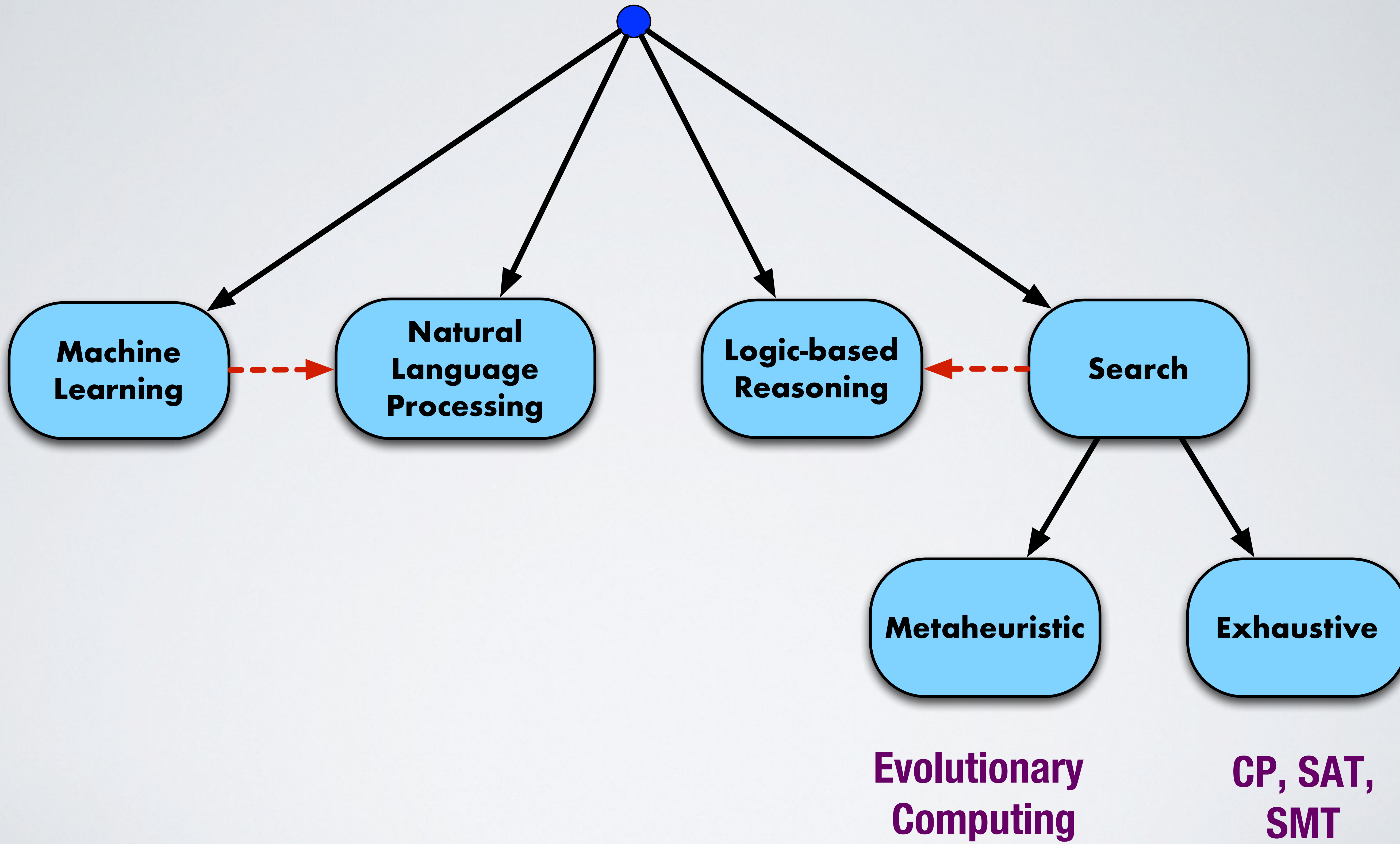
AI and Software Engineering: Past, Present, and Future

Lionel Briand

http://www.lbriand.info

CSER 2021

AI in SE


Machine Learning
Natural Language Processing
Logic-based Reasoning
Search
Metaheuristic
Exhaustive
Evolutionary Computing
CP, SAT, SMT

1991

Predicting faults in flight dynamics software

Colleagues and friends:
“Machine learning?
Why would you want to apply this?
This is not serious.”

Objectives

- Report on many years of experience about leveraging AI on industrial research SE projects.
- Personal experience, not a survey.
- Partial presentation (very much so).
- Focus on real problems, real solutions, in real contexts.
- Example projects and lessons learned.

< ~2000

Making software development predictable

Context

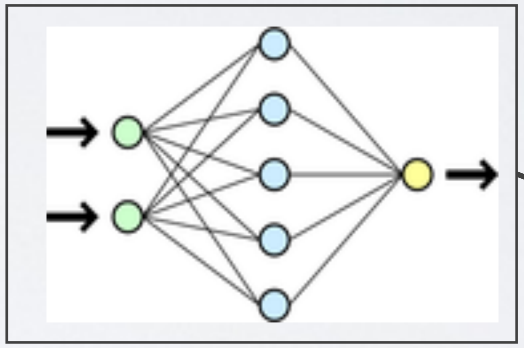
- Software development data repositories were few.
- Data available to researchers was scarce and hard to use.
- Research focused on resource and defect prediction.
- Hundreds of research papers.

Evolving Telecom Systems

For each type of Change Request (CR) involving a class:
- Number of CRs
- Lines of code added and deleted in this class
- Number of CPs involving this class
- Total number of files changed in CRs
- Total number of tests failed in CRs
- Total number of developers involved in CRs
- Total number of past CRs of the developers

Class complexity
- Class size
- Coupling
- Cohesion
- …

Class change and fault history:
For each type of CR involving this class
And for the past three releases:
- number of CRs (n-1, n-2, n-3)



