1], Risk Analysis methodologies can be classified at a high-level following these two criteria: deterministic vs stochastic, quantitative vs qualitative vs hybrid. Beside the previous classification, methodologies developed during recent years can be defined also as: • *Scenario-based vs asset-threat-vulnerability (ATV) approach:* in scenario-based analysis risk scenarios are identified a priori, while in ATV methodologies risk scenarios are a consequence of the assets, threats and vulnerabilities defined; • *Cross-sectorial*, when the methodology could be applied to several domains according to definition provided in [RAS2], or single-sectorial, when the methodology could be applied only in a specific sector. Quantitative Risk analysis indeed should be preferred when it comes the evaluation of a complex system with high safety standards, since it is more reliable and not affected by subjective judgment. One of the main challenges to address in the quantitative Risk Analysis is the scarcity of data, which makes it difficult to use probabilistic techniques to foresee risks and potential consequences. To solve this issue efforts have been spent in two ways: on one hand the research and analysis of existing dataset provided by official international organisation, such as Global Terrorism Database (GTD) [RAS4] for terrorism and Major Accidents Reporting System (eMARS) for industrial incidents[RAS5], has been carried out, while on the other hand Artificial Intelligence techniques, such as Neural Network, or Bayesian approaches have been applied to overcome the data gap.

Another challenging issue is the modelling of multi-hazard risk, i.e. when the risk is generated by a set of hazards which can occur together or in sequence, namely cascading effect. Indeed, the manifestation of various threats, either jointly or very close in time, is an eventuality that must be considered in order to perform an exhaustive risk analysis for a given system. It is possible dealing with this task in two different ways: considering the events as independent or introducing a relation among the threats. The first approach is the most diffused and the easiest one, because less data and a smaller number of calculations are needed. However, assuming the independence of the hazards leads to a reduction of information and to a less accurate model. For this reason, evaluating the correlations among events is the best practice, even if it is more expensive in terms of modelling and requires much information [RAS6].

Within VALU3S project, the Risk Analysis chosen to deal with complex electronic and automated systems is:

- Quantitative: to provide reliable, objective and measurable outputs;
- Deterministic: to provide exact values for outputs;
- ATV approach-based: to generate risk scenario starting from the knowledge of the system;
- Cross-sectorial: to be applied in different domains.

The considered risk analysis method generally takes the following steps:

1. Generation of attack/incident scenarios according to system structure and components;
2. Simulation of attack/incident scenarios and intervention of safety countermeasures to generate scenario outcomes (e.g. attack prevented, defused, mitigated or not mitigated);
3. Computation of outcomes likelihood;
4. Computation of outcomes impact;
5. Computation of outcomes risk.

The methodology is flexible because the generation of scenarios is performed according to the initial system modelling. This feature allows to apply the methodology to different systems, regardless of their peculiarities and interdependencies. On the other hand, this approach cannot allow a detailed modelling of system components.

The inputs needed by the algorithm (Figure 3.11) to perform a scenario-based cross-sectorial quantitative risk analysis are the following:

- A data model of the infrastructure (or system considered), e.g. which components are present;
- Historical data, e.g. data on past incident/attacks for systems similar to the one under consideration;
- Economic figures, such as the monetary values of system components;
- Safety and security measures which are present in the system.

The outputs of the scenario-based cross-sectorial quantitative risk analysis are the following:

- Risk level of each component of the system, expressed as €/year, according to each scenario;
- The likelihood of each scenario generated;
- Number of casualties and injuries due to attacks/incidents for each scenario;
- An estimation of the percentage physical damage to system components due to attacks/incidents for each scenario;
- The economic losses due to the impact of attack/incidents for each scenario;

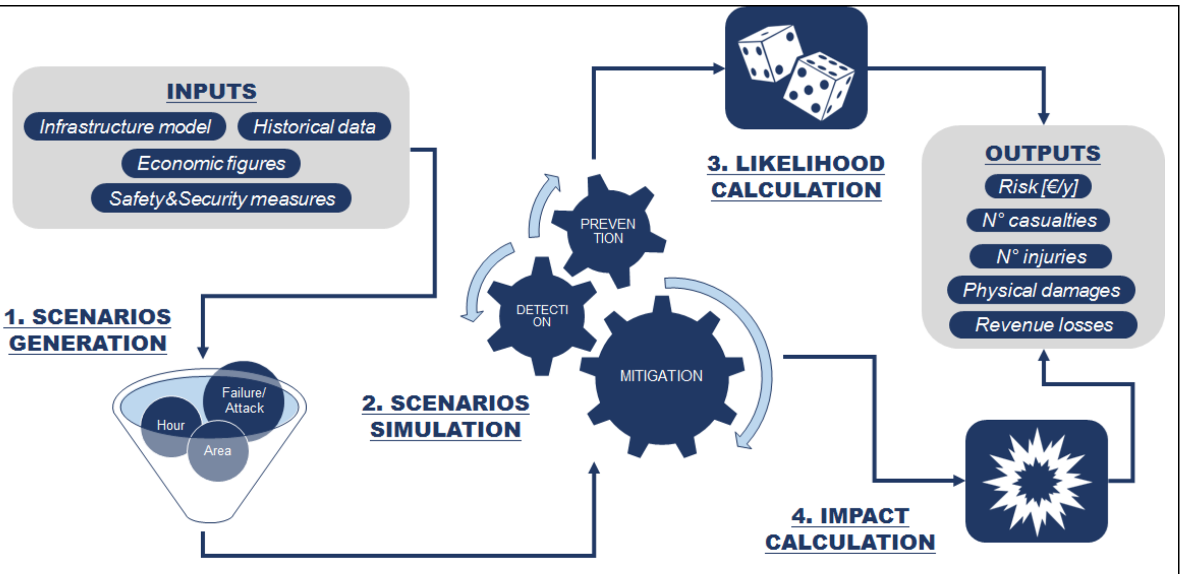


Figure 3.11 Scheme of the algorithm underlying scenario-based cross-sectorial quantitative risk analysis

Relationship with other methods: *Model-based safety analysis, Model-based threat analysis, Risk-based testing*

Tool support: The methodology described above has been implemented by STAM in the RAMSES tool (Risk Analysis Micro-SErvice Solution). RAMSES is indeed a computation engine capable to automatically carry out a ATV Cross-sectorial Quantitative Deterministic Risk Analysis having received the requested input for running the algorithm. RAMSES is provided as micro-service, indeed it can be easily incapsulated in other external tools and then interrogated through a request when the risk analysis outputs are needed.

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Semi-formal
- Type of component under evaluation: Hardware
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Acting
- Type of requirements under evaluation: Non-functional - Safety, Non-functional - Cybersecurity
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_ Agriculture_1 - Vehicle switching from parallel guidance to manual mode
- VALU3S_WP1_ Agriculture_2 - Vehicle switching from manual mode to parallel guidance
- VALU3S_WP1_ Agriculture_3 - Transmission line disturbances

Strengths

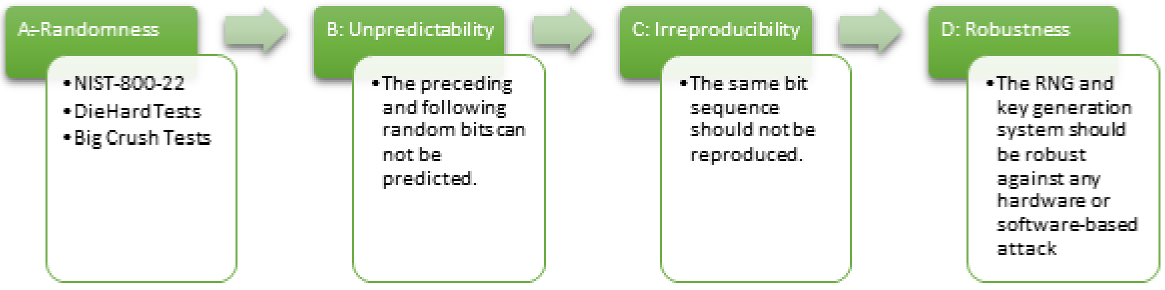
- Generation and analysis of thousands of attack/incident scenarios;
- Quantitative outputs which can measure the risk level of the system in an objective manner;
- Risk estimation based on reliable historical dataset mixed with AI techniques.

Limitations

- Need of a large input dataset, including a precise modelling of the infrastructure;
- Interdependencies among system components modelled only at high level.

References • [RAS1] Lichtenstein, S. (1996) "Factors in the selection of a risk assessment method", Information Management & Computer Security, Vol. 4, No. 4, 20-25, MCB University Press, U.K. • [RAS2] Jérôme Tixier, Gilles Dusserre, Olivier Salvi, Didier Gaston. Review of 62 risk analysis methodologies of industrial plants: review. Journal of Loss Prevention in the Process Industries, Elsevier, 2002, 15 (4), pp.291-303. ff10.1016/S0950-4230(02)00008-6ff. ffineris-00961858f 1 • [RAS3] European Risk Assessment and Contingency, EURACOM, Grant Agreement: 225579 Seventh Framework Programme Theme ICT-SEC-2007-7.0-0 • [RAS4] Global Terrorism Database, https://start.umd.edu/gtd/ • [RAS5] eMars Database, https://emars.jrc.ec.europa.eu/en/emars/content • [RAS6] Pasino A., Clematis A., Battista U., De Angeli S., Ottonello D. (2020) “A review of single and multi-hazard risk assessment approaches for critical infrastructures protection”, 2nd Scientific International Conference on CBRNe SICC Series
Related standards: ISO31000, ISO25119, ISO14849, ISO26262, IEC61508
Keywords: Risk analysis, Attack, Incident, Safety, Failure, Security, Threat, Vulnerability, Protection, Impact, Likelihood, Scenario, Prevention, Detection, Diffusion, Mitigation, Complex system, Interdependency.

3.6.1.7 Vulnerability Analysis of Cryptographic Modules Against Hardware-Based Attacks

Name: **Vulnerability analysis of cryptographic modules against hardware-based attacks**
Purpose: To analyse the potential vulnerabilities originated from key generation in any cyber-physical system. The aim of this analysis is to assure that the random number and cryptographic key generation scheme cannot be hacked and the generated bit sequences cannot be predicted.
Description: One of the basic principles of cryptography is that according to Kerckhoff’s hypothesis [VAC1], it is assumed that the overall security of any cryptosystem is **completely dependent on the security of the key**, and that **all other parameters of the crypto system are publicly observable**. Thus, the cryptographic algorithms are assumed as open as long as the key generation scheme is not secure. In real life, many systems actually use well-known symmetric and asymmetric cryptographic algorithms (AES, 3DES, RSA, ECDSA, SHA) which have been applied in many domains and all experts are aware of their strengths and weaknesses. Vulnerability analysis is complementary to cryptography. The robustness of a crypto system depends on the key used, or in other words, the attacker’s ability to predict the key. The hardware-based vulnerability analysis of key generation relies on 4 steps (Figure 3.12):  A: Randomness •NIST-800-22 •DieHard Tests •Big Crush Tests B: Unpredictability •The preceding and following random bits can not be predicted. C: Irreproducibility •The same bit sequence should not be reproduced. D: Robustness •The RNG and key generation system should be robust against any hardware or software-based attack *Figure 3.12 The 4-step evaluation process for cryptographic modules*

A: Randomness: There are two basic types of generators used to produce random sequences: true random number generators (TRNGs) and pseudorandom number generators (PRNGs) [VAC2]. TRNGs generate random numbers from a physical process (thermal noise, the photoelectric effect, ring oscillator) rather than by means of an algorithm. While TRNGs take the advantage of non-deterministic entropy sources, PRNGs generate bits in a deterministic manner. The PRNG-generated sequence is not truly random, because it is completely determined by an initial value, called the PRNG's seed (which may include truly random values). PRNGs tend to benefit from the external source of randomness (e.g., mouse movements, delay between keyboard presses etc.) which are practical in use but predictable either. In other words, an attacker can easily guess the outputs of a PRNG by applying statistical or machine learning methods so that the cryptographic keys or secrets (which uses the bit streams as outputs of PRNGs) also become guessable (the previous or next bits can be predicted). The first thing to apply for the vulnerability analysis of a cryptosystem is to check whether the system relies on a hardware-based RNG or not. Then, this RNG should be TRNG. To meet the true randomness criteria, three test suites are applied on a sufficient length of bit sequences: I) NIST-800-22 Randomness Test Suite [VAC3]; II) DieHard Test Suite [VAC4]; III) Big Crush Test Suite [VAC5]

The typical outputs of a TRNG must pass all 15 criteria of NIST-800-22 Randomness Test Suite which are listed as: i) Frequency (Monobit) ; ii) Frequency Test within a Block; iii) Runs; iv) Longest-Run-of-Ones in a Block; v) Binary Matrix Rank; vi) Discrete Fourier Transform (Spectral); vii) Non-overlapping Template Matching; viii) Overlapping Template Matching; ix) Maurer's "Universal Statistical"; x) Linear Complexity; xi) Serial ; xii) Approximate Entropy; xiii) Cumulative Sums (Cusums); xiv) Random Excursions; xv) Random Excursions Variant. Diehard and Big Crush Test Suites also cover additional criteria that should be met to evaluate the randomness.

