Automated Software Testing in Cyber-Physical Systems

Lionel Briand

NUS, Singapore, 2019
Dependable Breakfast
SnT Center: Mandate

Diagram:
- Partner
- Domain knowledge
- Innovation demand
- Fundamental research
- Applied research
- Collaborating entities
- SnT
SnT Center: Overview

- 40 industry partners and 4 spin-off companies
- 287 employees
- 51 nationalities
- Over 90 paper and individual awards
- Acquired competitive funding since launch (2009)
- 101 M€
SVV Dept.

- Established in 2012
- Requirements Engineering, Security Analysis, Design Verification, Automated Testing, Runtime Monitoring
- ~ 30 lab members
- Partnerships with industry
- ERC Advanced grant
Collaborative Research @ SnT

- Research in context
- Addresses actual needs
- Well-defined problem
- Long-term collaborations
- Our lab is the industry
Talk Objectives

• Applications of main AI techniques to test automation

• Focus on the specifics of CP systems

• Overview (partial) and lessons learned, with pointers for further information

• Industrial research projects, collaborative model, lessons learned

• Disclaimer: Inevitably biased presentation based on personal experience. This is not a survey.
Introduction
Definition of Software Testing

Software testing involves the execution of a software component or system to evaluate one or more properties of interest such as meeting the requirements that guided its design and development, responding correctly to all kinds of inputs, and performing its functions within acceptable resources.

Adapted from Wikipedia
Software Testing Overview

1. **SW Representation** (e.g., specifications)
2. **Executable**
3. **Expected Results or Properties**
4. **Derive Test cases**
5. **Execute Test cases**
6. **Get Test Results (state, output)**
7. **Compare**
   - [Test Result\!\!\!=\!Oracle]
   - [Test Result\!\!=\!Oracle]

**Test Oracle**
Main Challenge

• The main challenge in testing software systems is **scalability**

• **Scalability:** The extent to which a technique can be applied on large or complex artifacts (e.g., input spaces, code, models) and still provide **useful support within acceptable resources**.

• Effective automation is a **prerequisite for scalability**
Importance of Software Testing

• Software testing is by far the **most prevalent** verification and validation technique in practice

• It represents a **large percentage** of software development costs, e.g., >50% is not rare

• Testing services are a **USD 9-Billion market**

• The cost of software failures was estimated to be (a very minimum of) **USD 1.1 trillion** in 2016

• **Inadequate tools and technologies** is one of the most important factors of testing costs and inefficiencies

Cyber-Physical Systems

- A system of collaborating computational elements controlling physical entities
CPS Development Process

Model-in-the-Loop Stage

Functional modeling:
- Controllers
- Plant
- Decision

Continuous and discrete Simulink models

Model simulation and testing

Software-in-the-Loop Stage

Architecture modelling
- Structure
- Behavior
- Traceability

System engineering modeling (SysML)

Analysis:
- Model execution and testing
- Model-based testing
- Traceability and change impact analysis
- ...

(partial) Code generation

Hardware-in-the-Loop Stage

Deployed executables on target platform

Hardware (Sensors ...)
Analog simulators

Testing (expensive)
Testing Cyber-Physical Systems

- **MiL and SiL testing**: Computationally expensive (simulation of physical models)

- **HiL**: Human effort involved in setting up the hardware and analog simulators

- **Number of test executions tends to be limited** compared to other types of systems

- Test input space is often **extremely large**, i.e., determined by the complexity of the physical environment

- **Traceability** between system testing and requirements is mandated by standards
Artificial Intelligence

- Meta-heuristic search
- Machine learning
- Natural Language Processing
Metaheuristic Search

- **Stochastic optimization**
- **Evolutionary computing, e.g., genetic algorithms**
- Efficiently explore the search space in order to find good (near-optimal) feasible solutions
- Address both discrete- and continuous-domain optimization problems
- **Black-box optimization**
- Applicable to many practical situations, including SW testing
- Provide no guarantee of optimality
Search-Based Software Testing