Machine Learning

Faulty 1.0

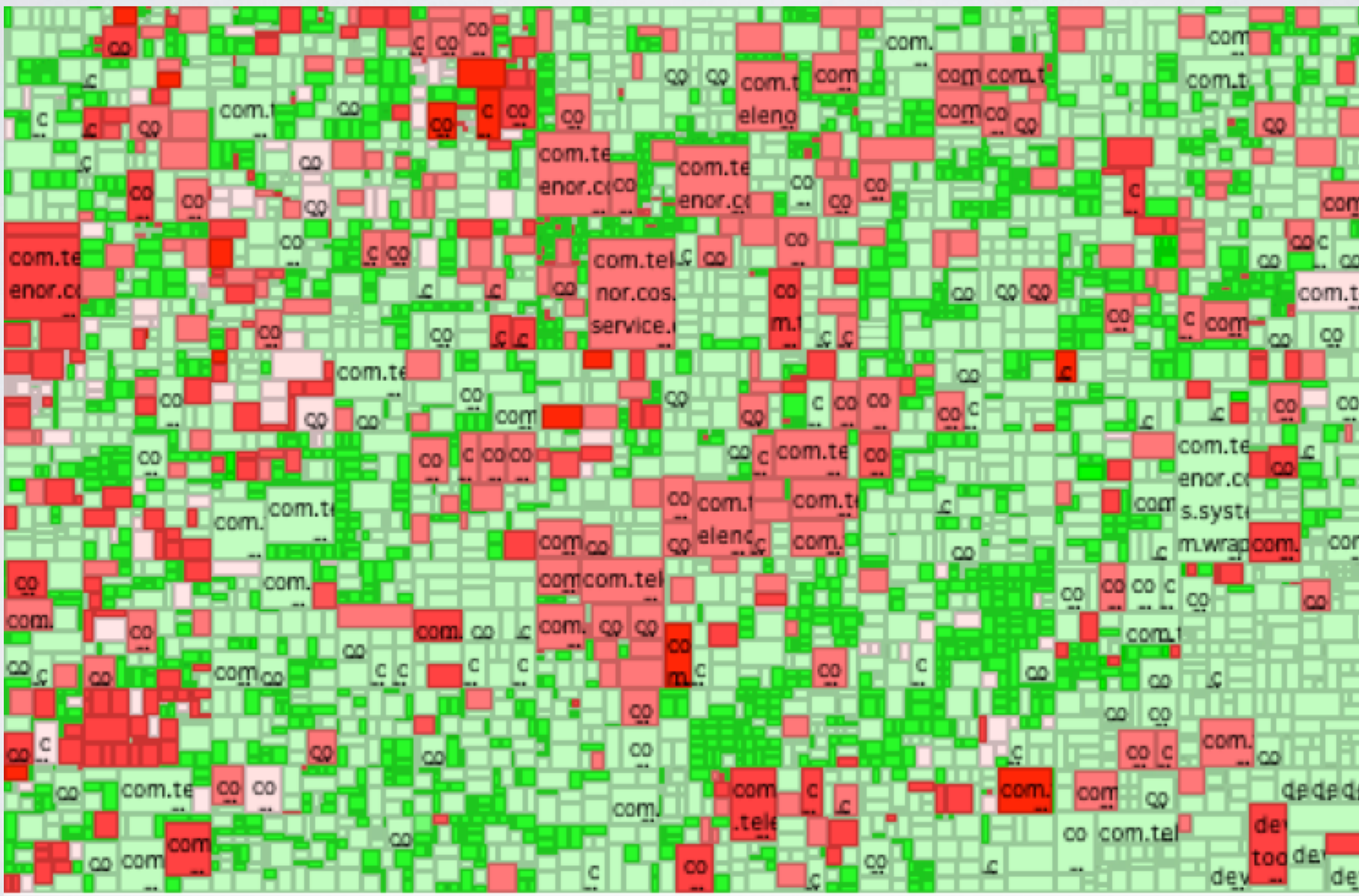
Not Faulty 0.0

Erik Arisholm

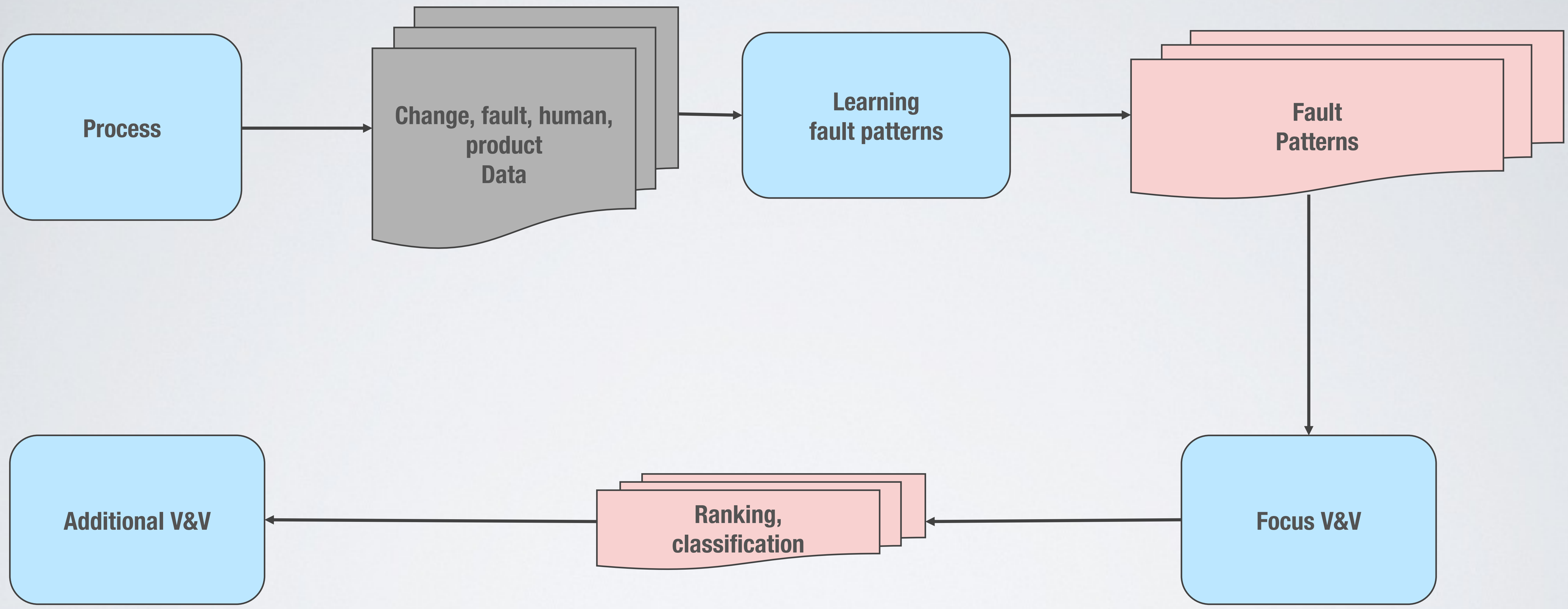
Eivind Johannessen

Arisholm et al. Journal of Systems and Software, 2010

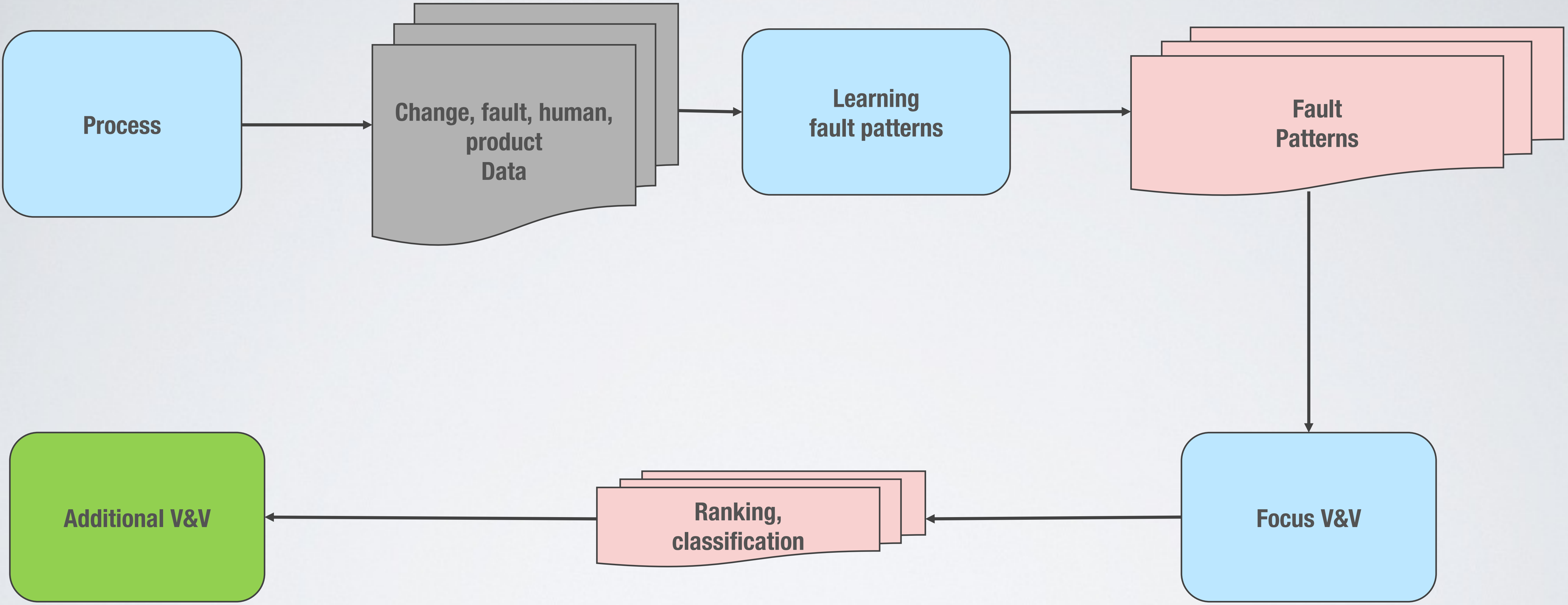
Tree Maps – Class Level



Cost-Effectiveness



Actionable?



Mapping predictions to V&V practices is not easy

ML for Prediction: Benefits

- Some machine learning techniques, such as random forests, tend to be more accurate than classical statistical techniques, e.g., based on regression.
- More flexible and robust (less assumptions), less prone to overfitting, etc.
- As larger amounts of data became increasingly available, their application became more widespread.
- Mining Software Repositories
- Many more applications: Test selection and prioritization, flaky tests, requirements identification and compliance, etc.

ML for Prediction: Challenges

- Building and maintaining a corporate prediction system.
- We are predicting **moving targets** as development practices and systems evolve quickly.
- Lots of papers on how to build prediction models.
- Very few papers on how to **effectively use such prediction models**, their benefits, etc.
- **How to use them in a cost-effective way is far from obvious.**

> ~2000

The rise of Search-Based SE (SBSE)

Why SBSE?

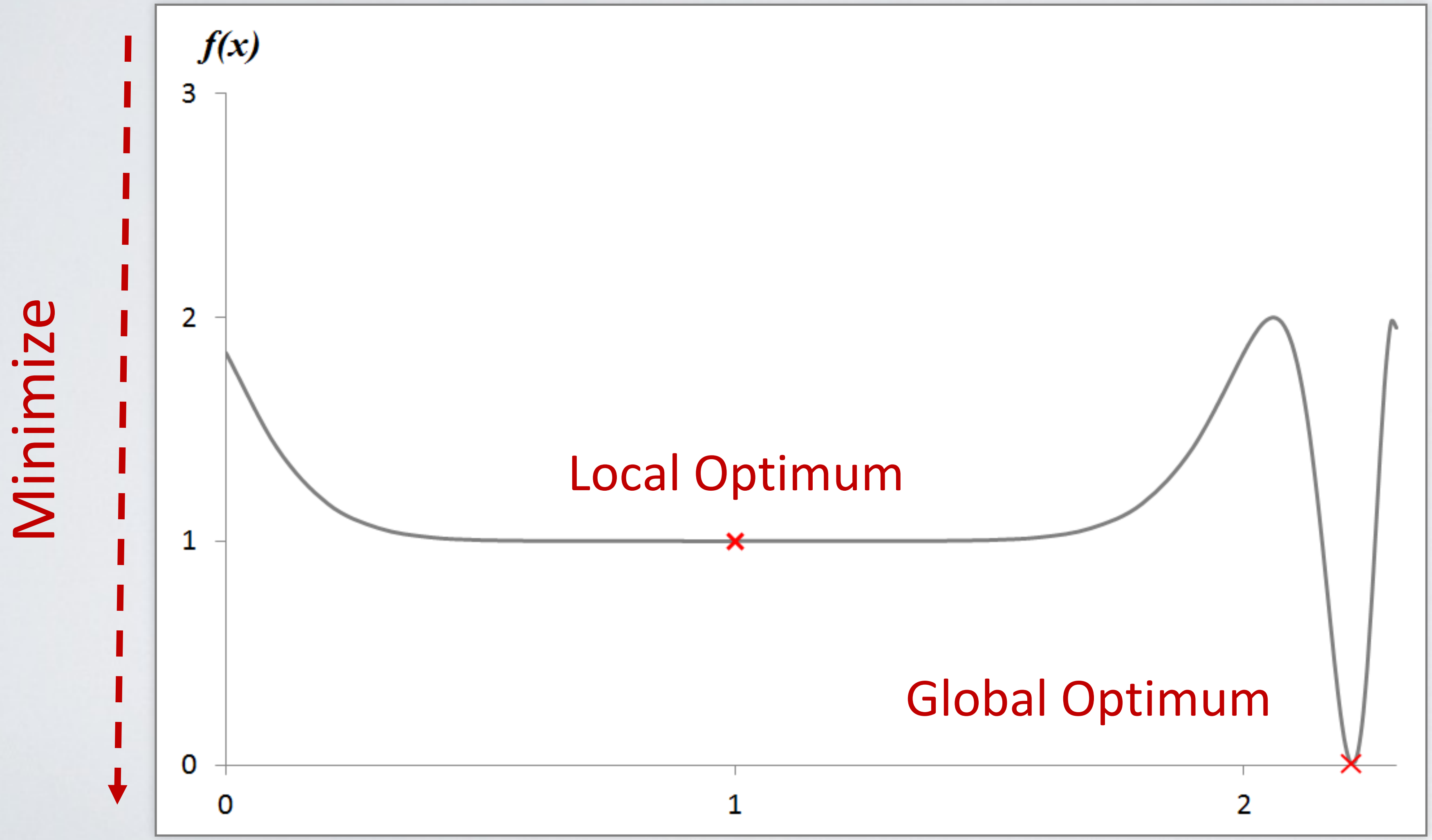
- After decades of research, there were no scalable, practical solutions for many automation problems.
- The community realized that many automation problems could be re-expressed as search problems.
- Stochastic optimization, Meta-heuristic search.
- Increasing realization that Search-Based Software Engineering has a much wider potential and is a research topic in itself.