B. Unpredictability: The randomness test suites [VAC3-5] are useful but the main message given by these tests is that the tested sequence is NOT random. As they rely on statistical techniques the randomness test suites do not give any guarantee about the true randomness of a bit sequence. Even if a bit sequence fulfils all the randomness test suites, they cannot present any quantification about their predictability. In order to say that a TRNG is reliable, the preceding and following random bits generated by the TRNG cannot be predicted by any technique. Although there exist alternative methods, the proposed predictability analysis was presented in [VAC6].

An ideal entropy source which is used as the core of a TRNG based on thresholding random noise should have constant power spectral density over its operating bandwidth. The proposed vulnerability analysis technique in VALU3S relies on the generation of independent and uniformly distributed bits, per-sample joint entropy of the generated bit sequence and the autocorrelation function of the output (also addressed in [VAC6]). Here, it is assumed that the underlying noise waveform is a continuous-time wide sense stationary Gaussian process, of which power spectral density is flat between two known frequencies. Given a continuous time wide-sense stationary flat band-limited Gaussian noise source, the test aims to investigate the necessary and sufficient conditions for generating independent and identically distributed random bits with uniform marginal probabilities via uniformly sampling from this process and providing analytical and numerical results for the per-sample joint entropy. This is realised by preparing an equivalent setup in place of the original one in order to make the derivations simpler.

C: Irreproducibility & D: Robustness: The strength of a crypto system almost depends on the strength of the key used or in other words on the difficulty for an attacker not only to predict the key but also assure the irreproducibility of the same bit sequence and recommend a robust design against attacks. The TRNGs should not rely on the deterministic sources, otherwise one can even predict the secret parameters of RNGs [VAC7]. In recent years, the majority of the TRNGs rely on non-linear systems, e.g. chaotic RNGs. So, in C and D, the vulnerability analysis of the TRNG is tackled so that the irreproducibility and robustness against attacks is jointly encountered.

This is realised by a novel predicting system (as addressed in [VAC6-9]) that is proposed to find out the security weaknesses of a chaotic RNG. Convergence of the predicting system is proved using auto-synchronization. Secret parameter of the target chaotic RNG is revealed where the public information is the design of the chaotic RNG and a scalar time series observed from the target chaotic system. Simulation and numerical results verifying the feasibility of the predicting system are also given similar to the recent papers of ERARGE [VAC8, VAC7, VAC9] such that next bit can be predicted as well while the same output bit sequence of the chaotic RNG can be regenerated.

Relationship with other methods: *Model-based threat analysis.*

Tool support: AnadigmDesigner2 (Discrete Chaotic Map design tool), MATLAB (ERARGE's Statistical Analysis Method), Xilinx Vivado Design Suite (FPGA Design Flow), BigCrush (TestU01, Statistical Test Suite) (https://www.iro.umontreal.ca/~lecuyer/), DieHard (Statistical Test) (http://stat.fsu.edu/_geo/diehard.htm), NIST 800-22 (Statistical Test Suite) (https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical- Semi-formal
- Type of component under evaluation: Hardware
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Validation
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-Functional - Cybersecurity
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_1 - Manipulation of sensor data
- VALU3S_WP1_Industrial_2 - Server and PLC communication
- VALU3S_WP1_Industrial_4 - Anomaly detection at component and system level

Strengths

- The proposed test covers not only one test suite but also a combination of three methods (NIST-800-22, DieHard, BigCrush). By doing so, the randomness analysis, when applied, will not miss any vulnerability related to true randomness which is a critical obligation to generate private cryptographic keys. The vulnerability analysis method is a strong methodology to verify and validate the cryptographic key generation which is an indispensable component for the end-to-end security of any cyber-physical smart system. It is strong because of two reasons: first it relies on hardware-based techniques so that it is not PRNG; second it does not address only the randomness of the bit sequence (generated by a RNG) but also its unpredictability, irreproducibility and the robustness (combined).

• The presented technique (within the context of VALU3S) can also be used to contribute to the privacy preservation as it prevents the leakage of any confidential information within/from the cyber-physical system components.
Limitations • The analysis and tests that relies on our previous work [VAC7-9], [VAC10-11] were done with "(discrete and continuous time) chaotic oscillator [VAC7-9] and ring oscillator-based RNGs [VAC11-13]. These techniques should be made compliant with the use cases and generalised within the scope of the project. For instance, The C. Irreproducibility and D. Robustness tests have not been generalised to any true random number generation scheme. So, our plan is to generalise the 4-step vulnerability analysis (A-B-C-D steps) valid and applicable to all domains where cryptographic components are used. • For Big Crush tests at least 1TB data is needed. Thus, this test can only be implemented over strong computers (e.g. work stations)
References • [VAC1] K. Martin, Everyday Cryptography: Fundamental Principles and Applications, 2nd Edition, Oxford University Press,2017. • [VAC2] S. Ergün, and S. Özoguz, Truly random number generators based on a non-autonomous chaotic oscillator, AEU-International Journal of Electronics and Communications 61, no. 4: 235-242, 2007. • [VAC3] L.E. Bassham III, A.L. Rukhin, J. Soto, J.R. Nechvatal, M.E. Smid, E.B. Barker, S.D. Leigh, M. Levenson, M. Vangel, D.L. Banks, N.A.Heckert, Sp 800-22 rev. 1a. A statistical test suite for random and pseudorandom number generators for cryptographic applications, National Institute of Standards & Technology, 2010. • [VAC4] G. Marsaglia, DIEHARD Statistical Tests: 2001. ttps://tams.informatik.uni-hamburg.de/paper/2001/SA_Witt_Hartmann/cdrom/Internetseiten/stat.fsu.edu/source.tar.gz, • [VAC5] P. L'Ecuyer, and R. Simard, TestU01: A C library for empirical testing of random number generators, ACM Transactions on Mathematical Software (TOMS) 33, no. 4: 1-40, 2007. • [VAC6] N.C. Göv, M.K. Mıhçak, S. Ergün, True random number generation via sampling from flat band-limited Gaussian processes, IEEE Transactions on Circuits and Systems I: Regular Papers 58, no. 5: 1044-1051, 2010. • [VAC7] S. Ergün: Predicting the Secret Parameters of a Chaotic Random Number Generator from Time Series, In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2547-2551. IEEE, 2019. • [VAC8] S. Ergün: On the security of chaos based "true" random number generators, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 99, no. 1: 363-369, 2016. • [VAC9] S. Ergün, and B. Acar: Revealing the Secret Parameters of an FPGA-Based "True" Random Number Generator, IEEE International Symposium on Circuits and Systems (ISCAS), 1-4, 2020 • [VAC10] B. Acar, S. Ergün: Correlation-based cryptanalysis of a ring oscillator based random number generator, 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), 1050-1053, 2018.

• [VAC10] B. Acar, S. Ergün: A Digital Random Number Generator Based on Irregular Sampling of Regular Waveform, 2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS), 221-224, 2019. • [VAC11] B. Acar, S. Ergün: A Reconfigurable Random Number Generator Based on the Transient Effects of Ring Oscillators, IEEE Transactions on Circuits and Systems II: Express Briefs 67 (9), 1609-1613, 2020. • [VAC12] R. Günay, S. Ergün: IC Random Number Generator Exploiting Two Simultaneous Metastable Events of Tetrahedral Oscillators, IEEE Transactions on Circuits and Systems II: Express Briefs 67 (9), 1634-1638, 2020. • [VAC13] K. Demir, S. Ergün: Random Number Generators Based on Irregular Sampling and Fibonacci–Galois Ring Oscillators, IEEE Transactions on Circuits and Systems II: Express Briefs 66 (10), 1718-1722, 2019.
Related standards: NIST-800-22, FIPS-140-2, Common Criteria EAL4+
Keywords: True random number generators, Cryptosystem, Predictability, Reproducibility, Cryptanalysis, Vulnerability analysis, Hardware-based attacks.

3.6.1.8 Wireless Interface Network Security Assessment

Name: **Wireless interface network security assessment**
Purpose: The main purpose is to evaluate the system's robustness against network security attacks carried out through wireless interfaces. In detail, it aims to evaluate CANBUS-based control network security and teleoperation and supervision network security in different system's domains.
Description: Many works have shown that attackers can succeed in remotely executing malicious code in Electronic Control Units (ECUs) of e.g. vehicles via different (wireless) remote-control interfaces. Such attacks can take place by sending some kinds of wireless signals, compromising listening ECUs, and subsequently injecting malicious code. This represents a huge risk even for low-criticality ECUs, because an attack can propagate to other, more critical ECUs through the control network inside the system. E.g. the control systems inside of road and agricultural vehicles are based on CANBUS, which is a highly insecure protocol. It is difficult to measure how susceptible a particular system is to remote attacks since it depends on the presence of different possible kinds of vulnerabilities. The first step is to analyse the entire attack surface of the system and use this information to estimate susceptibility to attacks. This kind of analysis includes assessing how large the remote attack surface is, how segmented the ECUs which have physical control of the system are from the accepting external input viewpoint, and the components present in the system which can allow malicious users to gain control over safety-critical aspects.
Relationship with other methods: *Vulnerability and attack injection.*
Tool support: Open-source tools for packet injection in the CANBUS network and security of the gateway, such as The Open Web Application Security Project - OWASP (https://owasp.org/)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Experimental – Testing, Analytical – Semi-formal • Type of component under evaluation: Model, Hardware, Software • Evaluation tool: Open Source • Evaluation stage: Validation, Verification

- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional - Cybersecurity
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Agriculture_1 - Vehicle switching from parallel guidance to manual mode
- VALU3S_WP1_Agriculture_2 - Vehicle switching from manual mode to parallel guidance
- VALU3S_WP1_Agriculture_3 - Transmission line disturbances

Strengths

- Many critical threats and hazards can be thoroughly investigated
- Ease of result exploitation
- Possible application in a large set of scenarios to investigate a wide set of possible attacks
- Allow schematization of common vulnerabilities in different wireless interfaces

Limitations

- Time consuming procedure
- Lack of a universal standard procedure

References

- [WIN1] Miller C., Valasek C., "A survey of remote automotive attack surfaces", black hat USA 2014, whitepaper, 2014.
- [WIN2] Miller C., Valasek C., "Adventures in automotive networks and control units", Def Con 21 whitepaper, 2013.
- [WIN3] Pan L., Zheng X., Chen, H. X., Luan T., Bootwala H., Batten L., "Cyber security attacks to modern vehicular systems", Journal of Information Security and Applications, volume 36, pages 90-100, 2017.
- [WIN4] "The Open Web Application Security Project (OWASP)", https://owasp.org/

Related standards: ISO 11898-1, ISO 15765-2, ISO 14229, ISO 14230

Keywords: CANbus attack, WiFi access exploitation, Radio link access exploitation.

3.6.2 General Semi-Formal Analysis

This sub-group of methods focuses on semi-formally evaluating general characteristics of a system, e.g. about the traceability between system artefacts. These characteristics indirectly address automated system SCP by confirming the fulfilment of aspects that contribute to it. For instance, requirements traceability contributes to assuring that the correct and expected functionality has been implemented in a system. This in turn contributes to developing confidence in system reliability and consequently in SCP. In other words, if someone cannot confirm that the correct and expected functionality has been implemented in a system, it might not be possible to develop sufficient confidence in system SCP.

General semi-formal methods are common in engineering and assurance standards. Traceability is among the main examples. The standards also usually refer to general V&V means that complement formal and informal ones, and in line with recent trends in system and component development. For instance, the reference to model-based means is common in the latest versions of functional safety standards, as model-based systems engineering is increasingly adopted in practice and its suitability to tackle safety needs is confirmed.

The review below of V&V methods for general semi-formal analysis deals with different artefact types and is based on different approaches. Model-based methods are reviewed, and also methods that are based on ontologies. Certain methods focus on specific artefact types, e.g. design specifications and source code, whereas others can be applied on different types, such as traceability management, which in principle might need to deal with any system artefact. As a whole, the set of methods complement each by either addressing different SCP evaluation needs or by proposing alternative general semi-formal means from which other methods can benefit. The methods could be combined as part of a common V&V process for general semi-formal analysis.

3.6.2.1 Code Design and Coding Standard Compliance Checking

Name: **Code design and coding standard compliance checking**
Purpose: To facilitate verifiability of the produced code.
Description: Coding standards can be used to facilitate verifiability of the produced code. If employed they tend to exclude dynamic variables or objects such as unwanted or undetected overlay of memory, and bottlenecks of resources during (safety-related) runtime. The software development shall involve the limitation of the use of interrupts, in order to keep the software verifiable and testable, as well as the use of pointers, which shall be strictly defined, in order to avoid the problems caused by uncontrolled accessing data and recursion, to avoid unverifiable and untestable use of subroutine calls. Detailed rules shall be fully agreed upon before coding. These typically require specific actions and patterns in the design of each software component, such as modularization guidelines, inheritance depth, etc. Other points covered: • Dynamic variables use and checking: using dynamic variables cannot be checked by the compiler or other off-line tools, if they are to be used, their disposition must be handled; • If interrupts are used, then parts not able to be interrupted shall have a specified maximum computation time, so that the maximum time for which an interrupt is inhibited can be calculated. Interrupt usage and inhibiting shall be thoroughly documented; • Pointer arithmetic shall be used at source code level only if the pointer data and value range are checked; • The use of recursion shall be documented in terms of depth of recursion These rules enable ease of software component testing, verification, assessment and maintenance.
Relationship with other methods: *Runtime Verification* methods and *Software Component Testing* are facilitated by coding rules.
Tool support: Spreadsheets, CodeBeamer (Intland; https://codebeamer.com/)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Analytical – Semi-formal • Type of component under evaluation: Software • Evaluation tool: Proprietary • Evaluation stage: Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Non-functional - Safety

• Evaluation performance indicator: V&V process criteria
Use case scenarios • VALU3S_WP1_Agriculture_1 - Vehicle switching from parallel guidance to manual mode • VALU3S_WP1_Agriculture_2 - Vehicle switching from manual mode to parallel guidance
Strengths • Uniforms code creation enhancing component testing, verification and assessment; • Development is more solid and maintenance is less time consuming;
Limitations • Could not be applied easily to model-based designs • Method compliance hard to verify • Limited tool support for fully coverage
References • [CDC1] Bagnara R., Bagnara A., Hill P.M. (2018) The MISRA C Coding Standard and its Role in the Development and Analysis of Safety- and Security-Critical Embedded Software. In: Podelski A. (eds) Static Analysis. SAS 2018. Lecture Notes in Computer Science, vol 11002. Springer, Cham. https://doi.org/10.1007/978-3-319-99725-4_2 • [CDC2] G. J. Holzmann, "The power of 10: rules for developing safety-critical code," in Computer, vol. 39, no. 6, pp. 95-99, June 2006, doi: 10.1109/MC.2006.212.
Related standards: ISO 25119, ISO 26262, IEC 61508, ISO 11849
Keywords: Coding rules.