- Express test generation problem as a search or optimization problem
- Search for test input data with certain properties, i.e., source code coverage
- Non-linearity of software (if, loops, ...): complex, discontinuous, non-linear search spaces

“Search-Based Software Testing: Past, Present and Future”
Phil McMinn
Genetic Algorithms (GAs)

**Genetic Algorithm**: Population-based, search algorithm inspired by evolution theory

**Natural selection**: Individuals that best fit the natural environment survive

**Reproduction**: Surviving individuals generate offsprings (next generation)

**Mutation**: Offsprings inherit properties of their parents with some mutations

**Iteration**: Generation after generation the new offspring fit better the environment than their parents
Machine Learning and Testing

- ML supports decision making and estimation based on data
- Test planning
  - Test cost estimation
- Test case management
  - Test case prioritization
  - Test case design
  - Test case refinement
  - Test case evaluation

- Debugging
  - Fault localization
  - Bug prioritization
  - Fault prediction

NLP and Testing

• Natural language is **prevailent in software development**

• User documentation, procedures, natural language requirements, etc.

• **Natural Language Processing (NLP)**

• Can it be used to **help automate testing**?

  • Help derive test cases, including oracles, from textual requirements or specifications

  • Establish traceability between requirements and system test cases (required by many standards)
Research Projects in Collaboration with Industry
Testing Advanced Driving Assistance Systems (SiL)

[Ben Abdessalem et al.]
Advanced Driver Assistance Systems (ADAS)

Automated Emergency Braking (AEB)

Pedestrian Protection (PP)

Lane Departure Warning (LDW)

Traffic Sign Recognition (TSR)
Advanced Driver Assistance Systems (ADAS)

Decisions are made over time based on sensor data
Automotive Environment

- **Highly varied environments**, e.g., road topology, weather, building and pedestrians …

- **Huge number of possible scenarios**, e.g., determined by trajectories of pedestrians and cars

- ADAS play an increasingly **critical role** in modern vehicles

- Systems must comply with **functional safety standards**, e.g., ISO 26262

- **A challenge for testing**
A General and Fundamental Shift

- Increasingly so, it is easier to learn behavior from data using machine learning, rather than specify and code
- Some ADAS components may rely on deep learning …
- Millions of weights learned (Deep Neural Networks)
- No explicit code, no specifications
- Verification, testing?
- State of the art includes adequacy coverage criteria and mutation testing for DNNs
Our Goal

• Developing an automated testing technique for ADAS

• To help engineers efficiently and effectively explore the complex test input space of ADAS

• To identify critical (failure-revealing) test scenarios

• Characterization of input conditions that lead to most critical situations, e.g., safety violations
Automated Emergency Braking System (AEB)

Decision making

Objects’ position/speed

Vision (Camera)

Sensor

“Brake-request” when braking is needed to avoid collisions

Brake Controller
Example Critical Situation

• “AEB properly detects a pedestrian in front of the car with a high degree of certainty and applies braking, but an accident still happens where the car hits the pedestrian with a relatively high speed”
Testing ADAS

On-road testing

- Time-consuming
- Expensive
- Unsafe

Simulation-based (model) testing

A simulator based on physical/mathematical models
Testing via Physics-based Simulation

- Physical plant (vehicle / sensors / actuators)
- Other cars
- Pedestrians
- Environment (weather / roads / traffic signs)

Simulator (Matlab/Simulink)

ADAS (SUT)

Model (Matlab/Simulink)

Test input

Test output

time-stamped output
AEB Domain Model

- VisibleRange
  - fog: Boolean
  - fogColor: FogColor

- Weather
  - visibility: VisibilityRange
  - fog: Boolean
  - fogColor: FogColor

- WeatherC
  - {OCL} self.fog=false implies self.visibility = "300" and self.fogColor=None