Harman and Jones, “Search-Based Software Engineering”, 2001

Optimization

Find a value ***x**** which minimises (or maximises) the objective/fitness function ***f*** over a search space ***X***:

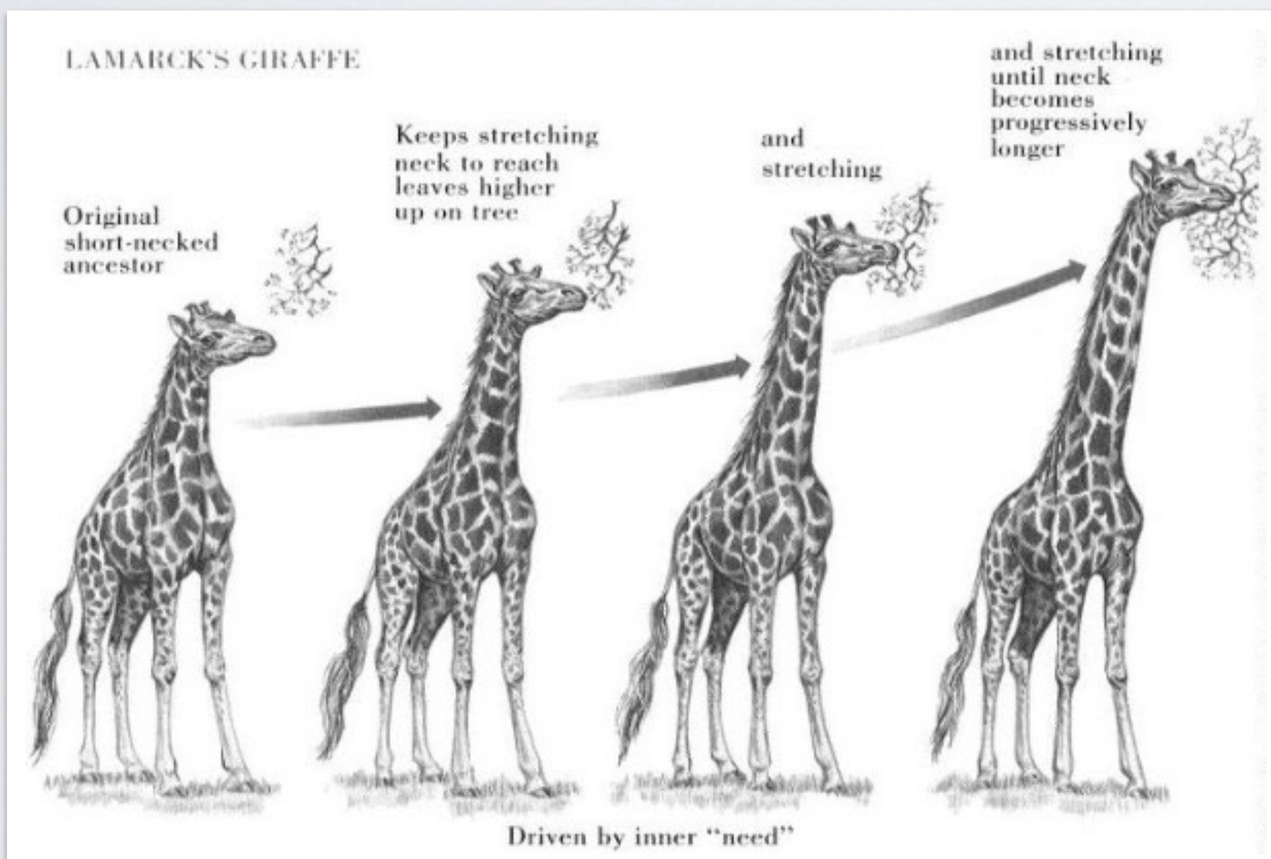
$$\forall x \in X : f(x^*) \leqslant f(x)$$



- No closed-form analytical description
- Black-box optimization
- Meta-heuristic search

Genetic Algorithms (GAs)

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

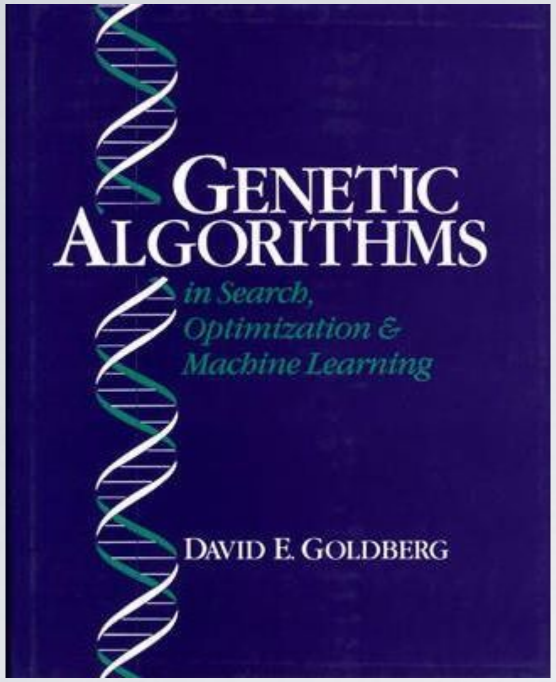


Natural selection: Individuals that best fit the natural environment survive

Reproduction: surviving individuals generate offsprings (next generation)

Mutation: offsprings inherits properties of their parents with some mutations

Iteration: generation after generation the new offspring fit better the environment than their parents



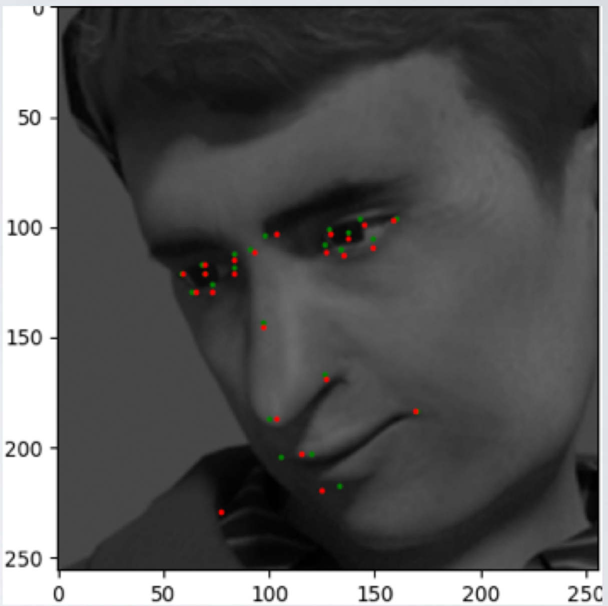
~1995

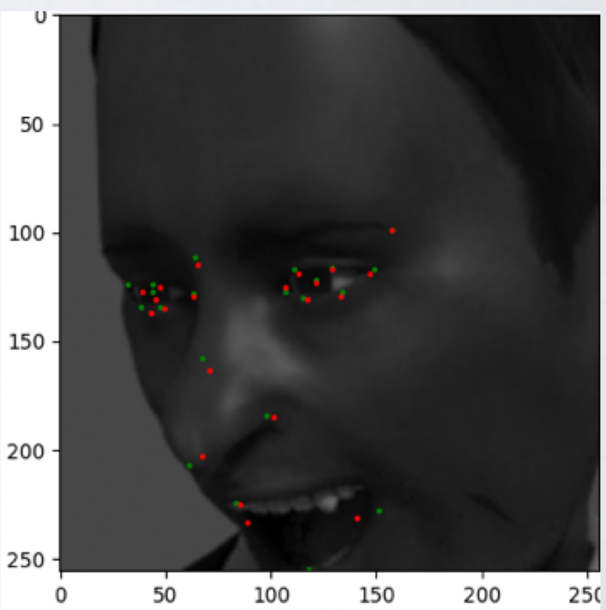
The first time I read about genetic algorithms, meta-heuristic search …

My first reaction:
“You must be kidding”

Example: Key-points Detection

- Automatically detecting key-points in an image or a video, e.g., face recognition, drowsiness detection
  - **Key-point Detection DNNs (KP-DNNs)** are widely used to detect key-points in an image
- It is essential to check how accurate KP-DNNs are when applied to various test data





Ground truth
Predicted

Problem Definition

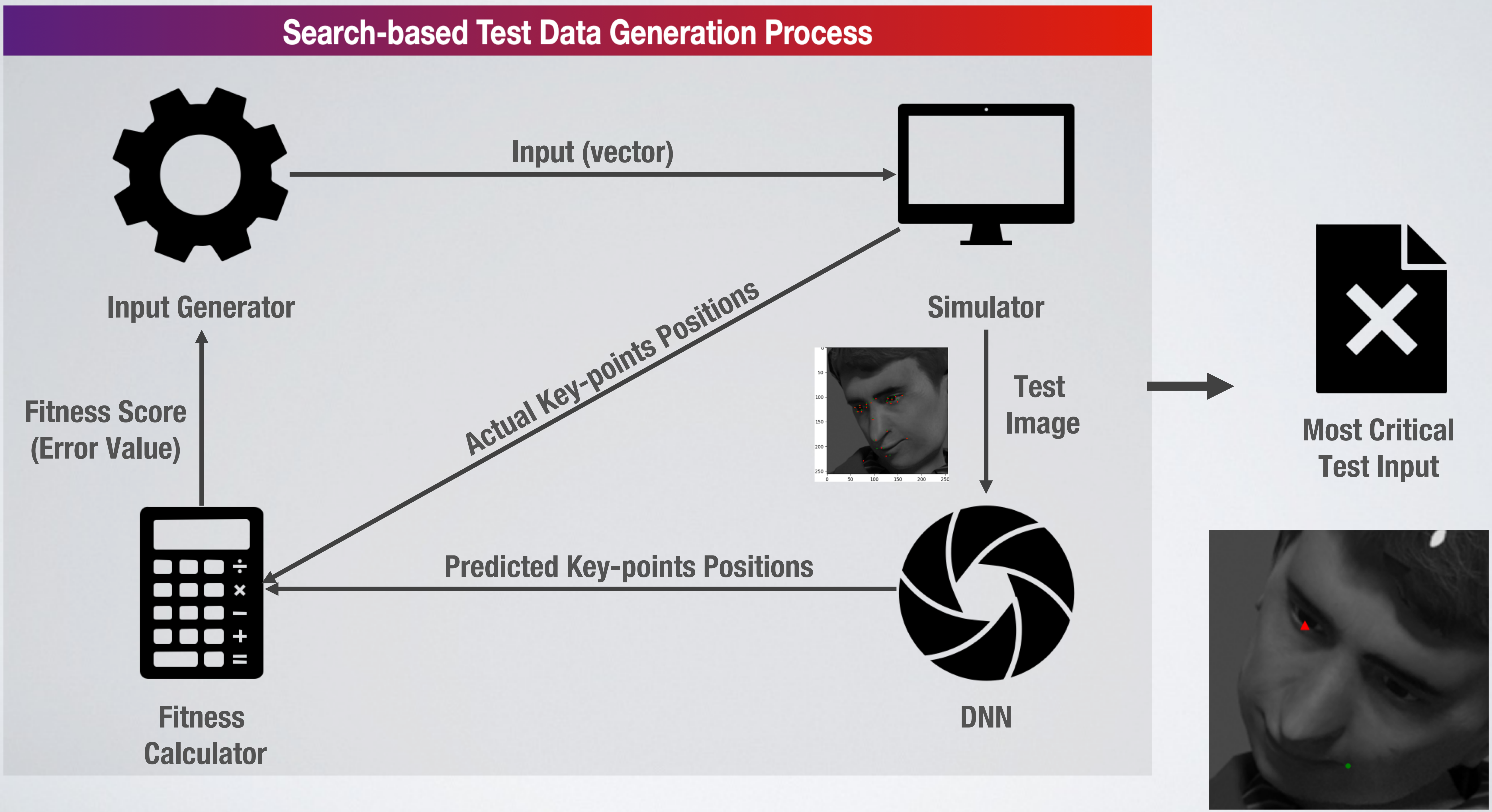
- In the drowsiness or gaze detection problem, each Key-Point (KP) may be highly important for safety
  - Each KP leads to a requirement and test objective
  - For our subject DNN, we have 27 requirements
- Goal: cause the DNN to mis-predict as many key-points as possible
- Solution: many-objective search algorithms combined with simulator

