Automated Analysis of Natural-Language Requirements:

Industrial Needs and Opportunities

AIRE'2018 @ RE'2018

Lionel Briand
Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT)
University of Luxembourg, Luxembourg

Acknowledgements

- Mehrdad Sabetzadeh
- Chetan Arora
- Shiva Nejati
- Fabrizio Pastore
- Chunhui Wang

Introduction

Applications

- Requirements to support a shared understanding among stakeholders in (large) projects, e.g., software engineers and domain experts
- Requirements as contract with customers
- Requirements to support compliance with standards, e.g., traceability to tests
- Requirements to support quality assurance, e.g., system (security) testing
- Requirements to support change control
- Requirements to support product-line configuration

For many of these applications, there is little automated tool support.

Forms of NL Requirements

- **Natural language** statements, complying or not with templates
- **User stories**, following various templates
- **Use case specifications**, possibly structured and restricted
- **Mixing natural language and models**, e.g., class and activity diagrams

The best form of requirements depends on context and targeted applications.

Contextual Factors

- No “right” way to express requirements

- Domain complexity and criticality
- Regulatory compliance, e.g., standards
- Project size, team distribution, and number of stakeholders
- Background of stakeholders and communication challenges
- Presence of product lines with multiple customers
- Importance of early contractual agreement
- Frequency and consequences of changes in requirements

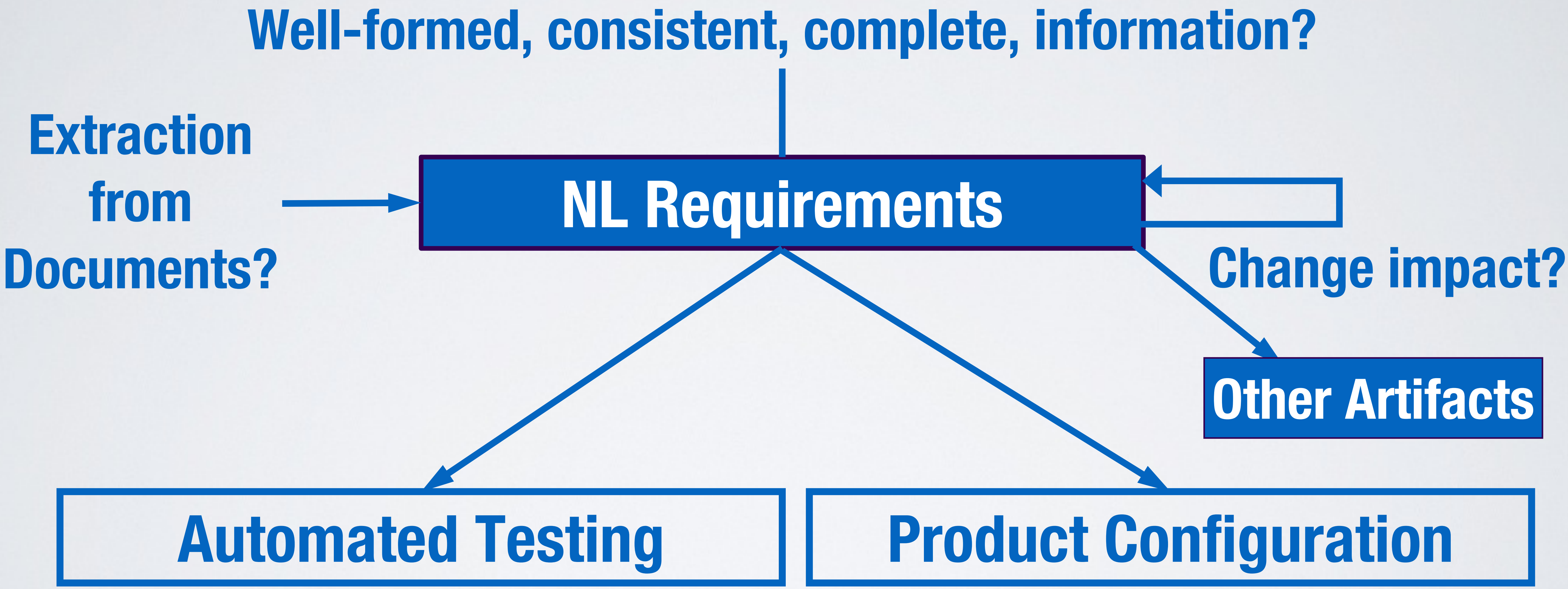
Observations

- Natural language in requirements **won't go away.**
- The cost of rigorous requirements engineering must be justified by **clear automation benefits**.
- **Limited research** given the many and varying industrial automation needs, in widely varying contexts.