- Snow
  - snowType: SnowType

- Rain
  - rainType: RainType

- CurvedRadius (CR)
  - 5 - 10 - 15 - 20
  - 25 - 30 - 35 - 40

- RampHeight (RH)
  - 4 - 6 - 8 - 10 - 12

- SnowType
  - ModerateSnow
  - HeavySnow
  - VeryHeavySnow
  - ExtremeSnow

- RainType
  - ModerateRain
  - HeavyRain
  - VeryHeavyRain
  - ExtremeRain

- FogColor
  - DimGray
  - Gray
  - DarkGray
  - Silver
  - LightGray
  - None

- Test Scenario
  - simulationTime: Real
  - timeStep: Real

- Road
  - frictionCoeff: Real

- Vehicle
  - v_0: Real

- Pedestrian
  - x_0^p: Real
  - y_0^p: Real
  - v_0^p: Real
  - \theta_0^p: Real

- Static input
- Dynamic input
- Output

- AEB Output
  - \( v_1 \): TTC: Real
  - \( v_2 \): certaintyOfDetection: Real
  - \( v_3 \): braking: Boolean

- Position
  - x: Real
  - y: Real

- Position vector

- Mobile object

- Output functions
  - \( F_1 \): Real
  - \( F_2 \): Real
ADAS Testing Challenges

• Test input space is multidimensional, large, and complex

• Explaining failures and fault localization are difficult

• Execution of physics-based simulation models is computationally expensive
Our Approach

• We use decision tree classification models

• We use multi-objective search algorithm (NSGAII)

• **Objective Functions:**
  1. Minimum distance between the pedestrian and the field of view
  2. The car speed at the time of collision
  3. The probability that the object detected is a pedestrian

• Each search iteration calls simulation to compute objective functions
Multiple Objectives: Pareto Front

A multi-objective optimization algorithm (e.g., NSGA II) must:

- Guide the search towards the global Pareto-Optimal front.
- Maintain solution diversity in the Pareto-Optimal front.

Individual A Pareto dominates individual B if A is at least as good as B in every objective and better than B in at least one objective.
Search-based Testing Process

Input data ranges/dependencies + Simulator + Fitness functions

Test input generation (NSGA II)
- Select best tests
- Generate new tests

Evaluating test inputs
- Simulate every (candidate) test
- Compute fitness functions

(candidate) test inputs
Fitness values

Test cases revealing worst case system behaviors
Search: Genetic Evolution

- Initial input
- Fitness computation
- Selection
- Breeding
Better Guidance

- Fitness computations rely on simulations and are very expensive
- Search needs better guidance
Decision Trees

Partition the input space into homogeneous regions
Genetic Evolution Guided by Classification

Initial input
Fitness computation
Classification
Selection
Breeding
Search Guided by Classification

Input data ranges/dependencies + Simulator + Fitness functions

Test input generation (NSGA II)
- Build a classification tree
- Select/generate tests in the fittest regions
- Apply genetic operators

Evaluating test inputs
- Simulate every (candidate) test
- Compute fitness functions

(candidate) test inputs
Fitness values

Test cases revealing worst case system behaviors + A characterization of critical input regions
NSGAII-DT vs. NSGAII

NSGAII-DT outperforms NSGAII
The generated critical regions consistently become smaller, more homogeneous and more precise over successive tree generations of NSGAII-DT.
Usefulness

• The characterizations of the different critical regions can help with:

  (1) **Debugging** the system model

  (2) **Identifying possible hardware changes** to increase ADAS safety

  (3) **Providing proper warnings** to drivers
System Integration

System Under Test (SUT)

- feature 1
- feature 2
- feature n

Integration component

Input:
- sensors
- cameras

Output:
- actuators
Generation of System Test Cases from Requirements (HiL)

[Wang et al.]
Context

Automotive Embedded Systems

- Small but safety critical systems
- Traceability from requirements to system test cases (ISO 26262)
- Requirements act as a contract
- Many requirements changes, leading to negotiations
Problem

Automatically verify the compliance of software systems (with HiL) with their functional requirements in a cost-effective way.
Working Assumption