Donghwan Shin

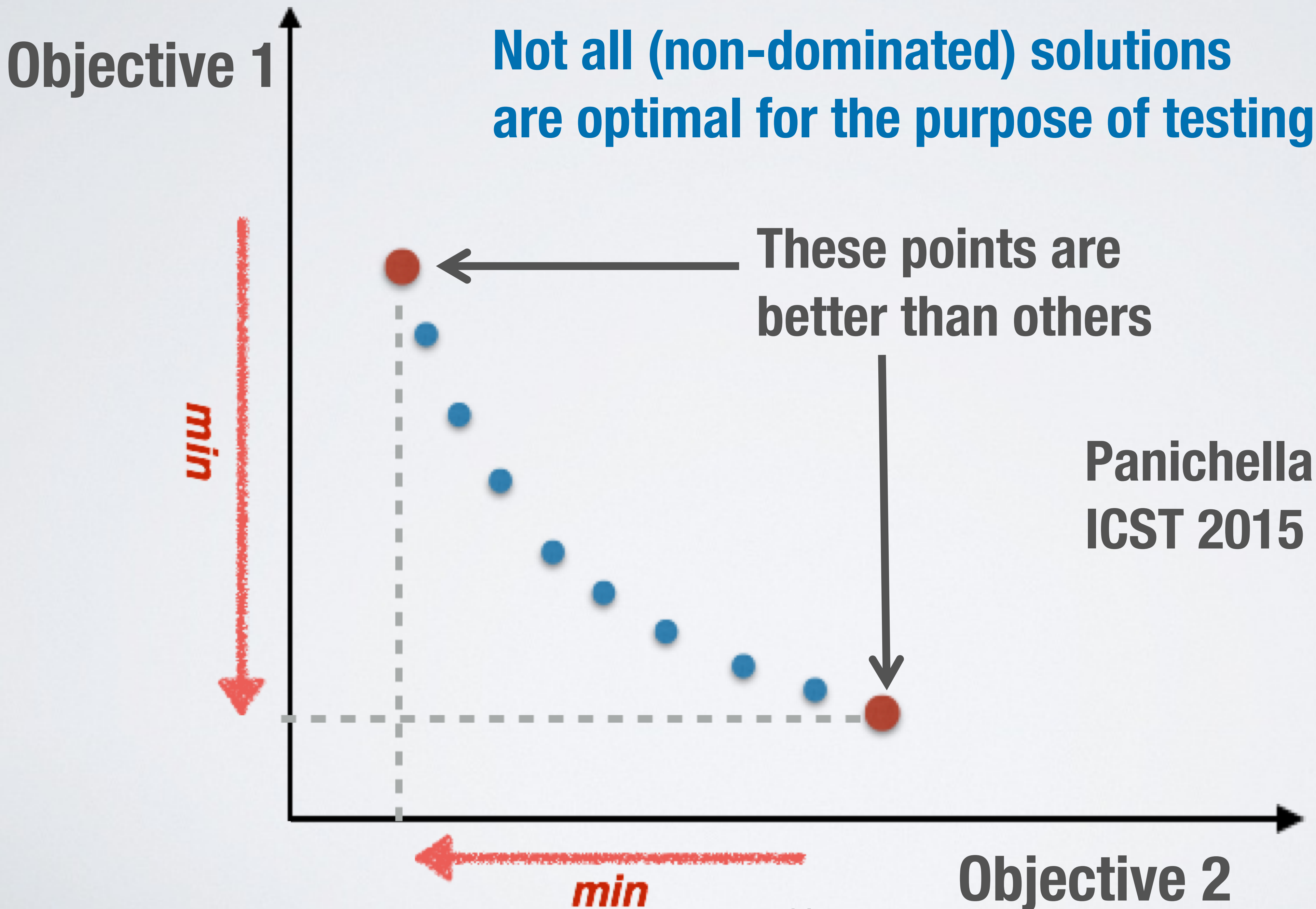
Fitash Ul Haq

Ul Haq et al., “Automatic Test Suite Generation for Key-points Detection DNNs Using Many-Objective Search”, ACM ISSTA 2021

Overview


Search-based Test Data Generation Process
Input (vector)
Input Generator
Simulator
Actual Key-points Positions
Test Image
Fitness Score (Error Value)
Predicted Key-points Positions
Fitness Calculator
DNN
Most Critical Test Input

MOSA: Many-Objective Search-based Test Generation



Panichella et. al.
ICST 2015

Results

- Our approach is effective in generating test suites that cause the DNN to severely mispredict more than 93% of all key-points on average
- Not all mispredictions can be considered failures …
- Some key-points are more severely predicted than others, detailed analysis revealed two reasons:
  - Under-representation of some key-points (hidden) in the training data
  - Large variation in the shape and size of the mouth across different 3D models (more training needed)

Interpretation

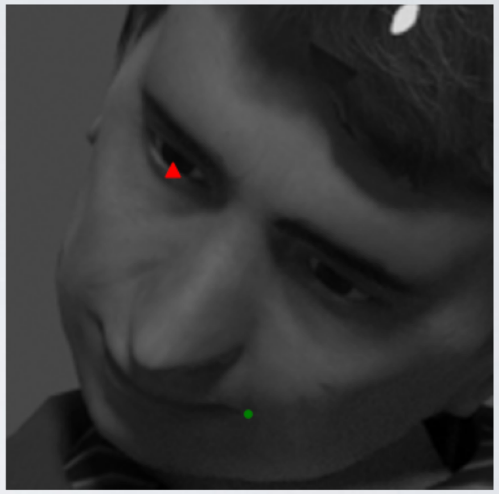
Representative rules derived from the regression tree for KP26
(M: Model-ID, P: Pitch, R: Roll, Y: Yaw, NE: Normalized Error)

Image Characteristics Condition	NE
$M = 9 \land P < 18.41$	0.04
$M = 9 \land P \geq 18.41 \land R < -22.31 \land Y < 17.06$	0.26
$M = 9 \land P \geq 18.41 \land R < -22.31 \land 17.06 \leq Y < 19$	0.71
$M = 9 \land P \geq 18.41 \land R < -22.31 \land Y \geq 19$	0.36

(A) A test image satisfying the first condition

NE = 0.013

(B) A test image satisfying the third condition



NE = 0.89

- **Regression trees**
- **Detailed analysis to find the root causes of high NE values, e.g., shadow on the location of KP26 is the cause of high error (NE) values**
- **The average MAE from all the trees is 0.01 (far less than the pre-defined threshold: 0.05) with average tree size of 25.7. Excellent accuracy, reasonable size.**

SBSE Applications

- **Many applications turned out to be promising**: Requirements prioritization, refactoring, test automation, program repair, etc.
- **My first SBSE paper:** “Using genetic algorithms and coupling measures to devise optimal integration test orders.” SEKE’02
- Many articles since then …

SBSE: Benefits

- Effective automation mechanism for a wide set of SE problems.
- Potentially **high scalability**: No exhaustive search.
- Can be effective under certain conditions: Search landscape, fitness computation, etc.
- Can be **effectively parallelized**.

SBSE: Challenges

- Performance can be an issue in large, high-dimensionality search spaces
- Computationally expensive fitness functions, e.g., simulations.
- Validation is experimental and computationally expensive.
- Many problems require dedicated, tailored search algorithms, e.g., many-objective search in testing.
- Devising the right search algorithm for a given problem requires expertise and experiments.
- Devising the right fitness functions is often a trade-off and is a trial and error process.

> ~2004

Increasingly Powerful Natural Language Processing

NLP

- Process and analyze large amounts of natural language data.
- Rule-based versus statistical NLP (based on machine learning).
- Preprocessing: Tokenizer, sentence splitter, POS tagger.
- Parsing: Constituency, dependency, semantic.
- NLP has made huge leaps forward (e.g., language models).

Arnaoudova et al., "The Use of Text Retrieval and Natural Language Processing in Software Engineering", ICSE'15

Why NLP?

- Significant documentation of many kinds in natural language …

- Example: NL Requirements

  - are **prominent** throughout industry sectors, even safety-critical ones,

  - are **not fading away** any time soon.

Why NLP?

- Check **well-formedness** of NL artifacts
- Extract useful **information** from NL artifacts
- Check **consistency and completeness** of NL artifacts
- Understand **relationship and dependencies** between NL artifacts (e.g., traceability)

Experiences in Requirements Engineering

- Conformance of requirements with **templates**. (Arora et al.)
- **Impact analysis** of requirements changes (Arora et al., Nejati et al.)
- Identification and **demarcation** of requirements in large documents. (Abualhaija et al.)
- Requirements-driven **system testing**. (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

Traceability

- In many sectors, traceability between requirements and test cases is **required by standards**, customers, certifiers ...
- Requirements **change**, and therefore test cases as well.
- Huge traceability **matrices** are built and maintained manually.
- Academic work on **automatically matching** requirements and test cases is not sufficiently accurate or practical.