3.6.2.2 Knowledge-Centric System Artefact Quality Analysis

Name: **Knowledge-centric system artefact quality analysis**
Purpose: Quantitatively determine the suitability of system artefacts by exploiting ontologies and semantic information, and according to selected criteria, e.g. correctness, consistency, and completeness.
Description: Knowledge-centric system artefact quality analysis [KCQ1, KCQ3] is based on the RSHP model [KCQ2] (see Figure 3.13), which allows the representation of any type of information or knowledge about a domain, including the semantics of the element under consideration. RSHP enables knowledge-centric systems engineering, which is a specialisation of model-based systems engineering based on the idea that all systems engineering processes are affected by the existence of a knowledge base (ontology) about a system and its lifecycle. Knowledge bases can be represented as ontologies with the structure shown in Figure 3.14. Such ontologies support system artefact quality analysis according to the information in the different layers: • The most inner layer (Terminology) corresponds to the terms of a domain together with their syntactic information, e.g., about whether a term such as 'car' is a noun. • Relationships between the terms can be specified in the conceptual model layer, as well as their semantics with clusters; e.g., the semantics of the terms 'car' and 'truck' can be 'system', and they specialise 'vehicle'. • Patterns can then be developed to provide templates (aka boilerplates) for system information specification. The patterns refer to aspects of the two underlying layers; e.g. in the pattern "The

[System] shall [Detect] [Item] at a minimum range of [Number] seconds", the elements in squared brackets correspond to semantic clusters.

- The Formalization layer deals with the semantic representation (in RSHP format) of system information according to patterns. This representation can correspond to system artefacts in different formats, e.g., text or a model, and of different types, e.g., requirements and design elements.
- Finally, at the Inference rules layer the data in all the others can be exploited for the specification of procedures to derive information, e.g., about specification correctness

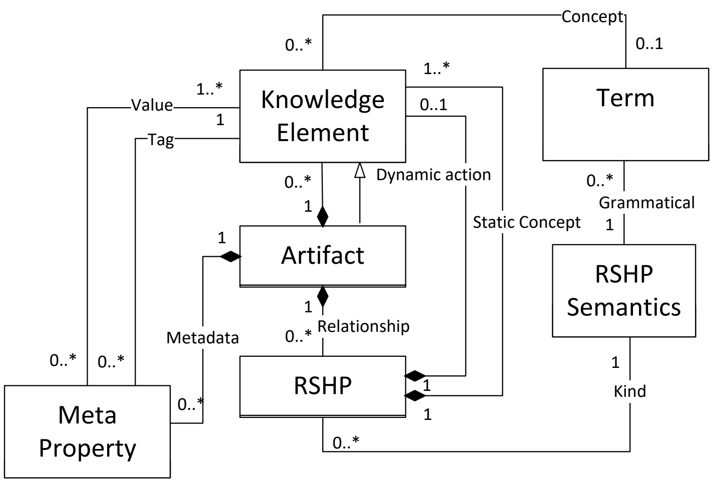


Figure 3.13 Core elements of the RSHP information representation language

The knowledge base can be exploited for artefact quality analysis, e.g., to determine if a given requirement contains valid terms or has been correctly specified according to some pattern. Quality can be quantitatively assessed by means of quality functions, which will indicate if artefact quality is good, medium, or bad according to ranges.

As RSHP enables the representation of any type of information, quality analysis can be conducted for different system artefact types (requirements, design…) and in different formats (text, models…).

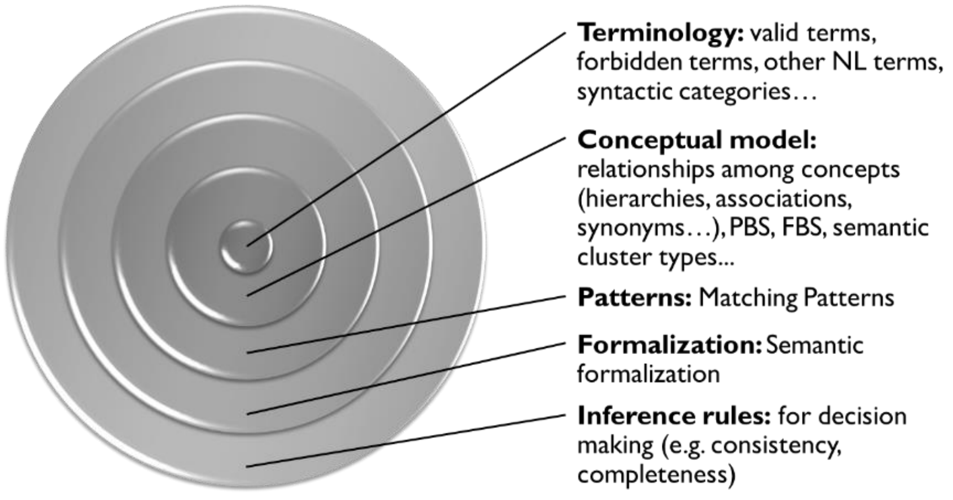

Terminology: valid terms, forbidden terms, other NL terms, syntactic categories…
Conceptual model: relationships among concepts (hierarchies, associations, synonyms…), PBS, FBS, semantic cluster types...
Patterns: Matching Patterns
Formalization: Semantic formalization
Inference rules: for decision making (e.g. consistency, completeness)

Figure 3.14 Ontology structure for Knowledge-centric system artefact quality analysis

Relationship with other methods: *Knowledge-centric traceability management* is based on RSHP and ontologies as well. Knowledge-centric system artefact quality analysis has been integrated with *Model-based assurance and certification*. Other methods that also assess artefact quality characteristics before execution include *Formal verification* ones.

Tool support: RQA Quality Studio (https://www.reusecompany.com/rqa-quality-studio)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Semi-formal
- Type of component under evaluation: Model
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional – Safety, Non-functional – Others
- Evaluation performance indicator: SCP criteria

Use case scenarios

- VALU3S_WP1_ Automotive_10 - System certification
- VALU3S_WP1_ Automotive_12 - ADAS system has to be reliable and has to comply with safety standards
- VALU3S_WP1_ Healthcare_3 - Certification needs of the NMT device
- VALU3S_WP1_ Healthcare_6 - Assurance needs of the NMT device

Strengths

- Industrial method in use for years
- More consistent and systematic system artefact quality analysis
- The method can be tailored to any system artefact type

Limitations

- There exist many types of system artefacts and the analysis of some is not supported
- No empirical evidence of cost-effectiveness

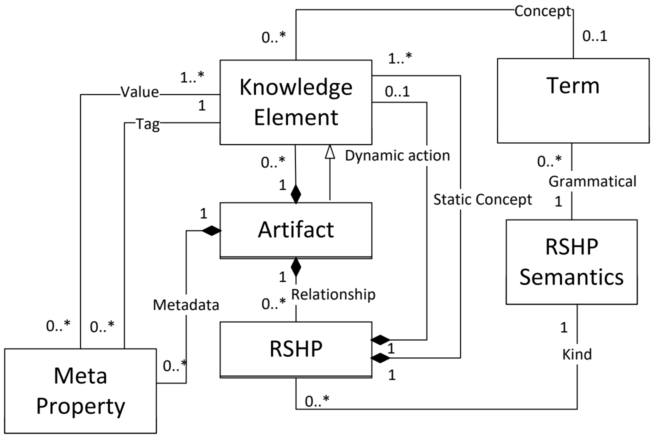
References

- [KCQ1] AMASS Project: Deliverable D3.3 - Design of the AMASS tools and methods for architecture-driven assurance (b). 2018
- [KCQ2] Llorens, J., Morato, J., Genova, G.: RSHP: an information representation model based on relationships. In Soft computing in software engineering. Springer. 2004
- [KCQ3] Parra, E., Alonso, L., Mendieta, R., de la Vara, J.L.: Advances in Artefact Quality Analysis for Safety-Critical Systems. 30th International Symposium on Software Reliability Engineering (ISSRE 2019)

Related standards: Assurance standards can request that certain quality characteristics of system artefacts such as requirements and design specifications are ensured, e.g., their correctness, consistency, and completeness. Examples of these standards include DO-178C, EN 50128, IEC 61508, and ISO 26262.

Keywords: System artefact, Quality, Quality analysis, Metric, Knowledge representation, Ontology, Knowledge-centric systems engineering

3.6.2.3 Knowledge-Centric Traceability Management

Name: **Knowledge-centric traceability management**
Purpose: To manage the relationships between system artefact by taking advantage of ontologies and semantic information.
Description: Traceability between system artefacts is a key activity to ensure that a critical system's lifecycle has progressed adequately and that characteristics of and between the artefacts are fulfilled. For example, it must be typically ensured that all system requirements are satisfied by some system design element and validated by some test case. Knowledge-centric traceability management [KCT1] is based on the RSHP model ([KCT3]; see Figure 3.15), which allows the representation of any type of information or knowledge about a domain, including the semantics of the element under consideration. RSHP enables knowledge-centric systems engineering, which as a specialisation of model-based systems engineering based on the idea that all systems engineering processes are affected by the existence of a knowledge base (ontology) about a system and its lifecycle.  Concept 0..* 0..1 1..* Knowledge Element Term Value 1..* 0..1 1 Tag 0..* Dynamic action 0..* Grammatical 1 Static Concept 1 1 Artifact RSHP Semantics 1 Metadata Relationship 0..* 0..* 0..* 1 RSHP Kind Meta Property 0..* 1 1 0..* *Figure 3.15 Core elements of the RSHP information representation language* As outlined in Figure 3.16, traceability is managed in the approach though Traceability Projects that contain Traceability Modules, which in turn consist of traces. A traceability module is a special kind of RSHP artefact whose traces link elements of two blocks (e.g. requirements in an Excel file and the elements of a Simulink model). The source and target elements of a trace will be part of the source and target block, respectively, of its traceability module. Impact analysis could be performed based on the information of a traceability module. Finally, this RSHP-based approach enables the evaluation of the adequacy of a trace. First, two elements might be linked but a relationship might actually not exist between them, or it might not seem so. Second, a linked element could be modified, and its traces might be become invalid. Tools implementing the RSHP model could support such actions because the aim is to represent not only metadata about artefacts but also artefact content.

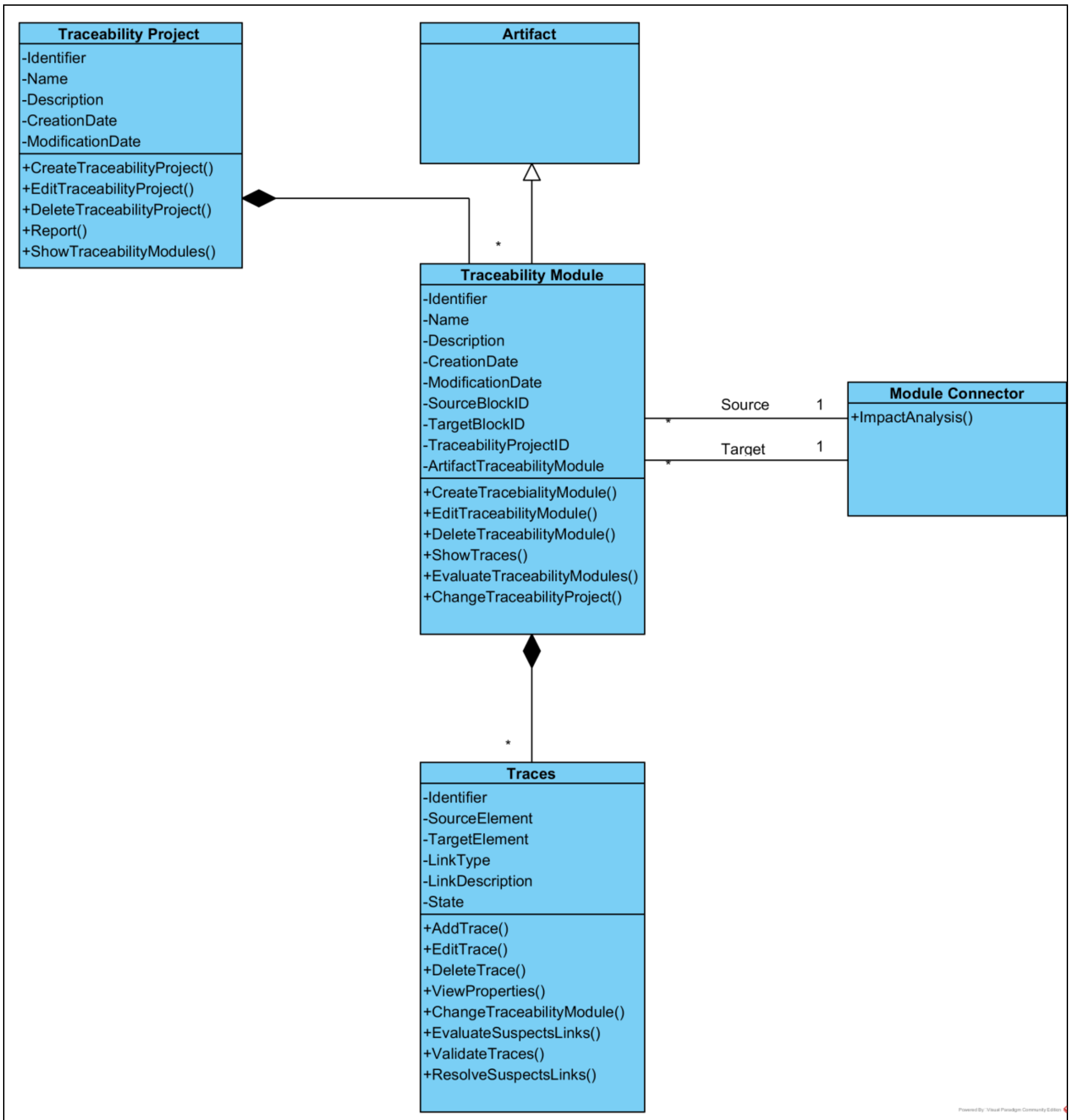


Figure 3.16 Main elements for Knowledge-centric traceability management [KCT1]