Outline

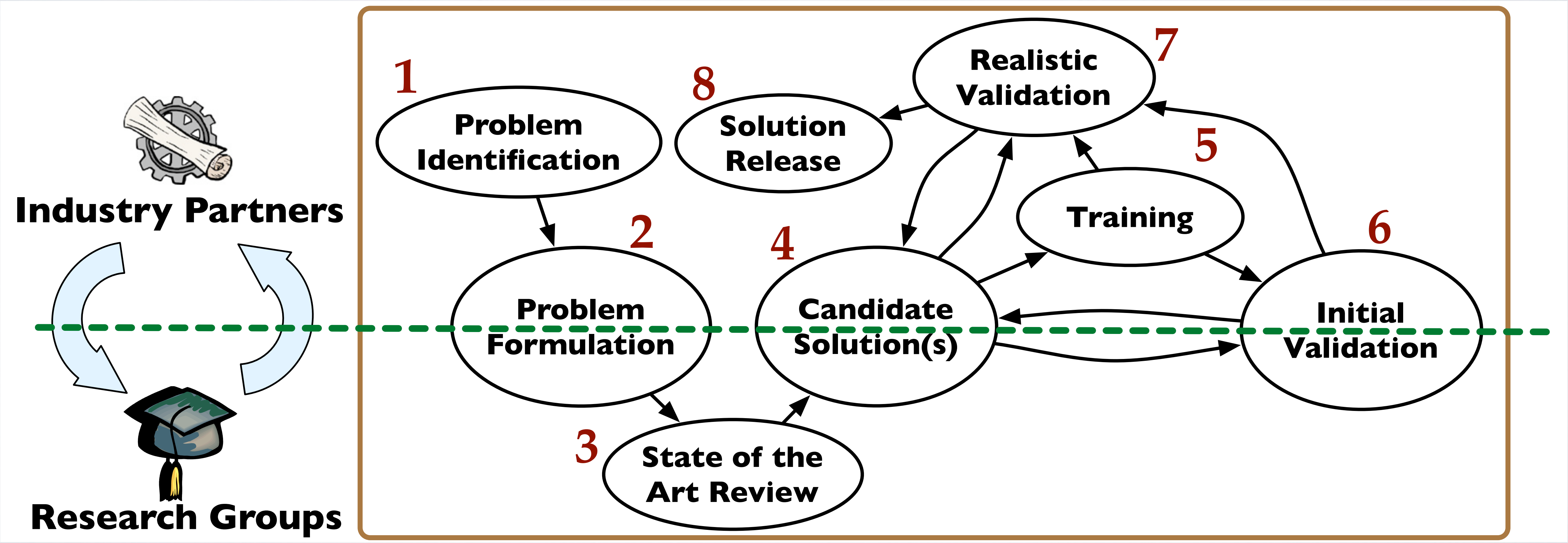
- Report on a variety of research projects
- Collaborations with industry
- Various objectives and applications
- Examples from automotive and satellite
- Lessons learned and reflections

Overview


Well-formed, consistent, complete, information?
Extraction from Documents?
NL Requirements
Change impact?
Other Artifacts
Automated Testing
Product Configuration

Mode of Collaboration

- Research driven by industry needs
- Realistic evaluations
- Combining research with innovation and technology transfer



Adapted from [Gorschek et al. 2006]

Change Impact Analysis

Supporting Change

- Requirements change frequently
- Changes have **side-effects** on other requirements, design decisions, test cases …
- How do we **support such changes** in ways that scale to hundreds of requirements or more?
- **Automated impact analysis**

Inter-Requirements

Inter-Requirements
Change Impact Analysis

Approach

- Hundreds of requirements
- No traceability
- We propose an approach based on: (1) Natural Language Processing, (2) Phrase syntactic and semantic similarity measures

Example

- **R1:** The mission operation controller shall transmit satellite status reports to the user help desk.
- **R2:** The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk.
- **R3:** The mission operation controller shall transmit any detected anomalies with the user help desk.