Problem

Automatically verify the compliance of software systems with their functional requirements in a cost-effective way

Objective

Support the Generation of System Test Cases from Requirements in Natural Language

Traceability is a by-product

Chunhui Wang

Fabrizio Pastore

Wang et al., “Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach”, IEEE TSE, 2020

Problem

Textual descriptions are often ambiguous, Incomplete, and not analyzable automatically

Compromise?

Stick to natural language but …

Restrict its usage so as to make it amenable to NLP for system testing purposes

Find the right balance

Restricted Use Case Specifications

- Use Case Modeling is widely used
- Restricted Use Case Modeling (RUCM)
- Experiments: RUCM yields better use cases
- Compliance is tool-supported (NLP)
- More analyzable with NLP

Yue et al. ACM TOSEM, 2013

RUCM Specifications Example

Precondition: The system has been initialized

Basic Flow

INPUT STEP

1. The SeatSensor **SENDS** the weight **TO** the system.

INCLUDE STEP

2. **INCLUDE USE CASE** Self Diagnosis.

CONDITIONAL STEP

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.

INTERNAL STEP

6. The system **SENDS** the occupancy status **TO** AirbagControlUnit.

OUTPUT STEP

RUCM Specifications Example

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.

INPUT STEP

INCLUDE STEP

CONDITIONAL STEP

INTERNAL STEP

OUTPUT STEP

Alternative Flow

RFS 4.

1. **IF** the weight is above 1 Kg **THEN**
2. The system sets the occupancy status to child.
3. …
4. **RESUME STEP** 6.

NLP for information extraction

Precondition: The system has been initialized

Basic Flow

1. The SeatSensor **SENDS** the weight **TO** the system.

DOMAIN ENTITY

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.

Alternative Flow

RFS 4.

1. **IF** the weight is above 1 Kg **THEN**
2. The system sets the occupancy status to child.
3. …
4. **RESUME STEP 6.**

NLP for information extraction

Precondition: The system has been initialized

Basic Flow

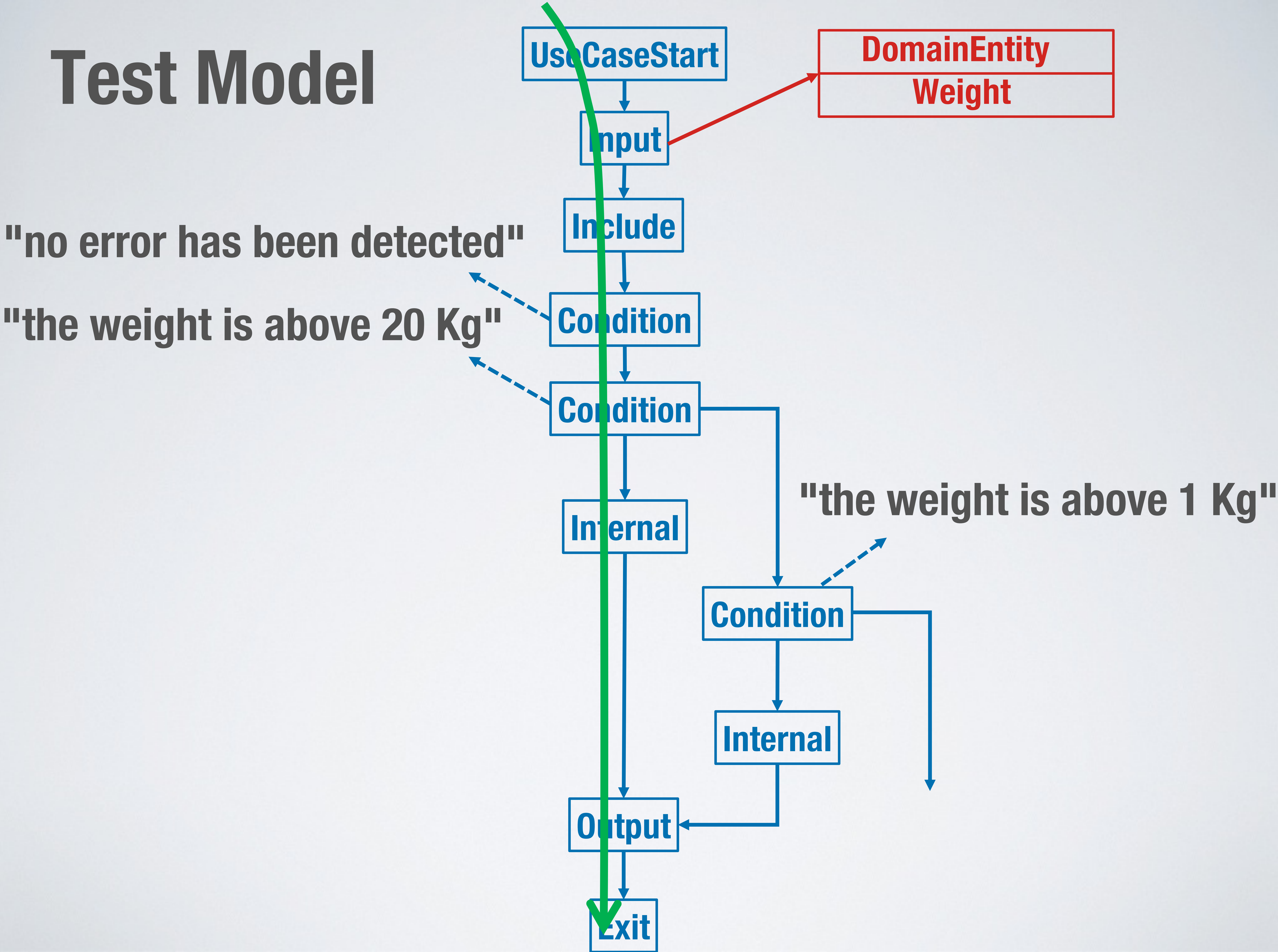
1. The SeatSensor **SENDS** the weight **TO** the system.
2. **INCLUDE USE CASE** Self Diagnosis **CONSTRAINT**
3. The system **VALIDATES THAT** no error has been detected.
4. The system **VALIDATES THAT** the weight is above 20 Kg.
5. The system sets the occupancy status to adult.
6. The system **SENDS**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

Alternative Flow

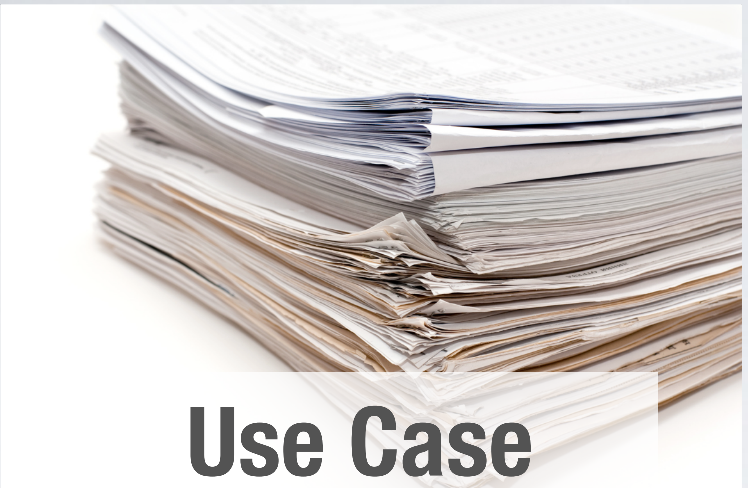
RFS 4.

1. **IF** the weight is above 1 Kg **THEN**
2. The system sets the occupancy status to child.
3. …
4. **RESUME STEP 6**.

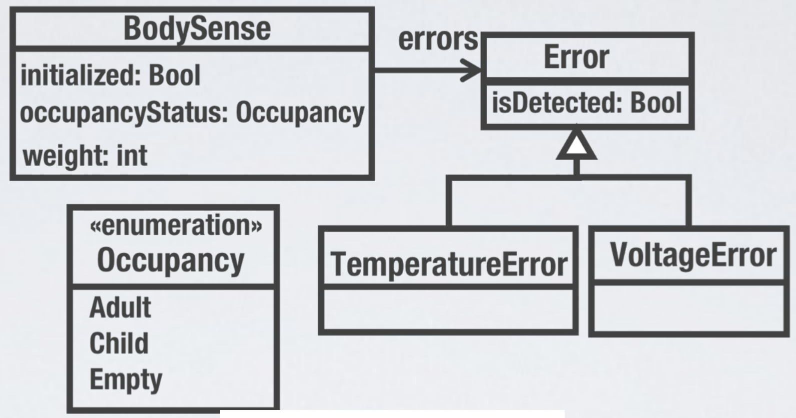
Test Model


UseCaseStart
DomainEntity
Weight
Input
Include
"no error has been detected"
"the weight is above 20 Kg"
Condition
Condition
Internal
"the weight is above 1 Kg"
Condition
Internal
Output
Exit

Automated Generation of System Test Cases for Embedded Systems from Requirements in NL



Use Case Specifications



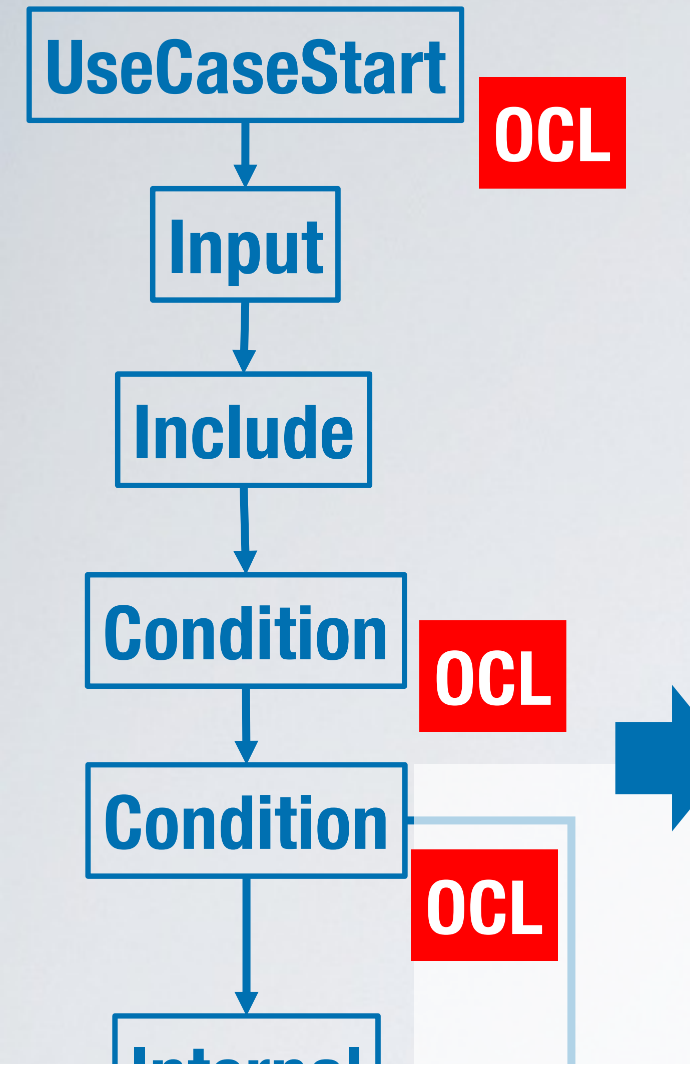
Domain Model

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

Constraints capturing the meaning of conditions

(Wang et al.) **UMTG** https://sntsvv.github.io/UMTG/

Executable Test Cases



Precondition: The system has been initialized.