Relationship with other methods: Method complementary to *Traceability of safety software*, representing a specific way of addressing it. Results from all the other methods could be used as artefacts to trace.

Tool support: Traceability Studio (https://www.reusecompany.com/traceability-studio)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Semi-formal
- Type of component under evaluation: Software, Hardware, Model
- Evaluation tool: Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting

• Type of requirements under evaluation: Functional, Non-functional – Safety, Non-functional – Cybersecurity, Non-functional – Privacy, Non-functional – Others • Evaluation performance indicator: SCP criteria
Use case scenarios • VALU3S_WP1_ Automotive_10 - System certification • VALU3S_WP1_ Automotive_12 - ADAS system has to be reliable and has to comply with safety standards • VALU3S_WP1_ Healthcare_3 - Certification needs of the NMT device • VALU3S_WP1_ Healthcare_6 - Assurance needs of the NMT device
Strengths • Industrial method that is already in use • Method that can be applied for any system artefact type • Enabler of automated trace discovery via semantic analysis
Limitations • Several activities of the traceability management process [KCT2] are not supported • No empirical evidence of cost-effectiveness • The method could be improved by integrating with other approaches, e.g. model-based traceability management
References • [KCT1] AMASS Project: Deliverable D5.3 - Design of the AMASS tools and methods for seamless interoperability (b). 2018 • [KCT2] Cleland-Huang, J., Gotel, O. and Zisman, A. (eds.): Software and systems traceability. Heidelberg, Springer. 2012 • [KCT3] Llorens, J., Morato, J., Genova, G.: RSHP: an information representation model based on relationships. In Soft computing in software engineering. Springer. 2004
Related standards: Most assurance standards require that certain traces are maintained in a system's lifecycle, e.g. between requirements and tests. Examples of these standards include DO-178C, EN 50128, IEC 61508, and ISO 26262.
Keywords: Traceability, System artefact, Relationship, Knowledge representation, Ontology, Knowledge-centric systems engineering.

3.6.2.4 Model-Based Assurance and Certification

Name: **Model-based assurance and certification**
Purpose: To justify system dependability in compliance with standards by exploiting model-based techniques, e.g. structured specifications that conform to a metamodel.
Description: Many critical systems, e.g., safety-critical ones, are subject to rigorous assurance and certification processes to guarantee that the systems are dependable. Assurance can be defined as the set of planned and systematic actions necessary to provide adequate confidence and evidence that a system satisfies given requirements, e.g., for system safety, and certification can be defined as the legal recognition that a system complies with standards and regulations designed to ensure that the system can be depended upon to deliver its intended service.

Assurance and certification are challenging, time-consuming, and costly processes. A means that can facilitate them is the use of models, i.e., of representations that conform to a reference information structure (aka metamodel). Models can facilitate the understanding of safety standards, the identification of inconsistencies in their text, the determination of the evidence to collect, the specification of traceability requirements, and compliance assessment, among other tasks [MAC2]. We review model-based assurance and certification as characterised in the AMASS project [MAC1]. From a process perspective [MAC3], six main stages can be distinguished for model-based assurance and certification (Figure 3.17). Not every stage and step should be performed for each assurance project. In particular, the first two stages ("Standards Compliance Definition" and "Process Reusability Definition") are project-independent and only need to be performed once, so the outcome and data provided from these steps could be re-used for multiple projects.

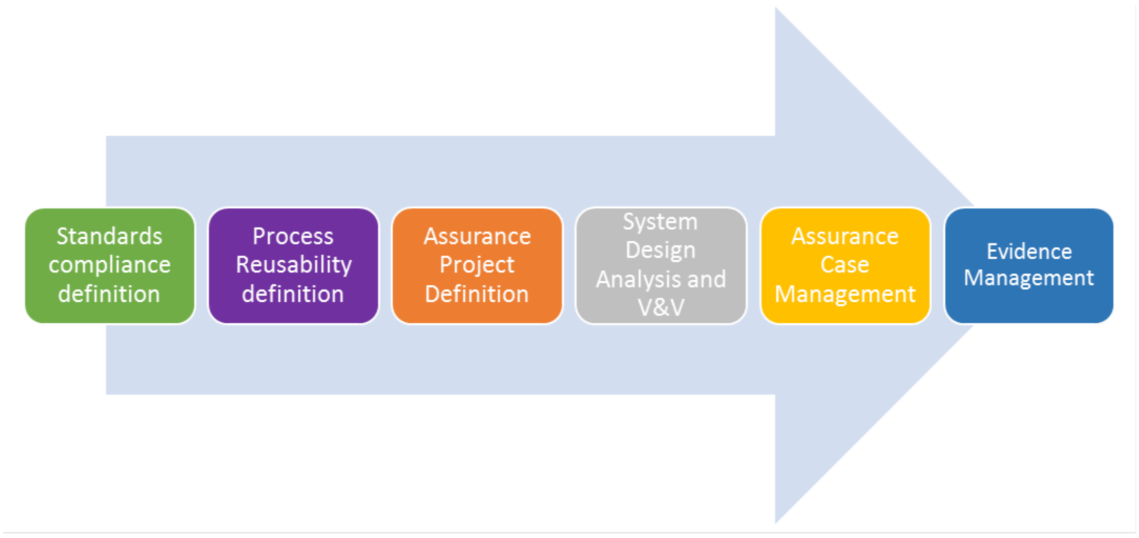

Standards compliance definition
Process Reusability definition
Assurance Project Definition
System Design Analysis and V&V
Assurance Case Management
Evidence Management

Figure 3.17 General process for model-based assurance and certification [MAC3]

- **Standards Compliance Definition** is a project-independent phase focused on capturing, digitalizing, storing and retrieving the different standard compliance knowledge. It should be performed by an expert in the regulatory frameworks that will be part of the reference knowledge included in the platform.
- **Process Reusability Definition** is conducted only once by a process expert. This expert will take care of tasks such as specifying reusable compliant processes and validating the process reusability.
- For **Assurance Project Definition**, the assurance manager defines the scope of compliance for a project in the context of a certain regulation. The manager will follow the project compliance lifecycle and, when it is feasible, check the different reuse possibilities and compliance means.
- The systems engineer performs **System Design Analysis and V&V** in collaboration with the safety and security engineers to define the system architecture, elicit system requirements, define component contracts, and conduct safety and security analyses. The validation of the components' contracts and V&V of safety and security analyses is performed by the V&V engineer.
- **Assurance Case Management** deals with the definition of argumentation using compliance arguments and product arguments. The assurance manager will take care of resolving safety and security trade-offs and of linking the assurance case information to the system architecture.

- During **Evidence Management**, the assurance manager will define the project artefacts that will be used as evidence and collect those artefacts. The manager will ensure artefact traceability, follow the progress of the process execution, and specify the compliance with standards and regulations.

Relationship with other methods: Model-based assurance and certification relates to all the other methods because their results could be used for assurance & certification purposes, e.g. as evidence.

Tool support: OpenCert (https://www.eclipse.org/opencert/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Semi-formal
- Type of component under evaluation: Software, Hardware, Model
- Evaluation tool: Open source
- Evaluation stage: Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-functional – Safety, Non-functional – Cybersecurity, Non-functional – Privacy, Non-functional – Others
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_ Automotive_10 - System certification
- VALU3S_WP1_ Automotive_12 - ADAS system has to be reliable and has to comply with safety standards
- VALU3S_WP1_ Healthcare_3 - Certification needs of the NMT device
- VALU3S_WP1_ Healthcare_6 - Assurance needs of the NMT device

Strengths

- Developed from large collaborative efforts in prior projects such as OPENCOSS (http://www.opencoss-project.eu/), SafeCer (https://cordis.europa.eu/project/id/295373), AMASS and (https://www.amass-ecsel.eu/).
- Extensive validation
- Already part of an Eclipse project
- Integrated with other methodologies and tools, and with integration extension features
- Considerable amount of methodological guidance available

Limitations

- Most often requires tailoring (selection of activities needed) to specific companies and projects
- Limited support for workflow configuration
- Tool support usability can be improved

References

- [MAC1] AMASS project: D2.5 - AMASS user guidance and methodological framework. 2018
- [MAC2] de la Vara, J.L., Ruiz, A., Attwood, K., Espinoza, H., Panesar-Walawege, R.K., Lopez, A., del Rio, I., Kelly, T.: Model-Based Specification of Safety Compliance Needs: A Holistic Generic Metamodel. Information and Software Technology 72: 16-30, 2016

- [MAC3] de la Vara, J.L., Ruiz, A., Gallina, B., Blondelle, G., Alaña, E., Herrero, J., Warg, F., Skoglund, M., Bramberger, R.: The AMASS Approach for Assurance and Certification of Critical Systems. embedded world Conference 2019

Related standards: In general, this method could be applied in the scope of any standard whose compliance with is necessary. Examples of standards for which the method has been applied include DO-178C, EN 50128, and ISO 26262.

Keywords: System assurance, System certification, Compliance, Model, Model-Driven Engineering.

3.6.2.5 Model-Based Design Verification

Name: **Model-based design verification**

Purpose: Verify correctness and dependability of models used to design the system under development. Models can represent requirements, architecture and/or behaviour of the system.

Description: Model-Based Design (MBD) is a method to address the design of complex systems with models. It prescribes the use of models through the development process in order to represent system requirements, architecture and behaviours, providing a basis for machine-assisted analysis of system properties, and supporting design decisions through processes of refinement into implementation. The purpose is:

1. To reduce the complexity of design by abstraction,
2. To ensure the quality of the system by rigorous analysis of its properties and
3. To reduce the cost of the development by detecting issues in the early phases of the development.

MBD is quite standard in software engineering and is becoming more and more relevant in system engineering, where it must integrate methods for control engineering and safety engineering [MBD1, MBD2, MBD3, MBD4, MBD5, MBD6].

Models can be analysed with different V&V methods. When the models are associated with a formal semantics, formal methods can be applied for a rigorous analysis of correctness and dependability properties. MBD V&V methods include: simulation, testing, model checking, model-based safety analysis, requirements analysis, schedulability analysis, performance analysis, design space exploration.

The models are typically described in high-level languages, usually with a graphical notation. These languages can be standardized (e.g., AADL, UML, SysML) or proprietary (e.g., Simulink).

Relationship with other methods: *Formal verification* methods, *Simulation* methods, *Model-Based Testing.*

Tool support: SCADE, Simulink, CHESS, TASTE, Autofocus3, COMPASS, Ptolemy and many more

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical – Formal, Analytical – Semi-formal
- Type of component under evaluation: Model
- Evaluation tool: Open Source, Proprietary
- Evaluation stage: Validation, Verification
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Functional, Non-Functional – Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios • VALU3S_WP1_Aerospace_1 - Robust and safe operation under sensor faults • VALU3S_WP1_Aerospace_2 - Robust operation under system parameter perturbation • VALU3S_WP1_Aerospace_3 - Robust operation under low probability hazardous events • VALU3S_WP1_Aerospace_4 - Robust fault detection, isolation and recovery
Strengths • Reduces the complexity of design by abstraction • Ensures the quality of the system by rigorous analysis of its properties • Reduces the cost of the development by detecting issues in the early phases of the development
Limitations • A formal semantics must be assigned to the models
References • [MBD1] Bouali, Amar & Dion, Bernard. (2005). Formal Verification for Model-Based Development. 10.4271/2005-01-0781. • [MBD2] Conrad, Mirko & Mosterman, Pieter. (2013). Model-Based Design Using Simulink - Modeling, Code Generation, Verification, and Validation. 159-181. 10.1002/9781118561898.ch4. • [MBD3] Antonio Cicchetti, Federico Ciccozzi, Silvia Mazzini, Stefano Puri, Marco Panunzio, Alessandro Zovi, Tullio Vardanega: CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. ASE 2012: 362-365 • [MBD4] Mazzini, S. & Baracchi, L. & Gérald, Garcia & Cimatti, Alessandro & Tonetta, Stefano. (2013). The FOREVER Methodology: a MBSE framework for Formal Verification. • [MBD5] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, Marco Roveri: The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. SAFECOMP 2009: 173-186 • [MBD6] Jeff C. Jensen, Danica H. Chang, Edward A. Lee: A model-based design methodology for cyber-physical systems. IWCMC 2011: 1666-1671
Related standards: -
Keywords: Model-based design, Model-based system and software engineering, Formal verification.