Use Case Specifications

Domain Model
Use Case Specifications (RUCM template)

Domain Model

Concise Mapping Table

Automated Generation

Executable Test Cases
Use Case Specifications (RUCM template)

Domain Model

Concise Mapping Table

Automated Generation

NL processing

Executable Test Cases
Use Case Specifications

Example

BodySense: embedded system that determines the occupancy status of seats in a car
Use Case Specifications
Example

--written according to RUCM (Yue’13) template--

**Precondition:** The system has been initialized

**Basic Flow**

1. The SeatSensor **SENDS** the weight **TO** the system.
2. **INCLUDE USE CASE** Self Diagnosis.
3. The system **VALIDATES THAT** no error has been detected.
4. The system **VALIDATES THAT** the weight is above 20 Kg.
5. The system sets the occupancy status to adult.
6. The system **SENDS** the occupancy status **TO** AirbagControlUnit.
Precondition: The system has been initialized

Basic Flow
1. The SeatSensor SENTS the weight TO the system.
2. INCLUDE USE CASE Self Diagnosis.
3. The system VALIDATES THAT no error has been detected.
4. The system VALIDATES THAT the weight is above 20 Kg.
5. The system sets the occupancy status to adult.
6. The system SENTS the occupancy status TO AirbagControlUnit.

Alternative Flow
RFS 4.
1. IF the weight is above 1 Kg THEN
2. The system sets the occupancy status to child.
3. ENDIF.
4. RESUME STEP 6.
Precondition: The system has been initialized.

The SeatSensor **Sends** the weight **To** the system.

**INCLUDE USE CASE** Self Diagnosis.

The system **Validates That** no error has been detected.

The system sets the occupancy status to adult.

The system **Sends** the occupant class **To** AirbagControlUnit.

**IF** the weight is above 1 Kg **THEN**

The system sets the occupancy status to child.

The system **Sends** the occupant class **To** AirbagControlUnit.

**Model-based Test Case Generation driven by coverage criteria**
Formalizing Conditions

**Domain Model:**

<table>
<thead>
<tr>
<th>BodySense</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialized: Bool</td>
</tr>
<tr>
<td>occupancyStatus: Occupancy</td>
</tr>
<tr>
<td>weight: int</td>
</tr>
</tbody>
</table>

```
Error.allInstances() -> forAll( i | i.isDetected = false)
```

**OCL constraint:**

“The system VALIDATES THAT no error has been detected.”
Precondition: The system has been initialized.

The SeatSensor SENDS the weight TO the system.

Path condition:

System.allInstances() ->forall( s | s.initialized = true )
AND System.allInstances() ->forall( s | s.initialized = true )
AND Error.allInstances() ->forall( e | e.isDetected = false )
AND System.allInstances() ->forall( s | s.occupancyStatus = Occupancy::Adult )

Test inputs:

system : BodySense
initialized = true
occupancyStatus = Adult
weight = 40

errors

errors

te : TemperatureError
isDetected = false

ve : VoltageError
isDetected = false
Automated Generation of OCL Expressions

“The system VALIDATES THAT no error has been detected.”

```
Error.allInstances()->forAll( i | i.isDetected = false)
```

**BodySense**
- initialized: Bool
- occupancyStatus: Occupancy
- weight: int

**Occupancy**
- Adult
- Child
- Empty

**Error**
- isDetected: Bool

**OCLgen**
Error.allInstances().forall(i | i.isDetected = false)

Entity Name

left-hand side (variable)

operator

right-hand side (variable/value)
OCLgen Solution

1. determine the role of words in a sentence

actor affected by the verb

final state

“The system sets the occupancy status to adult.”
OCLgen solution

1. determine the role of words in a sentence

"The system sets the occupancy status to adult."

actor affected by the verb
final state

2. match words in the sentence with concepts in the domain model

<table>
<thead>
<tr>
<th>«enumeration» Occupancy</th>
</tr>
</thead>
</table>
| Adult
| Child
| Empty

<table>
<thead>
<tr>
<th>BodySense</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialized: Bool</td>
</tr>
<tr>
<td>occupancyStatus: Occupancy</td>
</tr>
<tr>
<td>weight: int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDetected: Bool</td>
</tr>
</tbody>
</table>

errors
The system sets the occupancy status to adult.