The SeatSensor **SENDS** the weight **TO** the system.

Path condition:

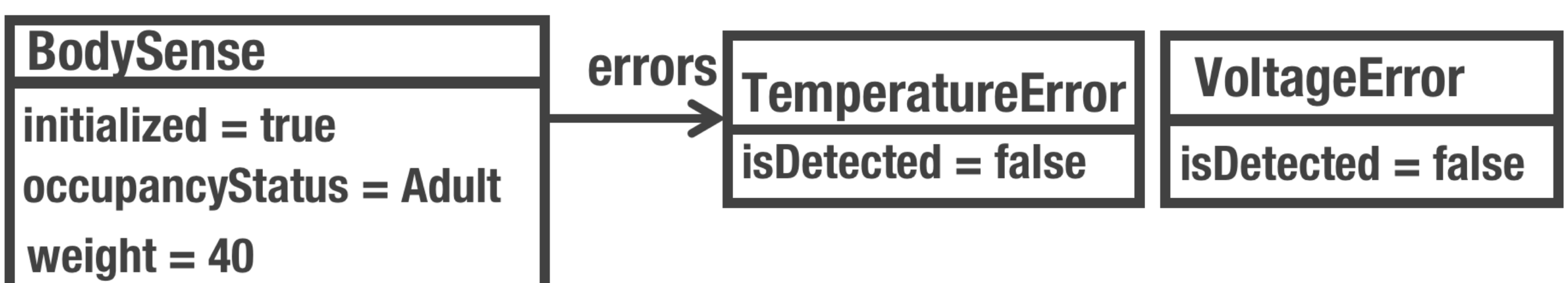
```
System.allInstances()->forAll( s | s.initialized = true )
AND System.allInstances()->forAll( s | s.initialized = true )
AND Error.allInstances()->forAll( e | e.isDetected = false)
AND System.allInstances()
        ->forAll( s | s.occupancyStatus = Occupancy::Adult )
```

https://sites.google.com/view/hybridoclsolver/

Soltana et al.

Constraint Solving (PLEDGE)

Test inputs:

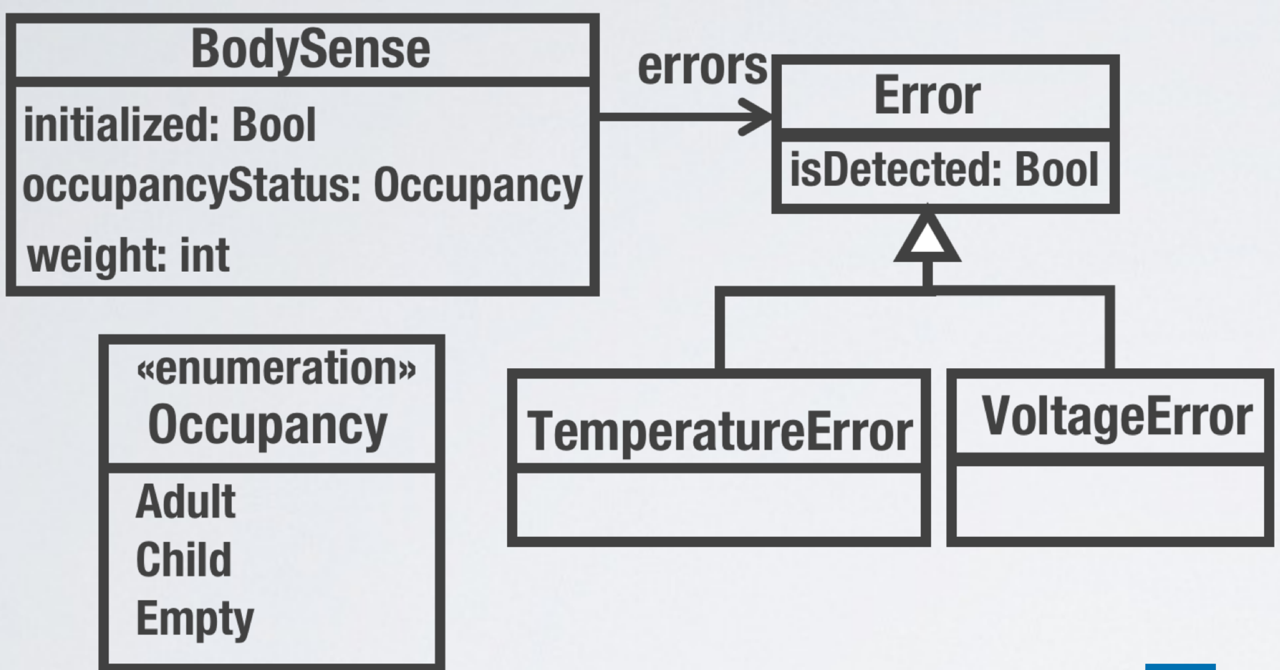


Challenge

Typically dozens of constraints

Engineers need help in defining constraints

Automatically Derive Formal Constraints



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

(Wang et al., 2020)

OCLgen

Based on NLP

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

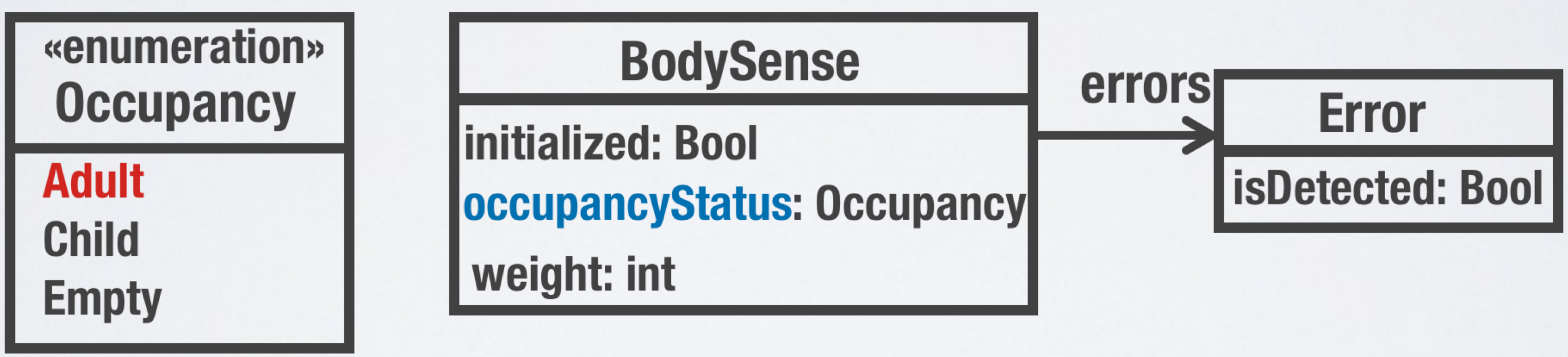
OCLgen Solution

1. determine the role of words in a sentence (Semantic Role Labeling)

actor affected by the verb   final state

"The system sets the occupancy status to adult."

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



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

```
BodySense.allInstances()
    ->forAll( i | i.occupancyStatus = Occupancy::Adult)
```

NLP in SE: Summary

- Increasingly powerful, many applications
- Wide variation across domain practices and documents.
- Inherent ambiguity and inconsistency of natural language.
- Relevant data is usually spread across artifacts.
- Designing the right NLP pipeline in not easy.
- NLP components are not fully accurate.
- Human in the loop.