Change

- **R1:** The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.
- **R2:** The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk.
- **R3:** The mission operation controller shall transmit any detected anomalies with the user help desk.

Challenge #1
Capture Changes Precisely

- R1: The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.
- R2: The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk.
- R3: The mission operation controller shall transmit any detected anomalies with the user help desk.

Challenge #2
Capture Change Rationale

- R1: The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.
- R2: The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk.
- R3: The mission operation controller shall transmit any detected anomalies with the user help desk.

Challenge #2
Change Rationale

- R1: The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.
- R2: The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk.
- R3: The mission operation controller shall transmit any detected anomalies with the user help desk.

Possible Rationales:

1: We want to globally rename "user help desk"
2: Avoid communication between "mission operation controller" and "user help desk" (R3)
3: We no longer want to "transmit satellite status reports" to "user help desk" but instead to "user document repository" (only R1)

Determine conditions under which change should propagate

Solution Characteristics

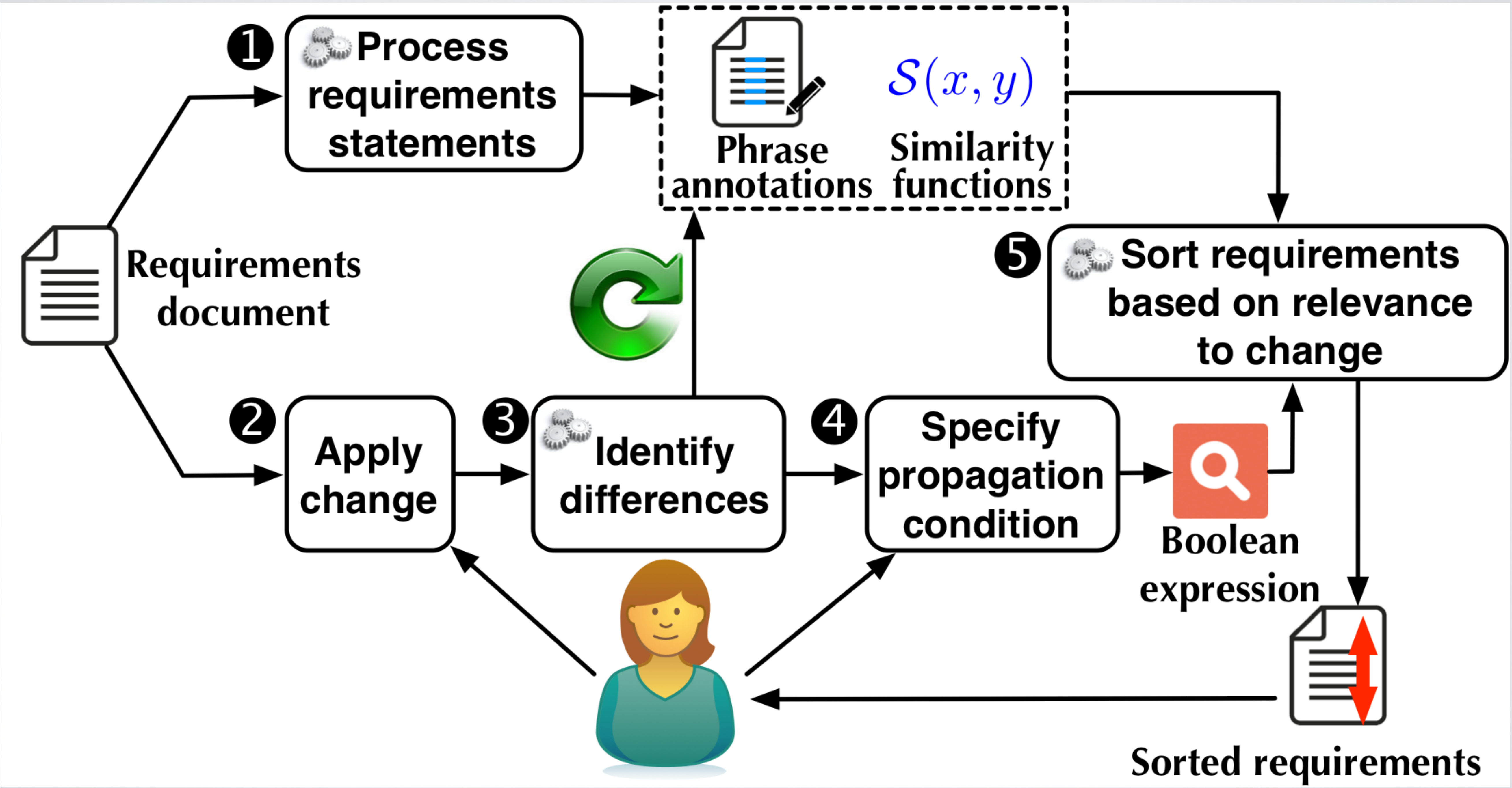
- **Account for the phrasal structure of requirements**