BodySense.allInstances() -> forall (i | i.occupancyStatus = Occupancy::Adult)
OCLgen solution

1. determine the role of words in a sentence

   Based on Semantic Role Labeling

   “The system sets the occupancy status to adult.”

2. match words in the sentence with concepts in the domain model

   Based on String similarity

3. generate the OCL constraint using a verb-specific transformation rule

   Lexicons that describe the sets of roles typically occurring with a verb
Case Study Results

- **88 OCL constraints** had to be generated from 88 use case specification sentences
- 69 constraints generated
  - 66 correct, only 3 incorrect
- Very high precision: 0.97
- High Recall: 0.75
- Problems due to use of **inconsistent terminology and imprecise requirements**
- Can be detected beforehand using NLP
Schedulability Analysis and Stress Testing (SiL and HiL)

[Di Alesio et al.]
System monitors gas leaks and fire in oil extraction platforms
Problem Definition

- **Schedulability analysis** encompasses techniques that try to determine whether (critical) tasks are schedulable, i.e., meet their deadlines.

- **Stress testing** runs carefully selected test cases that have a high probability of leading to deadline misses.

- **Stress testing** is complementary to schedulability analysis.

- Testing is typically expensive, e.g., hardware in the loop.

- Finding stress test cases is difficult.
Finding Stress Test Cases is Hard

\( j_0, j_1, j_2 \) arrive at \( at_0, at_1, at_2 \) and must finish before \( dl_0, dl_1, dl_2 \)

\( J_1 \) can miss its deadline \( dl_1 \) depending on when \( at_2 \) occurs!
Challenges and Solutions

• Ranges for arrival times form a very large input space

• Task interdependencies and properties constrain what parts of the space are feasible

• **Solution:** We re-expressed the problem as a constraint optimization problem and used a combination of constraint programming (CP, IBM CPLEX) and meta-heuristic search (GA)

• **GA** is scalable and CP offers guarantees
Solution Overview

UML Modeling (e.g., MARTE)

System Design

System Platform

Design Model (Time and Concurrency Information)

Constraint Optimization

Deadline Misses Analysis

Optimization Problem
(Find arrival times that maximize the chance of deadline misses)

Constraint Prog. and Genetic Algorithms

Solutions
(Task arrival times likely to lead to deadline misses)

Stress Test Cases

INPUT

OUTPUT
Combining CP and GA

![Search Space Diagram]
Summary

• We provided a solution for generating stress test cases by combining meta-heuristic search and constraint programming

  • Meta-heuristic search (GA) identifies high risk regions in the input space

  • Constraint programming (CP) finds provably worst-case schedules within these (limited) regions

• Achieve (nearly) GA efficiency and CP effectiveness

• Our approach can be used both for stress testing and schedulability analysis (assumption free)
Other Industrial Projects

• **Delphi & QRA (Automotive, Aerospace):** MiL Testing and verification of CPS Simulink models (e.g., controllers) [Matinnejad et al.]

• **SES (Satellite):** Hardware-in-the-Loop, acceptance testing of CPS [Shin et al.]

• **IEE:** Testing timing properties in embedded systems [Wang et al.]