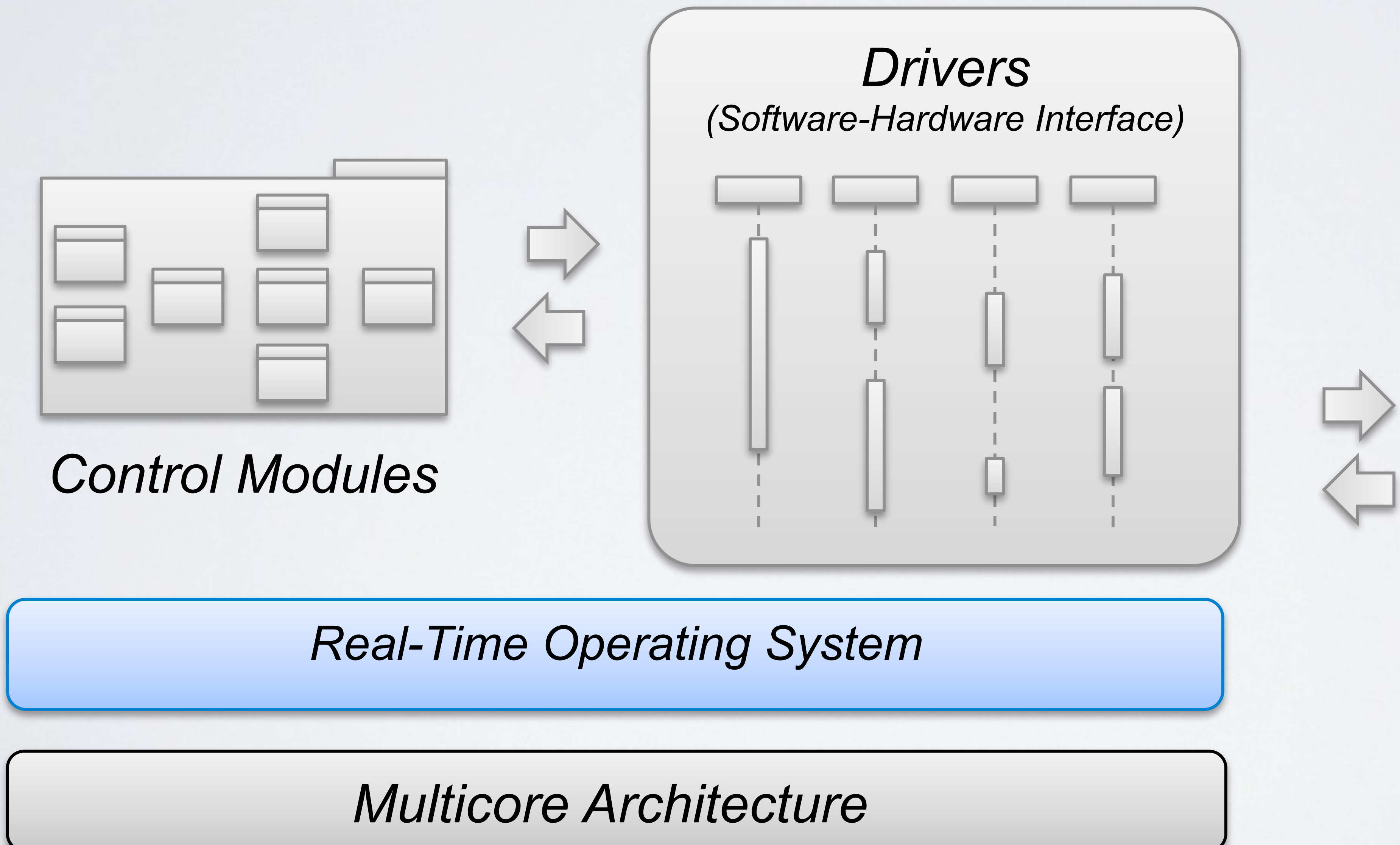
Combining Strengths

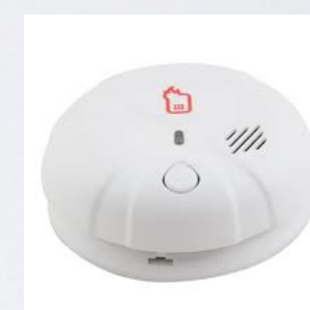
Multidisciplinary Approaches

- Single-technology approaches rarely work in practice
  - Meta-heuristic search, Machine learning
  - NLP
  - Solvers, e.g., CP, SMT
  - Statistical approaches, e.g., sensitivity analysis
  - System and environment modeling and simulation

Search+CP Example

System monitors gas leaks and fire in oil extraction platforms





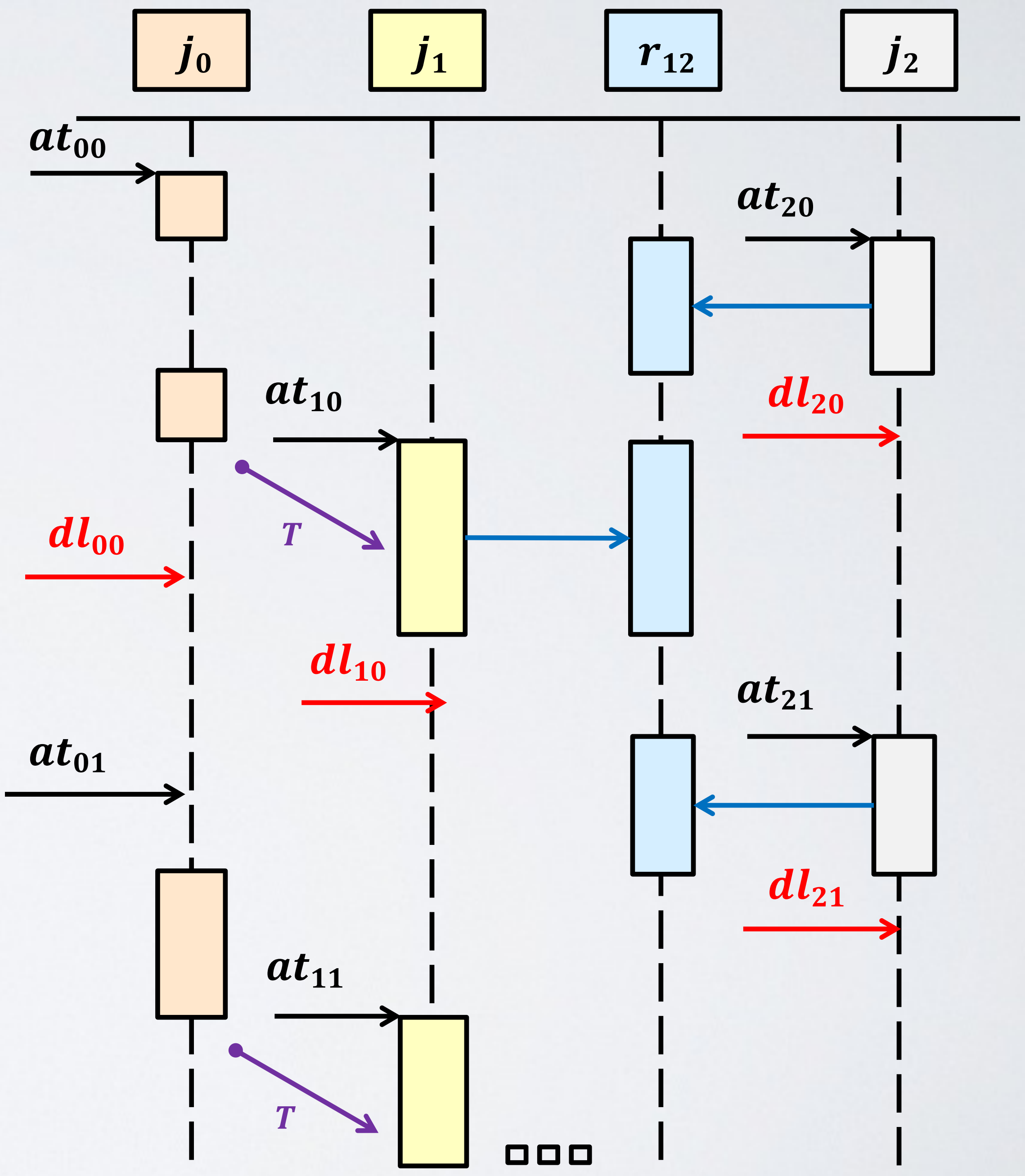
Alarm Devices
(Hardware)

RTES: Concurrent Tasks

Each task has a deadline (i.e., latest finishing time) w.r.t. its arrival time

Some task properties depend on the environment, some are design choices

Tasks can trigger other tasks, and can share computational resources with other tasks



Stress Testing

Stress Testing: *"Testing in which a system is subjected to [...] harsh inputs [...] with the intention of breaking it"*

— **Boris Beizer**

Arrival times for aperiodic tasks

Worst-case scenarios wrt. missing deadlines

Table B.6 – Performance testing
(referenced by tables A.5 and A.6)

	Technique/Measure*	Ref	SIL1	SIL2	SIL3	SIL4
1	Avalanche/stress testing	C.5.21	R	R	HR	HR
2	Response timings and memory constraints	C.5.22	HR	HR	HR	HR
3	Performance requirements	C.5.19	HR	HR	HR	HR

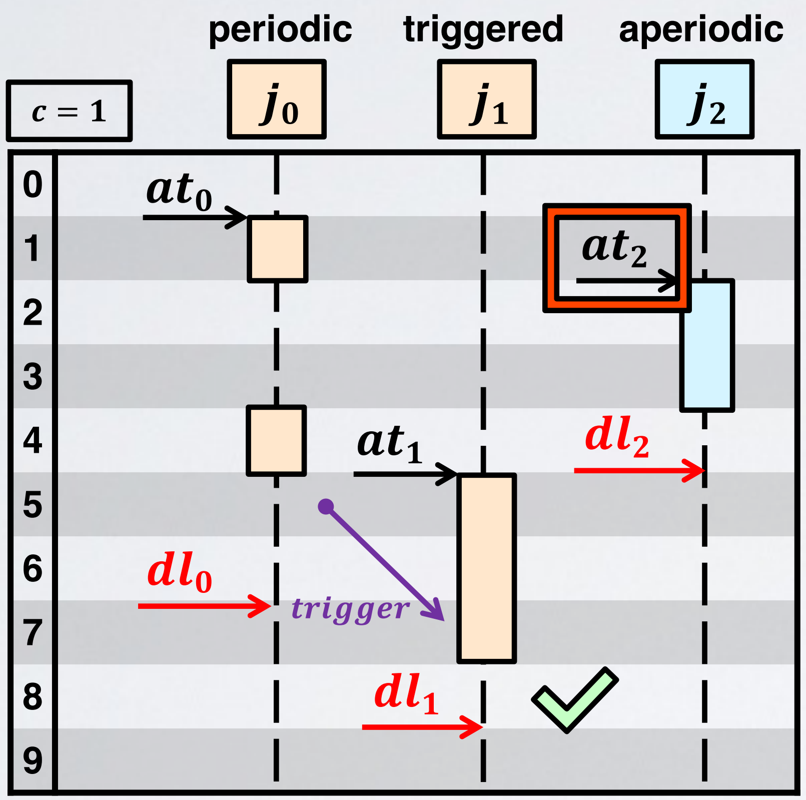
* Appropriate techniques/measures shall be selected according to the safety integrity level.