The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.

user help desk, Deleted
user document repository, Added

- **Account for change rationale expressed by user (propagation condition)**
- **Consider semantically-related phrases that are not exact matches and close syntactic variations across requirements: quantitative matching of condition**

Narcia



https://sites.google.com/site/svvnarcia/

Evaluation

SES
your satellite company

1 case study

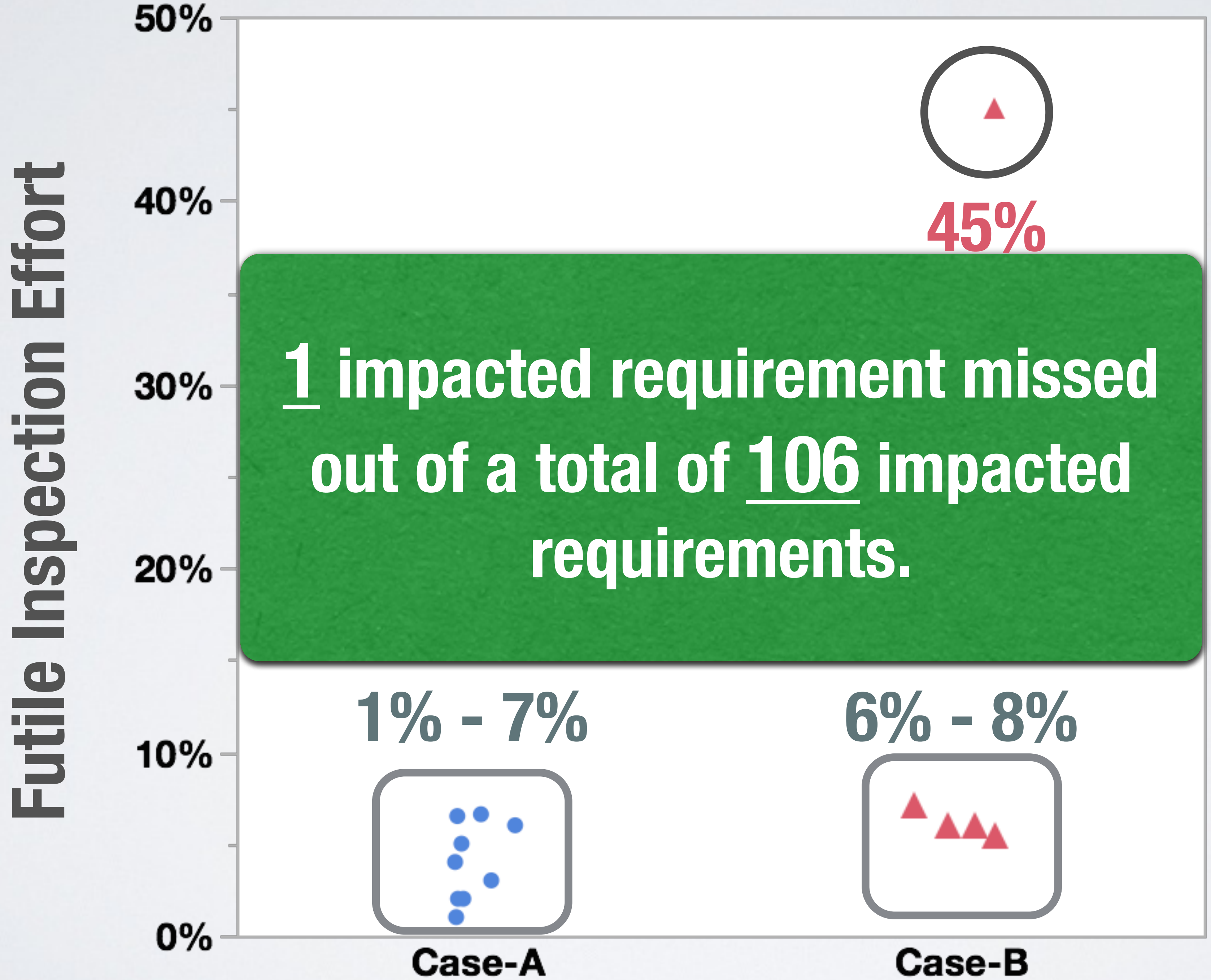
158 Requirements
9 change scenrios

novay

1 case study

72 Requirements
5 Change
Scenarios

Effectiveness of Our Approach



1 impacted requirement missed out of a total of 106 impacted requirements.