• **Luxembourg government:** Generating representative, synthetic test data for information systems [Soltana et al.]
Controller MiL Testing

Search-based Test Input Generation

Model Simulation

Fitness Evaluation

Requirements
Create Input signals $\Rightarrow$ Simulation $\Rightarrow$ Display Output

**Input:**
- $ia = [-1 -1 -1 -6 -6 -6 -10 -10 -10 -10]$
- $ib = [2 2 2 5 5 5 5 -7 -7 -7]$
- $ic = [-10 -10 -10 -5 -5 -5 -8 -8 -8 -8]$
- $PClimit = [1 1 1 1 1 1 1 1 1 1 1]$
- $Tlevel = [0 0 0 0 0 0 0 0 0 0 0]$

**Output:**
- $FC = [0 0 0 0 0 0 0 0 0 0 0]$
- $Selval = [-1.6 -1.6 -1.6 -5 -5 -5 -5 -8.4 -8.4 -8.4 -8.4]$

Model Inputs and Outputs
**R05:** If the tank 1 liquid height is greater than or equal to the sensor height of the tank 1 HIGH liquid sensor then the sensor should return an active (TRUE) state to the system.

<table>
<thead>
<tr>
<th>Req. Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀t 0 ≤ t ≤ T, T1Height[t] ≥ T1HSensorHeight → T1HSensorState[t] = 1.0</td>
</tr>
</tbody>
</table>

**Fitness Function**

\[
\max_{0 \leq t \leq T} \left( (T1Height[t] - T1HSensorHeight) \times (1 - T1HSensorState[t]) \right)
\]
In-Orbit Acceptance Testing

- Satellite on-board system
- Validation
- In-Orbit acceptance system testing
- Overhead of manipulating devices
- Risk of hardware damage
- Uncertainties in execution time
- Tight time budget and environmental constraints

Environment: space
Launch effect e.g., vibration
Approach Overview

Modeling
- Formalism
  - Components
  - Test cases
  - Test suites

Minimizing
- Heuristic algorithm
  - Initializations
  - Teardowns
  - Model checking

Prioritizing
- Search
  - Multi-objective
  - Simulations
Reflections
Role of AI

• **Metaheuristic search:**

  Most test automation problems can be re-expressed into search (stochastic optimization) problems

• **Machine learning:**

  Automation can be better guided and effective when learning from data: test execution results, fault detection …

• **Natural Language Processing:**

  Natural language is commonly used and is an obstacle to automated analysis and therefore test automation
Search-Based Solutions

• **Versatile**

• Helps relax assumptions compared to exact approaches

• Helps decrease modeling requirements

• Scalability, e.g., easy to parallelize

• Requires massive empirical studies for validation

• **Search is rarely sufficient by itself**
Multidisciplinary Approach

- **Single-technology approaches rarely work in practice**

- Combined search with:
  - Machine learning
  - Solvers, e.g., CP, SMT
  - Statistical approaches, e.g., sensitivity analysis
  - System and environment modeling and simulation
The Essential Role of Models

• No effective and scalable test automation is possible, in many contexts, without models: Guiding test generation, generating oracles

  • Requirements (e.g., use case specifications)
  • Architecture (e.g., task properties and dependencies)
  • Behavior of system and environment (e.g., state and timing properties)
  • ...


The Road Ahead

- We need **strike a balance** in terms of scalability, practicality, applicability, and offering a maximum level of dependability guarantees.

- We need **more multi-disciplinary research** involving AI.

- In most industrial contexts, offering absolute guarantees (correctness, safety, or security) is **illusory**.

- The best **trade-offs** between cost and level of guarantees is necessarily context-dependent.

- Research in this field cannot be oblivious to **context** (domain …)
Industrial Problems

- Many academic papers address problems that are unlikely to exist as defined, e.g., working assumptions.

- On the other hand, many industrial problems are insufficiently addressed by research.

- Context factors and working assumptions have a huge impact on software engineering solutions.

- Scalability and practicality aspects are largely ignored by research – they are often considered as an afterthought.

- Software engineering solutions are often trade-offs in practice between scalability, cost, accuracy …
The Case for Context-Driven Software Engineering Research

Generalizability Is Overrated

Lionel Briand, Domenico Bianculli, Shiva Najati, Fabrizio Pastora, and Mehrdad Taherzadeh

This article follows up on Lionel Briand's 2012 Sounding Board article on the significant disconnect between research and industrial needs in software engineering. Here, we argue that software engineering research to increase its impact and serve our community toward a more successful future, it must change. Specifically, we see the need to foster context-driven research.