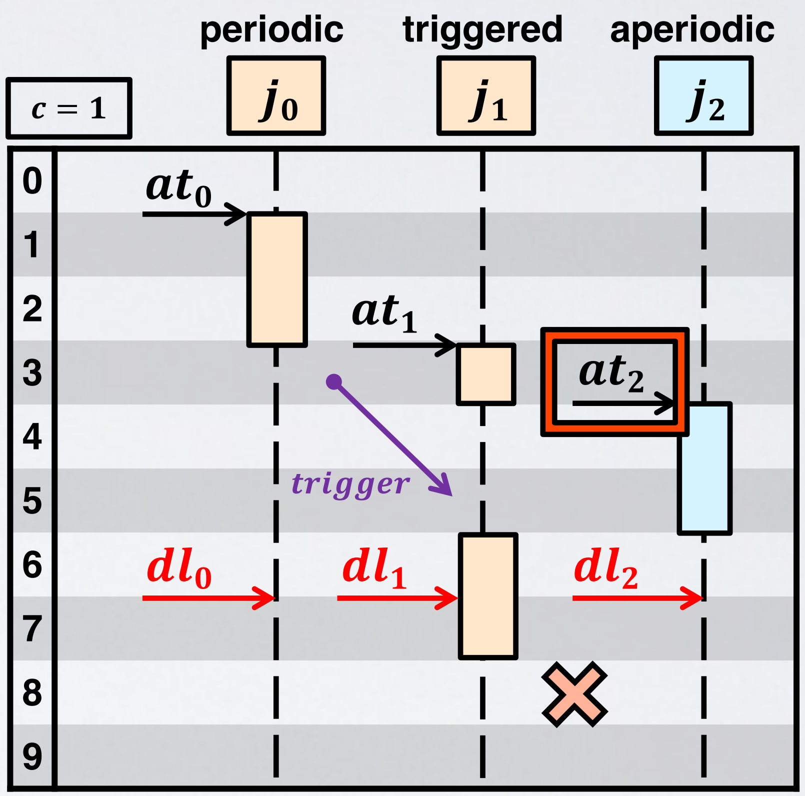
IEC 61508 deems stress testing as *highly recommended* for SIL 3-4

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!





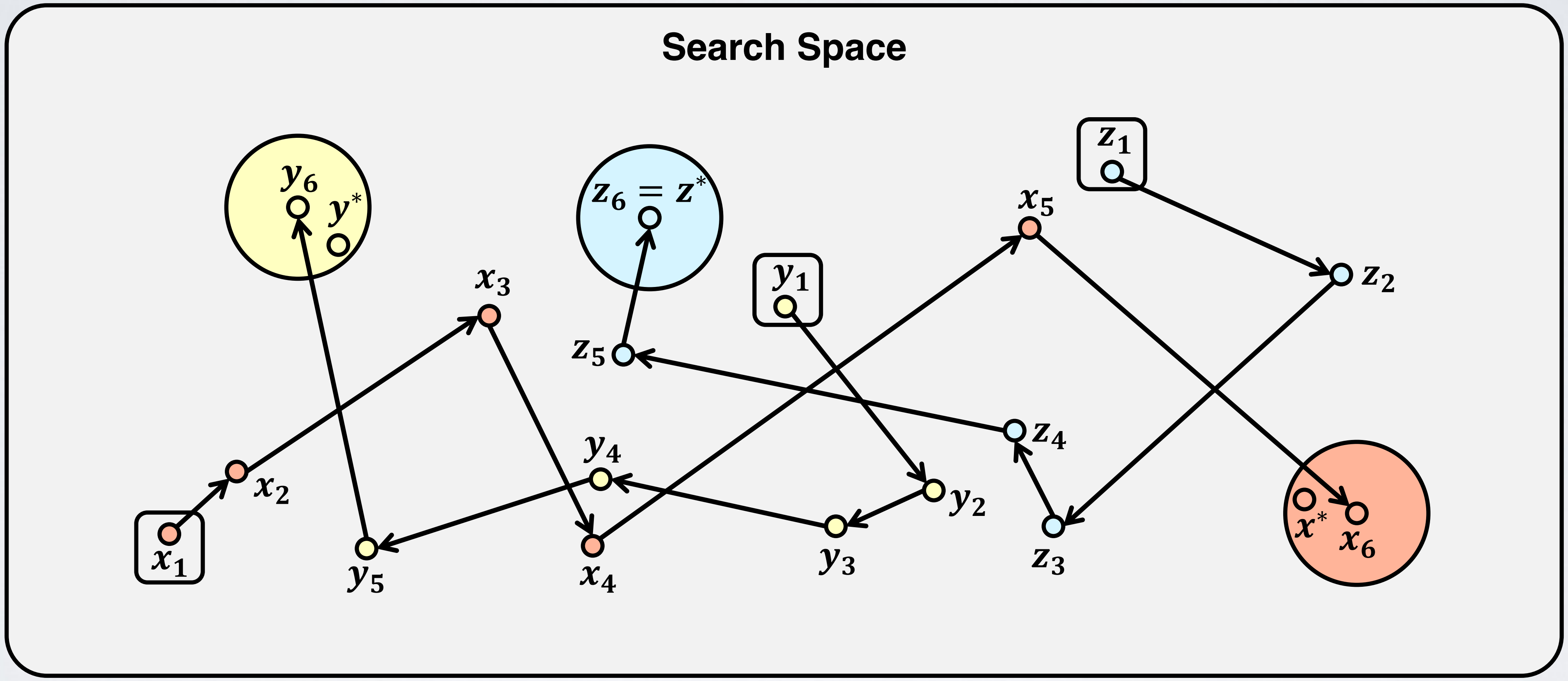
A sequence of arrival times which is likely to violate a task deadline characterizes a *stress test case*

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**

Combining CP and GA

The key idea behind GA+CP is to run complete searches with CP in the neighbourhood of solutions found by GA



Stefano Di Alesio

Shiva Nejati

Di Alesio et al., "Combining genetic algorithms and constraint programming to support stress testing of task deadlines", ACM TOSEM 2015

Conclusions

The Road Ahead

- AI plays a key role in automating many software engineering tasks and helping decision support

- Real solutions usually involve several techniques, combined to achieve the best trade-offs.

- Real solutions strike a balance in terms of scalability, practicality, applicability, and optimal results.

- Research in this field cannot be oblivious to context (e.g., domain): Working assumptions, desirable trade-offs …

- We need more multi-disciplinary research driven by (well-defined) problems in context.

uOttawa

Canada Research Chairs
Chaires de recherche du Canada
Canada

AI and Software Engineering: Past, Present, and Future

Lionel Briand

http://www.lbriand.info

CSER 2021

References

Selected References

- Arisholm et al., “A systematic and comprehensive investigation of methods to build and evaluate fault prediction models”, Journal of Systems and Software, 2010
- 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., “Practical Constraint Solving for Generating System Test Data”, ACM TOSEM, 2020
- Ul Haq et al., “Automatic Test Suite Generation for Key-points Detection DNNs Using Many-Objective Search”, ACM ISSTA 2021
- Yue et al., “Facilitating the transition from use case models to analysis models: Approach and experiments.”, ACM TOSEM, 2013
- Wang et al., “Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach”, IEEE TSE (accepted), 2020

Other References

References: CPS testing

- Matinnejad et al., “MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models”, ASE 2014
- Matinnejad et al., “Automated Test Suite Generation for Time-Continuous Simulink Models”, ICSE 2016.
- Matinnejad et al., “Test Generation and Test Prioritization for Simulink Models with Dynamic Behavior”, IEEE Transactions on Software Engineering, 2018
- Liu et al., “Effective Fault Localization of Automotive Simulink Models: Achieving the Trade-Off between Test Oracle Effort and Fault Localization Accuracy”, Empirical Software Engineering (Springer), 2019
- Liu et al., “Simulink Fault Localisation: An Iterative Statistical Debugging Approach”, Software Testing, Verification & Reliability (Wiley), 2016
- 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
- Nejati et al., “Evaluating Model Testing and Model Checking for Finding Requirements Violations in Simulink Models”, ESEC/FSE 2019.

References: System Testing

- Wang et al., “Automatic generation of system test cases from use case specifications”, ISSTA 2015
- Wang et al., “Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach”, IEEE TSE (accepted), 2020
- Wang et al., “Oracles for Testing Software Timeliness with Uncertainty”, ACM TOSEM, 2019
- Shin et al., “Test case prioritization for acceptance testing of cyber-physical systems”, ISSTA 2018
- Shin et al.,“HITECS: A UML Profile and Analysis Framework for Hardware-in-the-Loop Testing of Cyber Physical Systems”, MODELS 2018
- Soltana et al., “Practical Constraint Solving for Generating System Test Data”, ACM TOSEM, 2020
- Shin et al., “Dynamic Adaptive Network Configuration for IoT Systems: A Search-based Approach”, ACM/IEEE SEAMS, 2020

References: Stress Testing

- Shousha et al., "Using Genetic Algorithms for Early Schedulability Analysis and Stress Testing in Real-Time Systems", Journal of Genetic Programming and Evolvable Machines (Springer), 2006.
- Shousha et al., “A UML/MARTE Model Analysis Method for Uncovering Scenarios Leading to Starvation and Deadlocks in Concurrent Systems”, IEEE Transactions on Software Engineering, 2012.
- Nejati and Briand, “Identifying Optimal Trade-Offs between CPU Time Usage and Temporal Constraints Using Search”, ISSTA 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

References: Security Testing

- Jan et al., Automatic Generation of Tests to Exploit XML Injection Vulnerabilities in Web Applications. IEEE Transactions on Software Engineering (TSE), 2019
- Appelt et al., A Machine Learning-Driven Evolutionary Approach for Testing Web Application Firewalls. IEEE Transactions on Reliability (TR), 2018
- Appelt et al., Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks. IEEE International Symposium on Software Reliability Engineering (ISSRE 2017)
- Jan et al., Search-based Testing Approach for XML Injection Vulnerabilities in Web Applications. IEEE International Conference on Software Testing, Verification and validation (ICST 2017)
- Jan et al., Automated and Effective Testing of Web Services for XML Injection Attacks. International Symposium on Software Testing and Analysis (ISSTA 2016)
- Ceccato et al., SOFIA: An Automated Security Oracle for Black-Box Testing of SQL-Injection Vulnerabilities, IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)

References: Requirements

- Arora et al. “Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation”, MODELS 2016
- Arora et al. “Automated Extraction and Clustering of Requirements Glossary Terms”, IEEE TSE, 2017
- Arora et al., “An Active Learning Approach for Improving the Accuracy of Automated Domain Model Extraction”, ACM TOSEM, 2019
- Nejati et al., “Automated Change Impact Analysis between SysML Models of Requirements and Design”, FSE 2016
- Abualhaija et al., “A Machine-Learning Approach for Demarcating Requirements in Textual Specifications”, RE 2019
- Bettaieb et al., “Decision Support for Security-Control Identification Using Machine Learning”, REFSQ 2019
- Sleimi et al., “ A Query System for Extracting Requirements-related Information from Legal Texts”, RE 2019
- Sleimi et al., “Automated Extraction of Semantic Legal Metadata using Natural Language Processing”, RE 2018