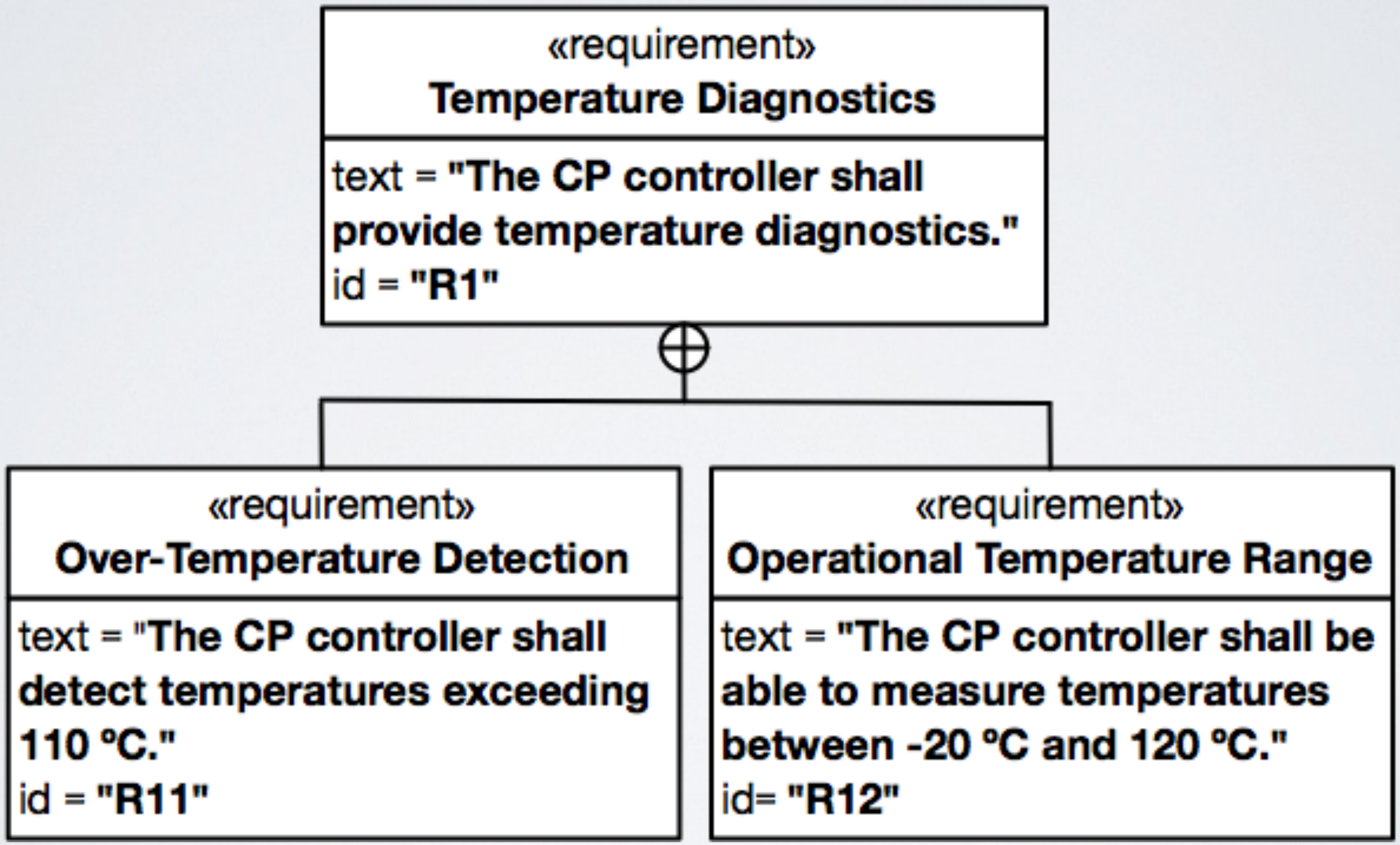
Requirements to Design

Requirements-to-Design
Change Impact Analysis

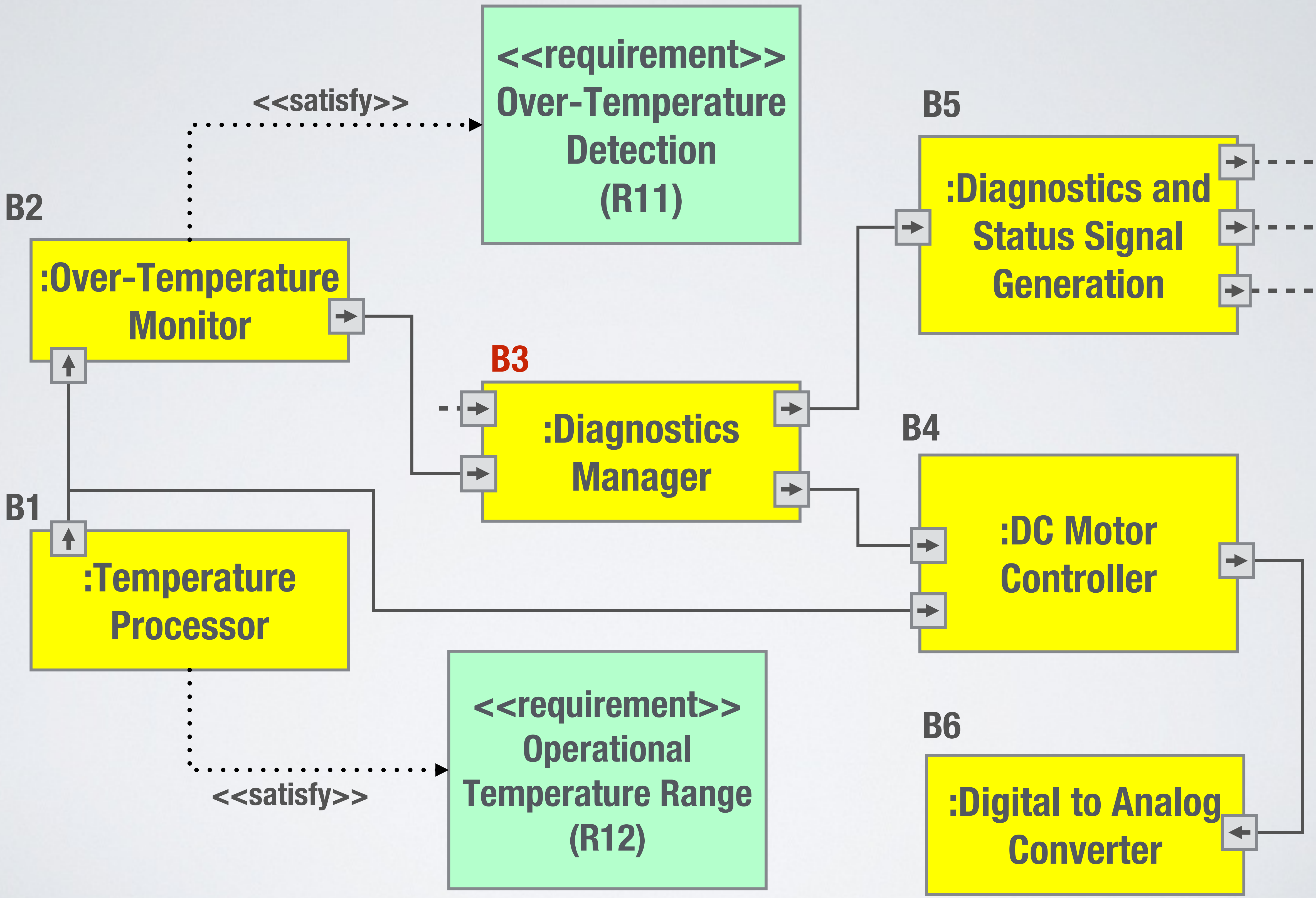
Motivations

- Rigorous change management required by many standards and customers in safety critical systems, and embedded systems in general in many industry sectors
- Impact of requirements changes on design decisions
- Complete and precise design impact set
- SysML commonly used as embedded and cyber-physical system design representation