By that, we mean research focused on problems defined in collaboration with industrial partners and driven by concrete needs in specific domains and development projects. By analyzing publications from the top software engineering research venues, one could easily conclude that only a small portion of the papers stem from such research.

Context-driven research doesn't try to frame a general problem and devise universal solutions. Rather, it makes clear wording assumptions, given a precise context, and relies on trade-offs that make sense in that context to achieve practicality and scalability. This research paradigm applies to any topic in software engineering and isn't meant to highlight any particular research area. Because context-driven research doesn't produce results that generalize easily to any arbitrary software development environment, the following questions arise: Does it have value? How so? Is it (engineering) science? This is what we discuss in the rest of this article.

The Importance of Context

The main motivation for the proposed paradigm shift is that software engineering solutions' applicability and scalability depend largely on contextual factors, whether human (such as engineers' background), organizational (such as cost and time constraints), or domain-related (such as the level of criticality and compliance with standards). For example, whether a verification technique's cost is justified will depend on the criticality of the software being assessed and the standards it must comply with. For many safety-critical systems, standards require traceability between requirements and system test cases. This naturally leads to the development of techniques supporting the definition of requirements enabling the automated or semi-automated derivation of test scenarios.

Testing and verification techniques typically take a specific set of inputs—such as...
Collaborative Research

- **Academic research needs industry:**
  - To define proper problems
  - To account for / learn from engineering best practice
  - To perform proper evaluations

- **Industry benefits in several ways:**
  - Mitigate the risks of innovation
  - Keep up-to-date with latest ideas and results
  - Train highly qualified engineers and scientists

**Challenges:** Commitment, difference in time horizons, IP

---

**FOCUS: SOFTWARE ENGINEERING’S 50TH ANNIVERSARY**

**Software Engineering Research and Industry**

**A Symbiotic Relationship to Foster Impact**

Victor Basili, University of Maryland, College Park
Lionel Briand, Domenico Dianculli, Steve Hovius, Fabrizio Pastore, and Mahdiad Sabatera, University of Luxembourg

// This article assesses the challenges that software engineering research faces in achieving its potential. It also proposes a way for the field to move forward and become more impactful through collaborative research and innovation between public research and industry. //

---
References
Selected References (SBST)

- Matinnejad et al., “MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models”, ASE 2014

- Di Alesio et al. “Combining genetic algorithms and constraint programming to support stress testing of task deadlines”, ACM Transactions on Software Engineering and Methodology, 2015

- Soltana et al., “Synthetic Data Generation for Statistical Testing”, ASE 2017

- Shin et al., “Test case prioritization for acceptance testing of cyber-physical systems”, ISSTA 2018


Selected References (ML+SBST)

• Briand et al., “Using machine learning to refine category-partition test specifications and test suites”, Information and Software Technology (Elsevier), 2009


• Ben Abdessalem et al., "Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms”, ICSE 2018

• Ben Abdessalem et al., "Testing Autonomous Cars for Feature Interaction Failures using Many-Objective Search”, ASE 2018
Selected References (NLP)

• Wang et al., “Automatic generation of system test cases from use case specifications”, ISSTA 2015

• Wang et al., “System Testing of Timing Requirements Based on Use Cases and Timed Automata”, ICST 2017

• Wang et al., “Automated Generation of Constraints from Use Case Specifications to Support System Testing”, ICST 2018

• Mai et al., “A Natural Language Programming Approach for Requirements-based Security Testing”, ISSRE 2018
Acknowledgments

• Shiva Nejati
• Raja Ben Abdessalem
• Fabrizio Pastore
• Chunhui Wang
• Mike Sabetzadeh
• Seung Yeob Shin
• Ghanem Soltana
• Stefano Di Alesio
Automated Software Testing in Cyber-Physical Systems

Lionel Briand

NUS, Singapore, 2019