3.6.2.6 Traceability Management for Safety Software

Name: **Traceability management of safety software**
Purpose: To ensure that the software resulting from development activities meets the requirements for correct operation of the safety-related system, consistency between the software development stages is essential. This can be achieved by tracing the system artefacts of the different stages.
Description: Traceability between activities is an impact analysis to check: 1. Decisions that were made at an earlier stage are adequately implemented in later stages (forward traceability); 2. Decision that were made at a later stage are actually required and mandated by earlier decisions (backward traceability). Forward traceability is broadly concerned with checking that a requirement is adequately addressed in later software development stages, contributing to its confirmation. Forward traceability is valuable at several points in the safety software development, for instance:

• From the system safety requirements to the software safety requirements; • From the software safety requirements specification to the software architecture; • From the software safety requirements specification to the software design; • From the software design specification to the module and integration test specifications; • From the system and software design requirements for hardware/software integration to the hardware/software integration test specifications; • From the software safety requirements specification to the software safety validation plan; • From the software safety requirements specification to the software modification plan (including re-verification and re-validation); • From the software design specification to the software verification (including data verification) plan. Backward traceability is broadly concerned with checking that every implementation (interpreted in a broad context, and not confined to code implementation) decision is clearly justified by some requirement. If this justification is absent, then the implementation contains something unnecessary that adds to the complexity but not necessarily address any genuine requirement of the safety-related system. Backward traceability is valuable at several points in the safety software development, such as: • From the safety requirements, to the perceived safety needs; • From the software architecture, to the software safety requirements specification; • From the software detailed design to the software architecture; • From the software code to the software detailed design; • From the software safety validation plan, to the software safety requirements specification; • From the software modification plan, to the software safety requirements specification; • From the software verification (including data verification) plan, to the software design specification
Relationship with other methods: Results from the other methods can be subject to traceability management.
Tool support: CodeBeamer (Intland; https://codebeamer.com/), Spreadsheet (not advisable)
Layers of the multi-dimensional framework • Evaluation environment: In the lab • Evaluation type: Analytical – Semi-formal • Type of component under evaluation: Model • Evaluation tool: Proprietary • Evaluation stage: Validation • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Non-functional - Safety • Evaluation performance indicator: SCP criteria
Use case scenarios • VALU3S_WP1_Agriculture_1 - Vehicle switching from parallel guidance to manual mode • VALU3S_WP1_Agriculture_2 - Vehicle switching from manual mode to parallel guidance
Strengths • Requirement coverage assurance • Robustness of the design

Limitations • Time-consuming (most often; it also depends on the tool support)
References • [TMS1] Cleland-Huang, J., Gotel, O. and Zisman, A. (eds.): Software and systems traceability. Heidelberg, Springer. 2012
Related standards: ISO19014
Keywords: Traceability.

3.7 System-Type-Focused V&V

This group of methods tackles general or several V&V needs specific to certain system types, typically covering different V&V areas, e.g., formal verification and testing, that can be combined for system V&V as a whole. Therefore, this group complements the previous ones by presenting wider V&V needs for specific situations, e.g. in the scope of some VALU3S use case. In this sense, the methods below can correspond to larger aspects of V&V processes that need to be reviewed in the project to set their current status, including their strengths and limitations.

The V&V method reviews below deal with CPUs, industrial systems, and robotic systems as specific system types. For robotic systems, failure analysis aspects, model-based techniques, and formal methods are considered.

3.7.1 CPU Verification

Name: **Central Processing Unit (CPU) verification**
Purpose: The purpose of Central Processing Unit (CPU) verification is to ensure that CPUs, which are the central part of modern cyber-physical systems, deliver their functionality correctly and as intended**.**
Description: Nowadays it is quite common that Cyber-Physical Systems (CPS) comprise processor sub-systems for executing domain-specific software applications. Designing processors is a hard task having high complexity, time-consuming development cycles, high costs and economic risks. Therefore, in hardware design (Application Specific Integrated Circuits (ASIC), Field-Programmable Gate Arrays (FPGA), Systems-On-Chip FPGA) processors with surrounding peripherals (e.g., caches) and integration test suits are rather bought than developed in-house. The recent developments in royalty free and open-source instruction set architectures (ISA) have motivated the development of various RISC-V designs, which range from freely licensed open-source cores to proprietary designs. CPU designs off-the-shelf may include test suites as well – however, if the designs are quite new the question of trust in the promised quality of verification completeness and correctness remains. Cost-free cores might unluckily come without proper verification. Next to rigorous verification techniques and tools for processor designs comes the ability to adapt the verification to new situations such as the evolution of the RISC-V ISA. In safety-critical applications, proven-in use processor cores are the natural candidates with the drawback of included license costs. Core designs based on a relatively new ISA (RISC-V) demand

proper verification to be applicable in commercial solutions and have thus not reached the point when proven-in-use may be used as an argument.

One major challenge for CPU verification concerns ISA compliance. First, a reference model is needed and second, since all combinations of instructions cannot be verified in reasonable time using simulation-based verification, trade-offs must be made. Next to ensuring that software written for a specific ISA (RISC-V ISA selection) works correctly on the CPU design is to verify the CPU design parts related to memory, load/store, paging, caching, pipelining, etc.

There are two main approaches to face the verification of a CPU, these are simulation-based verification and formal verification. The first is the most frequently used method, it can provide certain insurance that common functionality as well as corner cases were exercised, often measured using coverage metrics like functional coverage and structural/code coverage [CPU5]. Among the benefits of simulation-based verification are its high execution speed and the need to develop a model for checking proper behaviour that can also be used for software verification and virtual prototyping. However, an extensive test suite has to be manually written, which is a time-consuming task and more important it is not complete and therefore cannot prove compliance to a certain specification [CPU4]. On the other hand, formal verification can prove that a set of properties and assertions holds for all possible scenarios using formal engines based on mathematical techniques e.g., property checking (also called model checking), this is of vital importance for safety critical requirements. The drawback of formal verification is that, depending on the formal engine, it may not be able to provide proofs for large designs, but modern tools claim to be able to process a CPU design (e.g., [CPU6]). In the case of RISC-V, the ISA specifications must be translated into properties or assertions in the language that the formal verification tool uses, e.g., SystemVerilog Assertions. A RISC-V verification would take advantage of both verification methodologies depending on the requirements.

CPU design verification could include verification planning (define what and how should be verified), definition/development of Instruction Set Simulator (ISS) for CPU ISA selection plus custom instructions, simulation-based verification i.e. Constrained Random Verification with self-checking capabilities (monitors and checkers) using universal verification methodology (UVM), functional coverage collection and structural coverage measurement, and apply formal verification to ensure properties of the design.

An approach for the needed main ingredients for a CPU (RISC-V) verification flow is presented in [CPU1].

For CPU verification several approaches have been subject to research and development, such as the following examples regarding RISC-V:

- Formal verification, e.g., the RISC-V ISA Formal Proof Kit by axiomise [CPU2], riscv-formal [CPU7], OneSpin 360 DV RISC-V Verification App [CPU8]
- RISC-V verification on the RTL level [CPU3]
- Model-based test generation with the Scala-based Torture Test framework for RISC-V [CPU9]

Relationship with other methods: *Model Checking*, *Source Code Static Analysis*, *Test Optimization for Simulation-based Testing of Automated Systems*, *Model-Based Safety Analysis*, *Simulation-based Fault Injection at System-level*.

Tool support: -

Layers of the multi-dimensional framework

- Evaluation environment: In the lab

- Evaluation type: Experimental - Testing, Experimental - Simulation, Analytical - Formal
- Type of component under evaluation: Hardware, Model
- Evaluation tool: Open source, Proprietary
- Evaluation stage: Verification
- Logic of the component under evaluation: Thinking
- Type of requirements under evaluation: Functional, Non-functional - Safety, Non-functional - Cybersecurity, Non-functional - Others
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_5 - Motor speed control
- VALU3S_WP1_Industrial_6 – Fault tolerance for motor position data
- VALU3S_WP1_Industrial_7 – Safety behaviour for missing motor value position data
- VALU3S_WP1_Industrial_8 – Safety behaviour for remote control terminal connection failure
- VALU3S_WP1_Industrial_9 – Safety/Security behaviour for corrupted data from remote control terminal

Strengths

- CPU Verification enables the application of new CPU designs
- A method for CPU verification is ISA Formal Verification (Model Checking) for checking ISA compliance
- A method for CPU verification is Verification on RTL level enabling functional and structural tests

Limitations

- CPU Verification is a very expensive activity and takes a lot of time.
- Complete verification might not be reachable (100% coverage, etc.), trade-offs such as bound-proofs might be necessary

References

- [CPU1] Bipul Talukdar, RISC-V's CPU Verification Challenge, https://www.eeweb.com/risc-vs-cpu-verification-challenge/, 03.09.2020.
- [CPU2] Ashish Darbari, Axiomising RISC-V processors through formal verification, https://www.axiomise.com/risc-v-formal-verification.
- [CPU3] V. Herdt, D. Große, E. Jentzsch and R. Drechsler, "Efficient Cross-Level Testing for Processor Verification: A RISC- V Case-Study," 2020 Forum for Specification and Design Languages (FDL), Kiel, Germany, 2020, pp. 1-7, doi: 10.1109/FDL50818.2020.9232941.
- [CPU4] Nicolae Tusinschi, A Holistic View Of RISC-V Verification, https://semiengineering.com/a-holistic-view-of-risc-v-verification/, 07.08.2019.
- [CPU5] Mentor, A Siemens Business, Coverage Cookbook, https://verificationacademy.com/cookbook/coverage/.
- [CPU6] OneSpin, Assuring the integrity of RISC-V Cores and SoCs, https://www.onespin.com/resources/white-papers/, 2019.
- [CPU7] riscv-formal, https://github.com/SymbioticEDA/riscv-formal.
- [CPU8] OneSpin DV RISC-V Verification App, https://www.onespin.com/solutions/risc-v, 02/2020.
- [CPU9] risc-v torture, https://github.com/ucb-bar/riscv-torture.

Related standards: -
Keywords: CPU, verification, RISC-V, Formal verification, RTL-level verification, Model-based verification.

3.7.2 Penetration Testing of Industrial Systems

Name: **Penetration testing of industrial systems**
Purpose: Analysis of sensor data and server-PLC communication to evaluate the system robustness in the case of sensor data manipulation and to evaluate effects of data manipulation in communication between server and PLC, including several attack types such as man in the middle (MiTM), Denial of Service (DoS) and Address Resolution Protocol (ARP) Poisoning.
Description: There are plenty of different techniques in data manipulation where MiTM, DoS and ARP poisoning are emerging and commonly exploited. MiTM or called with other name person-in-the-middle (PITM) is a cyber-attack technique. Basically, in this technique, the attacker positioning himself between two sides of communication for listening and resolving any information in communication [DMD1]. DoS (a Denial-of-Service) attack is a cyber-attack in which the perpetrator aims to make a machine or network resource unavailable to its intended users by temporarily or permanently disrupting services of a host connected to the Internet [DMD2, DMD3]. ARP is a communication protocol for link layer in ISO reference model at RFC 826 [DMD1]. ARP Poisoning is called with different names like, ARP spoofing, ARP cache poisoning, or ARP poison routing. It is a technique by which an attacker sends (spoofed) ARP messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead [DMD1, DMD4]. Industrial systems can be tested to detect these issues.
Relationship with other methods: *Assessment of cybersecurity-informed safety.*
Tool support: Open source tools for MiTM, DoS, ARP Poisoning attacks such as ARP Spoofing, Wireshark (https://www.wireshark.org/), Ettercap (https://github.com/Ettercap/ettercap), Greenbone (https://www.openvas.org/), Metasploit (https://github.com/rapid7/metasploit-framework)
Layers of the multi-dimensional framework • Evaluation environment: Open • Evaluation type: Experimental - Testing • Type of component under evaluation: Hardware, Software • Evaluation tool: Open Source • Evaluation stage: Validation, Verification • Logic of the component under evaluation: Sensing, Thinking, Acting • Type of requirements under evaluation: Non-functional - Cybersecurity • Evaluation performance indicator: V&V process criteria, SCP criteria
Use case scenarios • VALU3S_WP1_Industrial_1 - Manipulation of sensor data • VALU3S_WP1_Industrial_2 - Server and PLC communication • VALU3S_WP1_Industrial_4 - Anomaly detection at component and system level

Strengths • Ability to apply real world attacks • Relatively shorter test duration compared to model-based and simulation-based approaches
Limitations • Test can be carried out after full commissioning • Possible side effects on other IT systems • Not include zero-day vulnerabilities
References • [DMD1] Yang, Y., McLaughlin, K., Littler, T., Sezer, S., Im, E. G., Yao, Z. Q., & Wang, H. F. (2012).: Man-in-the-middle attack test-bed investigating cyber-security vulnerabilities in smart grid SCADA systems. • [DMD2] Bechtsoudis, A., & Sklavos, N. (2012). Aiming at higher network security through extensive penetration tests. IEEE latin america transactions, 10(3), 1752-1756. • [DMD3] Denial-of-service attack. https://en.wikipedia.org/wiki/Denial-of-service_attack • [DMD4] Denis, M., Zena, C., & Hayajneh, T. (2016, April). Penetration testing: Concepts, attack methods, and defense strategies. In 2016: IEEE Long Island Systems, Applications and Technology Conference (LISAT) (pp. 1-6). IEEE.
Related standards: ISO/IEC 27002:2013
Keywords: Pen-test, ISO 27002.

3.7.3 Failure Detection and Diagnosis (FDD) in Robotic Systems

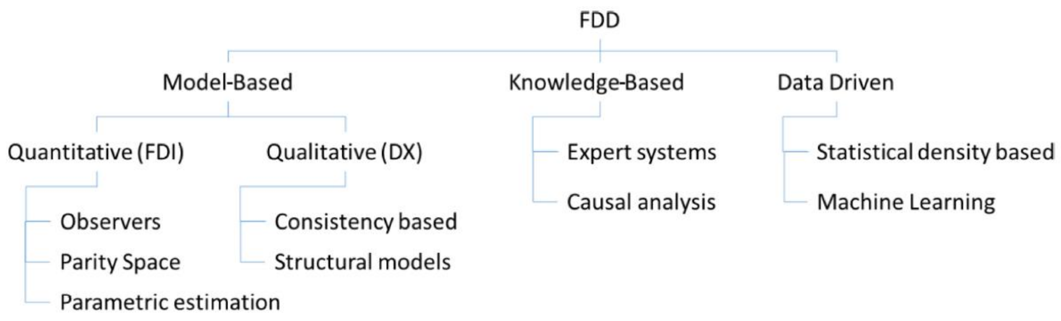
Name: **Failure detection and diagnosis (FDD) in robotic systems**
Purpose: Detect and diagnose failures in Robotic System components provoked by failure injection
Description: In the industrial robotics domain, faults have the potential to affect the efficiency of the underlying process, namely causing failures of internal physical components (e.g. robot, IPC, sensors, actuators), or even compromising the safety of humans interacting with the robot. When detecting a fault, usually a diagnosis process is induced in order to identify which internal components are involved. Applying FDD for industrial robotics is a relatively new approach, thus, there exists a wide range of different types of industrial robots, and on the other hand there exist different FDD approaches such as data-driven, model-based, and knowledge-based approaches. FDD approaches can be distinguished into data-driven, model-based, and knowledge-based approaches [FDD1]. Data-driven approaches for instance are based on near real-time process data with the aim of statistically differentiating a potential fault from historical data, e.g., via clustering techniques such as Figure 3.18 highlights a classification of existing FDD approaches.  FDD Model-Based Knowledge-Based Data Driven Quantitative (FDI) Qualitative (DX) Observers Parity Space Parametric estimation Consistency based Structural models Expert systems Causal analysis Statistical density based Machine Learning

Figure 3.18 Classification of FDD approaches

Detecting faults emerging from external sensors of highly dynamic HRI work environments is mandatory as faults may lead to incorrect decisions within the robotic control unit and eventually to unexpected behaviours of the collaborative robot. Analytical or stochastic a priori models are particularly used in respect to internal sensors of a robotic system when the system operates in a well-known work environment. For robotic systems operating in unknown environments, predicting the values of external sensors is hardly possible. For these situations data-driven approaches are the better choice by applying sensor fusion techniques for external sensor fault detection. This means that multiple sensors sense different aspects of the environment (e.g. orientation and location), while their readings can be fused to form a consensus. Sensor-fusion-based fault detection approaches for robotic systems include different algorithms such as: Kalman filters, Dempster-Shafer, correlation and distribution-based, and Bayesian networks.

The figure below (Figure 3.19) illustrates a robotic system which is composed out of hardware and software modules. Hardware components such as internal sensors, power, controller, and actuators might be subject to different hardware faults, which are typically diagnosed by model-based or knowledge-based diagnosis approaches. However, for the internal and external (exteroceptive) sensors (bottom left), data-driven approaches are more appropriate for FDD; in particular, sensor fusion for sensor fault detection, and data analytics techniques such as machine learning and particle filtering for utilizing sensor-readings for diagnosis.

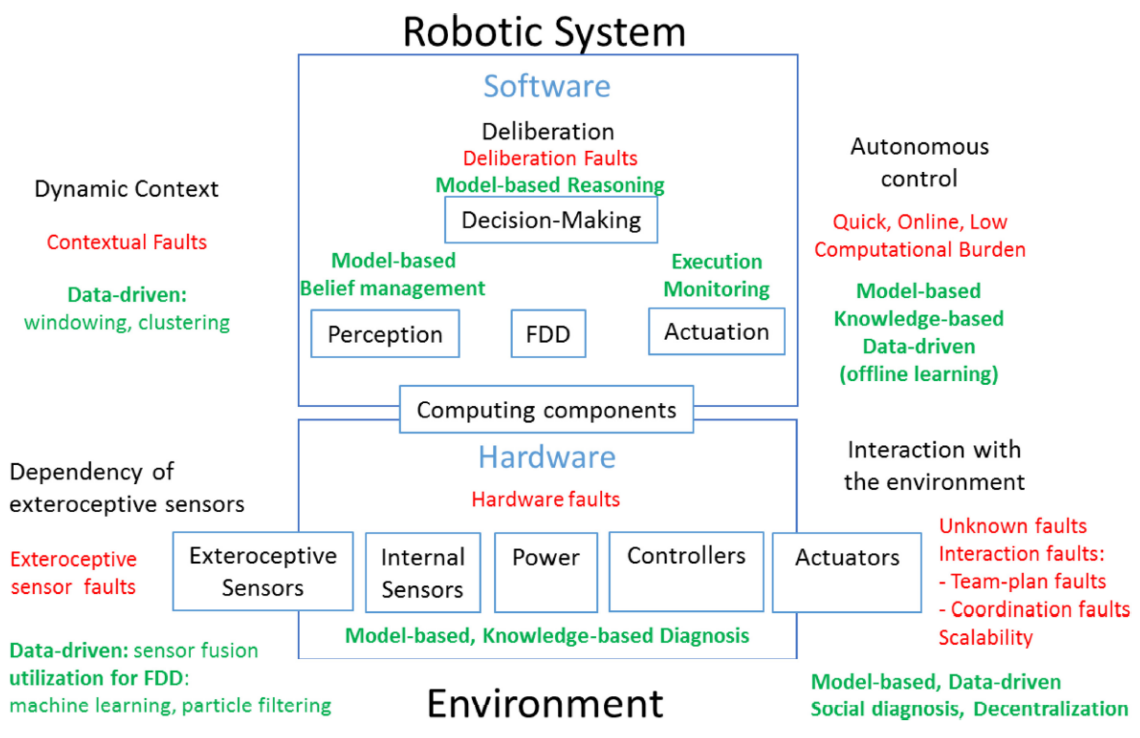


Figure 3.19 Components of a robotic system and mapping of FDD approaches

Relationship with other methods: *Fault injection, Machine learning model validation, Model-based testing.*

Fault Injection (simulation-based/hardware-based) is definitely one of the testing methods directly related to robotic systems, particularly in the area of collaborative robotics. Relying on data-driven approaches in the context of FDD, a relationship with machine learning techniques (Machine Learning Model Validation) such as supervised learning is apparent. Supervised learning induces expressions that are sensed by the external sensors and form the basis of creating training data sets.

Usually, supervised learning algorithms such as clustering and regression are applied for these purposes. Sophisticated FDD approaches must therefore also consider techniques such as dimensionality reduction that are capable of extracting the most essential data features. Apart from this, there will also be a relationship to Model-Based Testing methods. When FDD is conducted by coupling HW components of the robotic system with a simulation environment, "hardware/Software-in-the-loop" (HiL, SiL) represent other methods which are related to the main methods in the use case.
Furthermore, relationship or interdependences are assumed with injection-based methods.

Tool support: CIROS Studio by FESTO Didactic (https://www.festo-didactic.com/de-de/lernsysteme/software-e-learning/ciros/ciros-studio-virtuelle-lernumgebungen-erstellen.htm?fbid=ZGUuZGUuNTQ0LjEzLjE4LjExMTAuODE4Ng), FERAL from FHG IESE (https://www.iese.fraunhofer.de/en/services/virtual-architecture-development-and-evaluation.html), and PYTHON (programming language)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab (virtual and physical)
- Evaluation type: Experimental - Simulation, Experimental – Monitoring, Analytical – Semi-formal
- Type of component under evaluation: Model, Software, Hardware
- Evaluation tool: Proprietary
- Evaluation stage: Verification, Validation
- Logic of the component under evaluation: Sensing, Thinking, Acting
- Type of requirements under evaluation: Non-functional - Safety, Non-functional - Cybersecurity, Non-functional – Others (Reliability, Throughput, Data Integrity, Performance, Robustness, Operability)
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_4 - Human-Robot-Interaction in Semi-Automatic Assembly Processes
- VALU3S_WP1_Industrial_7 - Human-Robot Collaboration in a disassembly process with workers with disabilities
- VALU3S_WP1_Industrial_11 - Automated robot inspection cell for quality control of automotive body- in-white

Strengths

- Combination of FDD with advanced AI techniques such as machine learning techniques to provide a useful and sophisticated diagnosis model.

Limitations

- Deficit for using FDD approaches (particularly data-driven approaches) that are dedicated to detect and diagnose HRI-related faults. HRI is subject to uncertainty due to the fact that unexpected outcomes might lead to unknown faults and failed interactions.
- Interaction-related faults between humans and robots, humans may have the tendency to compensate for a faulty behaviour of a robot during interaction.
- A robotic systems may provide a lot of data per se, thus, pure, data-driven approaches do not tend to exploit existing and available knowledge about the robotic system. As such, mining streamed data in an online manner for fault detection may be impractical for some reasons, and

faults might not be detected immediately. Thus, when using fault injection (simulated or real), one cannot confidently account for all possible faults.

- It is further a limitation when injecting a fault directly into a data stream since this strategy might not sufficiently represent the full impact of the fault on the whole system. Moreover, injecting a real fault may damage the robotic system, thus it has to be ensured that FDD experiments and tests are continuously supervised.

References

- [FDD1] Eliahu Khalastchi and Meir Kalech. 2018. On Fault Detection and Diagnosis in Robotic Systems. ACM Comput. Surv. 51, 1, Article 9 (January 2018), 24 pages.
- [FDD2] Juez Uriagereka, Garazi & Amparan, Estibaliz & Martinez, Cristina & Martinez, Jabier & Ibanez, Aurelien & Morelli, Matteo & Radermacher, Ansgar & Espinoza, Huáscar. (2019). Design-Time Safety Assessment of Robotic Systems Using Fault Injection Simulation in a Model-Driven Approach. 577-586. 10.1109/MODELS-C.2019.00088.

Related standards: EN ISO TS 15066, ISO 10218-2, ISO 13849, ISO 12100, ISO 61508

Keywords: FDD, Machine learning, Supervised learning, Unsupervised learning, Data stream processing, Hardware-in-the-loop, Sensor fusion, HRI.

3.7.4 Model-Based Formal Specification and Verification of Robotic Systems

Name: **Model-based formal specification and verification of robotic systems**

Purpose: Enabling formal verification of robotic systems by developing models that cope with the intractable state space of complex robotic system software, and improving the verification coverage and assurance on the operation of robotic systems by using the formal methods in combination with runtime verification.

Description: The software of autonomous robot systems is generally complex and safety-critical. Therefore, formal specification and verification is a rather challenging task. Widely used testing and simulation alone are insufficient to ensure the correctness and safety of the system [MBF1, MBF2]. Although the software of robot systems is complex, fortunately, it can be designed as interacting but separate modules. Many people in the robotics research community use the robot operating system (ROS) which is a framework. Due to the modular structure of robot software, ROS allows the control of a robot to be carried out by functional modules operating as separate processes. As presented in Figure 3.20 (a), a simple control scheme of a mobile robot system has multiple functional modules. However, although the robotic inspection system (UC11) is targeted for verification, many industrial robot systems perform similar stages (Figure 3.20 (b)). Thus, the realization of the tasks performed by industrial robot systems takes place in a hierarchical series of stages. If the robot autonomy is increased, such as re-planning when the environment state changes and moving by feedback from the sensors that detect the environment during the movements, there may be a mutual flow between the stages and stages can become more complex.

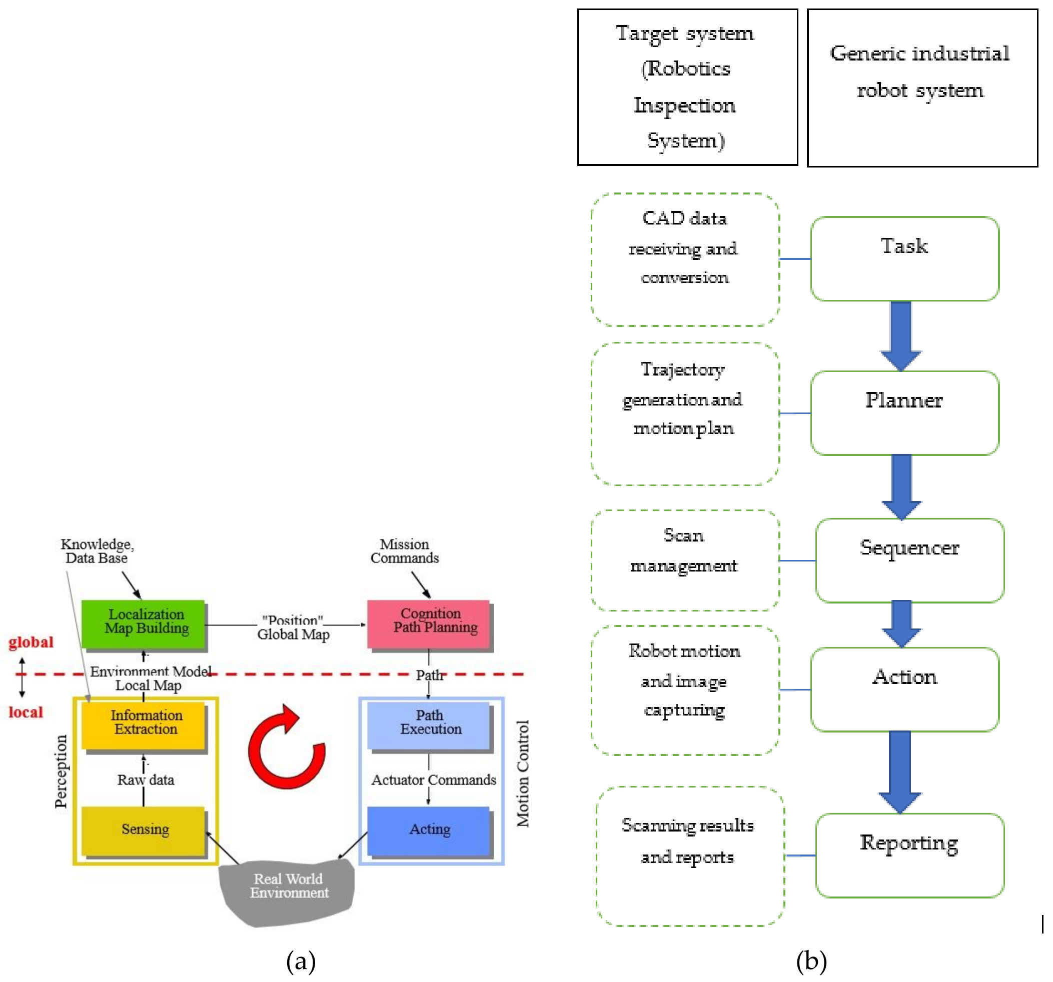


Figure 3.20 Examples of the modular structure of robotic system software: (a) General Control Scheme for mobile robot systems [BMF3], (b) General Control Scheme for Industrial robot system within UC11

In the formal verification of robotic systems with *model checking* methods, the widely preferred and suitable model is timed automata [MBF5, MBF6]. However, *run-time verification* methods are applied to robotics systems. Users provide formal safety and security specifications, and monitors are automatically generated and incorporated into the system to ensure the safety and security of robots [MBF7]. Model-checking, when combined with runtime verification, will improve the verification coverage and assurance on the operation of robotic systems [MBF4]. Verification methods to be used to verify the safety of robotic systems must comply with standards such as ISO 12100, IEC 61508, ISO 13849, ISO 10218-1, ISO 10218-2, ISO TS 15066 [MBF9, MBF10].

Relationship with other methods: *Simulation-based robot verification*, Model *checking*, *Runtime verification*.

Tool support: ROS (https://www.ros.org/), UPPAAL (https://www.uppaal.com/), ROSRV (https://github.com/Formal-Systems-Laboratory/ROSRV), GAZEBO (http://gazebosim.org/)

Layers of the multi-dimensional framework

- Evaluation environment: In the lab
- Evaluation type: Analytical - Formal, Experimental - Monitoring
- Type of component under evaluation: Model, Software
- Evaluation tool: Open source, Proprietary
- Evaluation stage: Verification

- Logic of the component under evaluation: Sensing, Acting, Thinking
- Type of requirements under evaluation: Functional, Non-functional - Safety
- Evaluation performance indicator: V&V process criteria, SCP criteria

Use case scenarios

- VALU3S_WP1_Industrial_1 - Manipulation of sensor data
- VALU3S_WP1_Industrial_3 - Safety trajectory optimization
- VALU3S_WP1_Industrial_4 - Anomaly detection at component and system level

Strengths

- Less time-consuming comparing to simulation-based for limited state space.
- Once the system model and property specification are provided to the model checker, the verification process is fully automatic.
- Long simulation runs are not required to obtain good coverage
- The verification results do not rely on the quality of the test cases; in fact, the algorithms prove correctness for all test cases.
- Require minimal human intervention on verification process. So, human error is minimized.

Limitations

- The system should be small in terms of the number of states to be processed by the model checker.
- Errors in specification. An implementation verification requires two versions of a design. Therefore, a bug in the specifications voids the reference model, may escape implementation verification, and may trickle down to implementations. A typical type of specification error is missing specifications.
- Incomplete functional coverage of specifications. Formal methods verify properties that, of course, can be specifications. In practice, however, it is rare that all specifications are given to a formal checker on the complete design, because of memory and runtime limitations. A partial specification is checked against the relevant portion of the design. Therefore, it leaves the door open to errors resulting from not checking all properties mandated by the specifications and from situations when, for example, a property succeeds on a sub circuit but does not succeed on the whole circuit.
- User errors. Example user errors are incorrect representation of specifications and over constraining the design.
- Formal verification software bugs. Bugs in formal verification software can miss design errors and thus give false confirmation. It is well-known that there is no guarantee for a bug-free software program.
- Formal specification languages need to be adopted for Autonomous Industrial robots
- A lack of clear guidance when it comes to choosing a suitable form for a particular system.

References

- [MBF1] Luckcuck, M., Farrell, M., Dennis, L., Dixon, C., Fisher, M. (2019), Formal Specification and Verification of Autonomous Robotic Systems: A Survey, ACM Computing Surveys, 52: 1-41.
- [MBF2] Ingrand, F., (2019), Recent Trends in Formal Validation and Verification of Autonomous Robots Software. 2019 Third IEEE International Conference on Robotic Computing (IRC).
- [MBF3] Siegwart, R. and Nourbakhsh, I, (2004), Introduction to Autonomous Mobile Robots, MIT Press.

• [MBF4] Desai, A., Dreossi, T., Seshia, S.A., (2017), Combining Model Checking and Runtime Verification for Safe Robotics, Runtime Verification RV 2017, LNCS 10548, pp. 172–189. • [MBF5] Wang, R., Guan, Y., Song, H., Li, X., Li, X., Shi, Z., Song, X., (2018), A Formal Model-Based Design Method for Robotic Systems, IEEE Systems Journal, PP: 1-12. • [MBF6] Halder, R., Proença, J., Macedo, N., Santos, A (2017), Formal Verification of ROS-Based Robotic Applications Using Timed-Automata, 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE). • [MBF7] Huang, J., Erdogan, C., Zhang, Y., Moore, B., Luo, Q., Sundaresan, A., Rosu, G., (2014), ROSRV: Runtime Verification for Robots, International Conference on Runtime Verification, Springer. 8734: 247-254. • [MBF8] W.K. Lam, Hardware Design Verification: Simulation and Formal Method-Based Approaches, Pearson Publishing, 2005. • [MBF9] Villani, V. P., F. Leali, F. Secchi, C. (2018), Survey on human-robot collaboration in industrial settings: Safety, intuitive interfaces and applications, Mechatronics, 55: 248-266. • [MBF10] Jérémie Guiochet, M. M., Hélène Waeselynck (2017). "Safety-critical advanced robots: A survey." Robotics and Autonomous Systems, 94: 43-52.
Related standards: ISO 12100, IEC 61508, ISO 13849, ISO 10218-1, ISO 10218-2, ISO TS 15066
Keywords: Formal verification, Model checking, Runtime verification, Simulation, Robotic system.

Chapter 4 Conclusion

This deliverable reviews state-of-the-art and state-of-the-practice methods that are relevant to VALU3S for SCP requirements evaluation. The review provides, on the one hand, a wide picture of V&V needs for SCP of automated systems and of possible ways to meet the needs. On the other hand, the review provides details about the characteristics of the methods, how they can be applied, what benefits they have, and what improvement opportunities exist in their usage. All in all, the review presented contributes to extending the body of knowledge about V&V methods by providing a new analysis focused specifically on VALU3S needs and expectations. It is also a reference for others interested in knowing how V&V methods can map to the multi-dimensional layered framework and to VALU3S use case scenarios.

In total, 53 methods have been reviewed, considering 13 groups and refining what was proposed in VALU3S project proposal according to the latest needs and interests in the project, e.g. in the use cases. The methods most often fall into the scope of In the lab as Evaluation Environment, Experimental - Testing as Evaluation Type, Software as Type of Component under Evaluation, Proprietary as Evaluation Tool, Verification as Evaluation Stage, Thinking as Logic of the Component under Evaluation, Non-functional - Safety as Type of Requirement under Evaluation, and SCP criteria as Evaluation Performance Indicator. The work on these methods in the next WP3 tasks will contribute to VALU3S KPI3 "Improve at least 14 V&V methods in order to create VALU3S repository of improved V&V methods". The achievement of this KPI will result in either further strengths or fewer limitations in the methods.

The results presented in D3.1 pave the way towards the analysis of possible gaps in SCP V&V methods, the development of new methods, and the improvement of existing methods in the next WP3 tasks. Information from the review will also be included in VALU3S' web-based repository of V&V solutions. The data presented is also subject to update as the input from other tasks evolve, most notably the multi-dimensional layered framework and the use case evaluation scenarios.

Last but not least, the work on SCP V&V methods in VALU3S is not an isolated effort, but it relates to other projects and initiatives. This includes both prior projects whose results are a basis for VALU3S, e.g. AMASS (https://www.amass-ecsel.eu/), and ongoing projects with which synergies exist or could be defined, e.g. iRel4.0 (https://www.irel40.eu/).

References

1. AMALTHEA project: Deliverable D3.1 – Analysis of state of the art V&V techniques. 2015
2. A. Avizienis, J.-C. Laprie, B. Randell et al., Fundamental Concepts of Dependability. University of Newcastle, Computing Science, 2001.
3. DoD: Defense Modeling & Simulation Coordination Office, V&V Techniques. Online, https://vva.msco.mil/default.htm?Ref_Docs/VVTechniques/default.htm, 2001
4. Engel, A.: Verification, Validation, and Testing of Engineered Systems. Wiley, 2010
5. ESA: PSS-05-10 - Guide to software verification and validation. 1995
6. IEC: IEC 61508-7, Functional safety of electrical/electronic/programmable electronic safety-related Systems - Part 7: Overview of techniques and measures. 2nd ed. 2010
7. IEEE: IEEE Std 1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation. 2017
8. IEEE: SWEBOK V3.0 – Guide to the Software Engineering Body of Knowledge. 2014
9. INCOSE: Systems Engineering Handbook – A Guide for System Life Cycle Processes and Activities. 4th ed. 2015
10. ISO: ISO 26262-6, Road vehicles – Functional safety – Part 6: Product development at the software level. 2nd ed. 2018
11. ISO: ISO/IEC/IEEE 24765, Systems and software engineering —Vocabulary. 2nd ed. 2017
12. Leucker, M., Schallhart, C.: A brief account of runtime verification. JLAMP, 78(5):293-303, 2009
13. Nair, S., de la Vara, J.L., Sabetzadeh, M., Briand, L.: An Extended Systematic Literature Review on Provision of Evidence for Safety Certification. Information and Software Technology 56(7): 689-717, 2014.
14. Oxford UK Dictionary: method. Online, https://www.lexico.com/definition/method, 2020
15. VALU3S project: Deliverable D1.1 - Description of use cases as well as scenarios. 2020
16. VALU3S project: Deliverable D1.2 - SCP requirements as well as identified test cases. 2020
17. VALU3S project: Deliverable D2.1 - Initial multi-dimensional layered framework. 2020
18. VALU3S project: Deliverable D2.2 - Final multi-dimensional layered framework. 2020
19. VALU3S project: Deliverable D6.5 - Initial report on the results of the standardisation survey (methods, tools, concepts suggested by the standards). 2020
20. Wallace, D.R., Fujii, R.U.: Software verification and validation: an overview. IEEE Software 6(3): 10-17, 1989

Appendix A Mapping of the V&V Methods to the Multi-Dimensional Layered Framework

Table A.1 V&V Method classification according to the Evaluation Environment dimension of the VALU3S framework

In the lab	Assessment of cybersecurity-informed safety
	Behaviour-driven formal model development
	Behaviour-driven model development and test-driven model review
	Code design and coding standard compliance checking
	CPU verification
	Deductive verification
	Dynamic analysis of concurrent programs
	Failure detection and diagnosis (FDD) in robotic systems
	Fault injection in FPGAs
	Formal requirements validation
	Human Interaction Safety Analysis (HISA)
	Interface fault injection
	Intrusion detection for WSN based on WPM state estimation
	Kalman filter-based fault detector
	Knowledge-centric system artefact quality analysis
	Knowledge-centric traceability management
	Machine learning model validation
	Model checking
	Model-based assurance and certification
	Model-based design verification
	Model-based formal specification and verification of robotic systems
	Model-based mutation testing
	Model-based robustness testing
	Model-based safety analysis
	Model-based testing
	Model-based threat analysis
	Model-Implemented Attack injection
	Model-Implemented Fault injection
	Reachability-analysis-based verification for safety-critical hybrid systems
	Risk analysis
	Risk-based testing
	Runtime verification based on formal specification
	Signal analysis and probing
	Simulation-based attack injection at system-level
	Simulation-based fault injection at system-level
	Simulation-based robot verification

In the lab (cont.)	Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Traceability management of safety software V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Vulnerability analysis of cryptographic modules against hardware-based attacks Vulnerability and attack injection Wireless interface network security assessment
Closed	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Model-based mutation testing Model-based robustness testing Model-based testing Risk-based testing Runtime verification based on formal specification Signal analysis and probing Test oracle observation at runtime
Open	Model-based mutation testing Model-based robustness testing Model-based testing Penetration testing of industrial systems Risk-based testing Runtime verification based on formal specification Signal analysis and probing Test oracle observation at runtime

Table A.2 V&V Method classification according to the Evaluation Type dimension of the VALU3S framework

Experimental – Simulation	Assessment of cybersecurity-informed safety CPU verification Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs

Experimental – Simulation (cont.)	Kalman filter-based fault detector Model-Implemented Attack injection Model-Implemented Fault injection Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-based robot verification Simulation-based testing for human-robot collaboration Test optimization for simulation-based testing of automated systems V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts
Experimental – Testing	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review CPU verification Fault injection in FPGAs Interface fault injection Machine learning model validation Model-based mutation testing Model-based robustness testing Model-based testing Penetration testing of industrial systems Risk-based testing Signal analysis and probing Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Test parallelization and automation Vulnerability and attack injection Wireless interface network security assessment
Experimental – Monitoring	Behaviour-driven model development and test-driven model review Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Intrusion detection for WSN based on WPM state estimation Model-based formal specification and verification of robotic systems Runtime verification based on formal specification Simulation-based robot verification Test oracle observation at runtime
Analytical – Formal	Behaviour-driven formal model development CPU verification

Analytical – Formal (cont.)	Deductive verification Formal requirements validation Model checking Model-based design verification Model-based formal specification and verification of robotic systems Model-based safety analysis Reachability-analysis-based verification for safety-critical hybrid systems Runtime verification based on formal specification Source code static analysis Theorem proving and SMT solving V&V of machine learning-based systems using simulators
Analytical – Semi-formal	Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking Failure detection and diagnosis (FDD) in robotic systems Human Interaction Safety Analysis (HISA) Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Model-based assurance and certification Model-based design verification Model-Based Safety Analysis Model-based threat analysis Risk analysis Source code static analysis Traceability management of safety software Vulnerability analysis of cryptographic modules against hardware-based attacks Wireless interface network security assessment

Table A.3 V&V Method classification according to the Type of Component under Evaluation dimension of the VALU3S framework

Model	Behaviour-driven formal model development Behaviour-driven model development and test-driven model review CPU verification Deductive verification Failure detection and diagnosis (FDD) in robotic systems Formal requirements validation Human Interaction Safety Analysis (HISA) Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Knowledge-centric system artefact quality analysis Knowledge-centric traceability management

Model (cont.)	Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-based safety analysis Model-based testing Model-based threat analysis Model-Implemented Attack injection Model-Implemented Fault injection Reachability-analysis-based verification for safety-critical hybrid systems Runtime verification based on formal specification Simulation-based robot verification Simulation-based testing for human-robot collaboration Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Theorem proving and SMT solving Traceability management of safety software V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Wireless interface network security assessment
Software	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking Deductive verification Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Human Interaction Safety Analysis (HISA) Interface fault injection Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Knowledge-centric traceability management Model checking Model-based assurance and certification Model-based formal specification and verification of robotic systems Model-based mutation testing

Software (cont.)	Model-based robustness testing Model-based safety analysis Model-based testing Penetration testing of industrial systems Reachability-analysis-based verification for safety-critical hybrid systems Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-based robot verification Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Vulnerability and attack injection Wireless interface network security assessment
Hardware	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review CPU verification Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Human Interaction Safety Analysis (HISA) Knowledge-centric traceability management Model-based assurance and certification Model-based mutation testing Model-based robustness testing Model-based safety analysis Model-based testing Penetration testing of industrial systems Risk analysis Risk-based testing Runtime verification based on formal specification

Hardware (cont.)	Signal analysis and probing Software-implemented fault injection Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Vulnerability analysis of cryptographic modules against hardware-based attacks Wireless interface network security assessment

Table A.4 V&V Method classification according to the Evaluation Tool dimension of the VALU3S framework

Open Source	Assessment of cybersecurity-informed safety Behaviour-driven formal model development CPU verification Deductive verification Dynamic analysis of concurrent programs Formal requirements validation Human Interaction Safety Analysis (HISA) Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based safety analysis Model-based testing Penetration testing of industrial systems Reachability-analysis-based verification for safety-critical hybrid systems Runtime verification based on formal specification Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-based robot verification Simulation-based testing for human-robot collaboration Software component testing Source code static analysis Test parallelization and automation Theorem proving and SMT solving V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V Vulnerability analysis of cryptographic modules against hardware-based attacks Wireless interface network security assessment

Proprietary	Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking CPU verification Deductive verification Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Interface fault injection Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Model checking Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-Based Safety Analysis Model-based testing model-based threat analysis Model-Implemented Attack injection Model-Implemented Fault injection Reachability-analysis-based verification for safety-critical hybrid systems Risk analysis Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-based robot verification Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Traceability management of safety software V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V Virtual architecture development and simulated evaluation of software concepts Vulnerability analysis of cryptographic modules against hardware-based attacks Vulnerability and attack injection

Table A.5 V&V Method classification according to the Evaluation Stage dimension of the VALU3S framework

Verification	Behaviour-driven formal model development
	Behaviour-driven model development and test-driven model review
	Code design and coding standard compliance checking
	CPU verification
	Deductive verification
	Dynamic analysis of concurrent programs
	Failure detection and diagnosis (FDD) in robotic systems
	Human Interaction Safety Analysis (HISA)
	Interface fault injection
	Knowledge-centric system artefact quality analysis
	Knowledge-centric traceability management
	Model checking
	Model-based assurance and certification
	Model-based design verification
	Model-based formal specification and verification of robotic systems
	Model-based mutation testing
	Model-based robustness testing
	Model-based safety analysis
	Model-based testing
	Model-Implemented Attack injection
	Model-Implemented Fault injection
	Penetration testing of industrial systems
	Reachability-analysis-based verification for safety-critical hybrid systems
	Risk analysis
	Risk-based testing
	Runtime verification based on formal specification
	Signal analysis and probing
	Simulation-based attack injection at system-level
	Simulation-based fault injection at system-level
	Simulation-based robot verification
	Simulation-based testing for human-robot collaboration
	Software-implemented fault injection
	Source code static analysis
	Test optimization for simulation-based testing of automated systems
	Test oracle observation at runtime
	Test parallelization and automation
	Theorem proving and SMT solving
	V&V of machine learning-based systems using simulators
	Virtual architecture development and simulated evaluation of software concepts
	Vulnerability and attack injection
	Wireless interface network security assessment

Validation	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Interface fault injection Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Machine learning model validation Model-based design verification Model-based safety analysis Model-based threat analysis Penetration testing of industrial systems Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test parallelization and automation Traceability management of safety software V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Vulnerability analysis of cryptographic modules against hardware-based attacks Vulnerability and attack injection Wireless interface network security assessment

Table A.6 V&V Method classification according to the Type of the Component under Evaluation dimension of the VALU3S framework

Sensing	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking Deductive verification Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Human Interaction Safety Analysis (HISA) Intrusion detection for WSN based on WPM state estimation

Sensing (cont.)	Kalman filter-based fault detector Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-based safety analysis Model-based testing Model-based threat analysis Model-Implemented Attack injection Model-Implemented Fault injection Penetration testing of industrial systems Reachability-analysis-based verification for safety-critical hybrid systems Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-based robot verification Software component testing Test optimization for simulation-based testing of automated systems Test parallelization and automation Theorem proving and SMT solving Traceability management of safety software Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Vulnerability analysis of cryptographic modules against hardware-based attacks Wireless interface network security assessment
Thinking	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking CPU verification Deductive verification Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs

Thinking (cont.)	Formal requirements validation Human Interaction Safety Analysis (HISA) Interface fault injection Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Knowledge-centric traceability management Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-Based Safety Analysis Model-based testing Model-based threat analysis Model-Implemented Attack injection Model-Implemented Fault injection Penetration testing of industrial systems Reachability-analysis-based verification for safety-critical hybrid systems Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-Based Robot Verification Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Traceability management of safety software V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts

Thinking (cont.)	Vulnerability analysis of cryptographic modules against hardware-based attacks Vulnerability and attack injection Wireless interface network security assessment
Acting	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking Deductive verification Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Human Interaction Safety Analysis (HISA) Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-Based Safety Analysis Model-based testing Model-based threat analysis Model-Implemented Attack injection Model-Implemented Fault injection Penetration testing of industrial systems Reachability-analysis-based verification for safety-critical hybrid systems Risk analysis Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-based robot verification Software component testing Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Traceability management of safety software

Acting (cont.)	Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Vulnerability analysis of cryptographic modules against hardware-based attacks Wireless interface network security assessment

Table A.7 V&V Method classification according to the Type of Requirement under Evaluation dimension of the VALU3S framework

Functional	Behaviour-driven formal model development Behaviour-driven model development and test-driven model review CPU verification Deductive verification Dynamic analysis of concurrent programs Formal requirements validation Knowledge-centric traceability management Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-Based Safety Analysis Model-based testing Reachability-analysis-based verification for safety-critical hybrid systems Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-Based Robot Verification Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Virtual architecture development and simulated evaluation of software concepts
Non-functional – Safety	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Code design and coding standard compliance checking CPU verification Deductive verification

Non-functional – Safety (cont.)	Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Human Interaction Safety Analysis (HISA) Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-Based Safety Analysis Model-Implemented Fault injection Reachability-analysis-based verification for safety-critical hybrid systems Risk analysis Runtime verification based on formal specification Simulation-based fault injection at system-level Simulation-Based Robot Verification Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Theorem proving and SMT solving Traceability management of safety software V&V of machine learning-based systems using simulators Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts
Non-functional – Cybersecurity	Assessment of cybersecurity-informed safety CPU verification Failure detection and diagnosis (FDD) in robotic systems Intrusion detection for WSN based on WPM state estimation Kalman filter-based fault detector Knowledge-centric traceability management Model-based assurance and certification Model-Based Safety Analysis Model-based threat analysis Model-Implemented Attack injection Penetration testing of industrial systems

Non-functional – Cybersecurity (cont.)	Risk analysis Simulation-based attack injection at system-level Source code static analysis V&V of machine learning-based systems using simulators Vulnerability analysis of cryptographic modules against hardware-based attacks Vulnerability and attack injection Wireless interface network security assessment
Non-functional – Privacy	Knowledge-centric traceability management Model-based assurance and certification Source code static analysis
Non-functional – Others	CPU verification Failure detection and diagnosis (FDD) in robotic systems Interface fault injection Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Model-based assurance and certification Software-implemented fault injection Source code static analysis Virtual & augmented reality-based user interaction V&V and technology acceptance

Table A.8 V&V Method classification according to the Evaluation Performance Indicator dimension of the VALU3S framework

V&V process criteria	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven model development and test-driven model review Code design and coding standard compliance checking CPU verification Deductive verification Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Interface fault injection Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based mutation testing Model-based robustness testing Model-based safety analysis

V&V process criteria (cont.)	Model-based testing Model-Implemented Attack injection Model-Implemented Fault injection Penetration testing of industrial systems Reachability-analysis-based verification for safety-critical hybrid systems Risk-based testing Runtime verification based on formal specification Signal analysis and probing Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-Based Robot Verification Simulation-based testing for human-robot collaboration Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test parallelization and automation Theorem proving and SMT solving Virtual architecture development and simulated evaluation of software concepts Vulnerability and attack injection Wireless interface network security assessment
SCP criteria	Assessment of cybersecurity-informed safety Behaviour-driven formal model development Behaviour-driven formal model development CPU verification Deductive verification Dynamic analysis of concurrent programs Failure detection and diagnosis (FDD) in robotic systems Fault injection in FPGAs Formal requirements validation Human Interaction Safety Analysis (HISA) Interface fault injection Intrusion detection for WSN based on WPM State Estimation Kalman filter-based fault detector Knowledge-centric system artefact quality analysis Knowledge-centric traceability management Machine learning model validation Model checking Model-based assurance and certification Model-based design verification Model-based formal specification and verification of robotic systems Model-based safety analysis Model-based threat analysis

SCP criteria (cont.)	Model-Implemented Attack injection Model-Implemented Fault injection Penetration testing of industrial systems Risk analysis Runtime verification based on formal specification Simulation-based attack injection at system-level Simulation-based fault injection at system-level Simulation-based robot verification Simulation-based testing for human-robot collaboration Software component testing Software-implemented fault injection Source code static analysis Test optimization for simulation-based testing of automated systems Test oracle observation at runtime Test parallelization and automation Theorem proving and SMT solving Traceability management of safety software Virtual & augmented reality-based user interaction V&V and technology acceptance Virtual architecture development and simulated evaluation of software concepts Vulnerability analysis of cryptographic modules against hardware-based attacks Vulnerability and attack injection Wireless interface network security assessment

www.valu3s.eu

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.