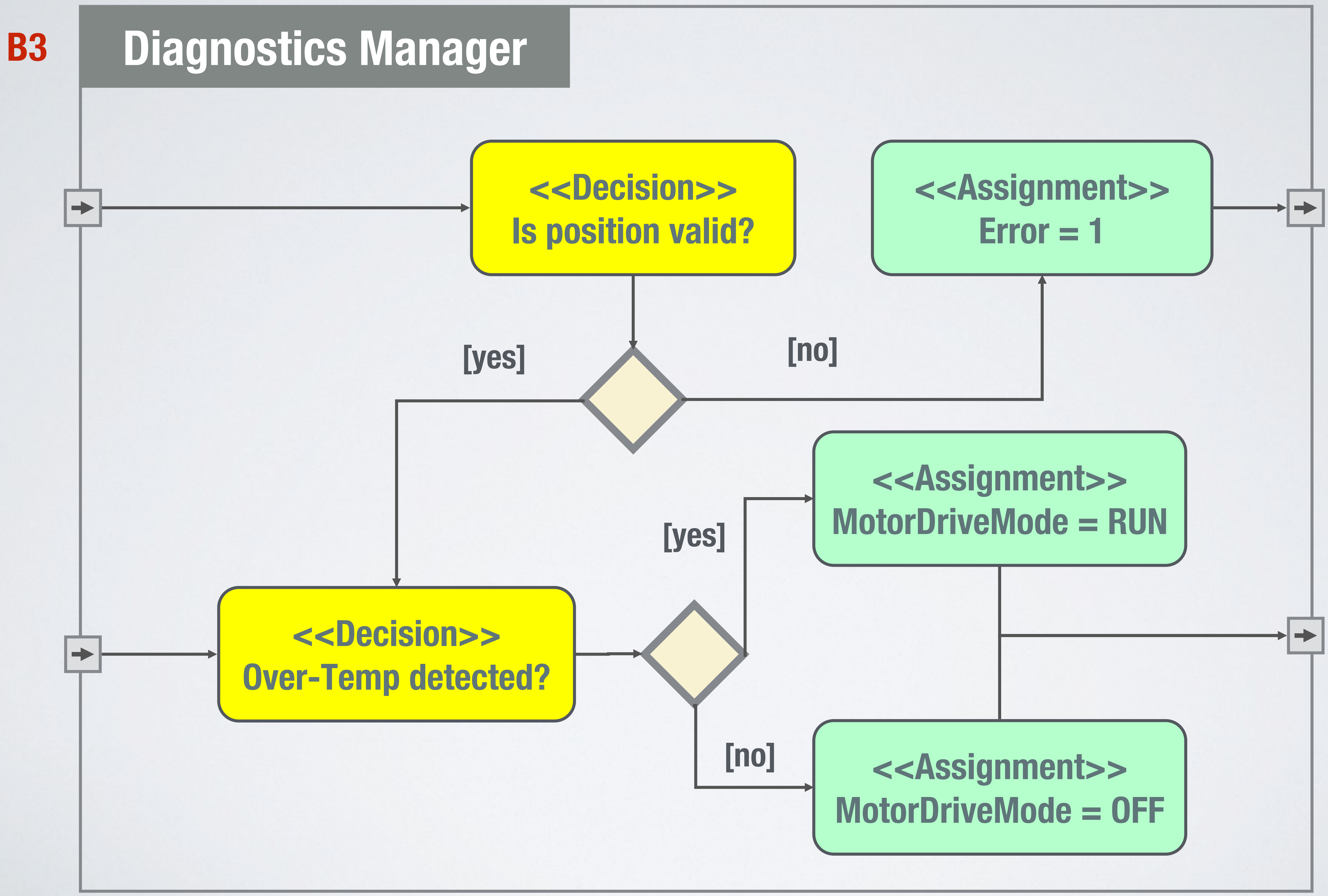
Requirements Diagram



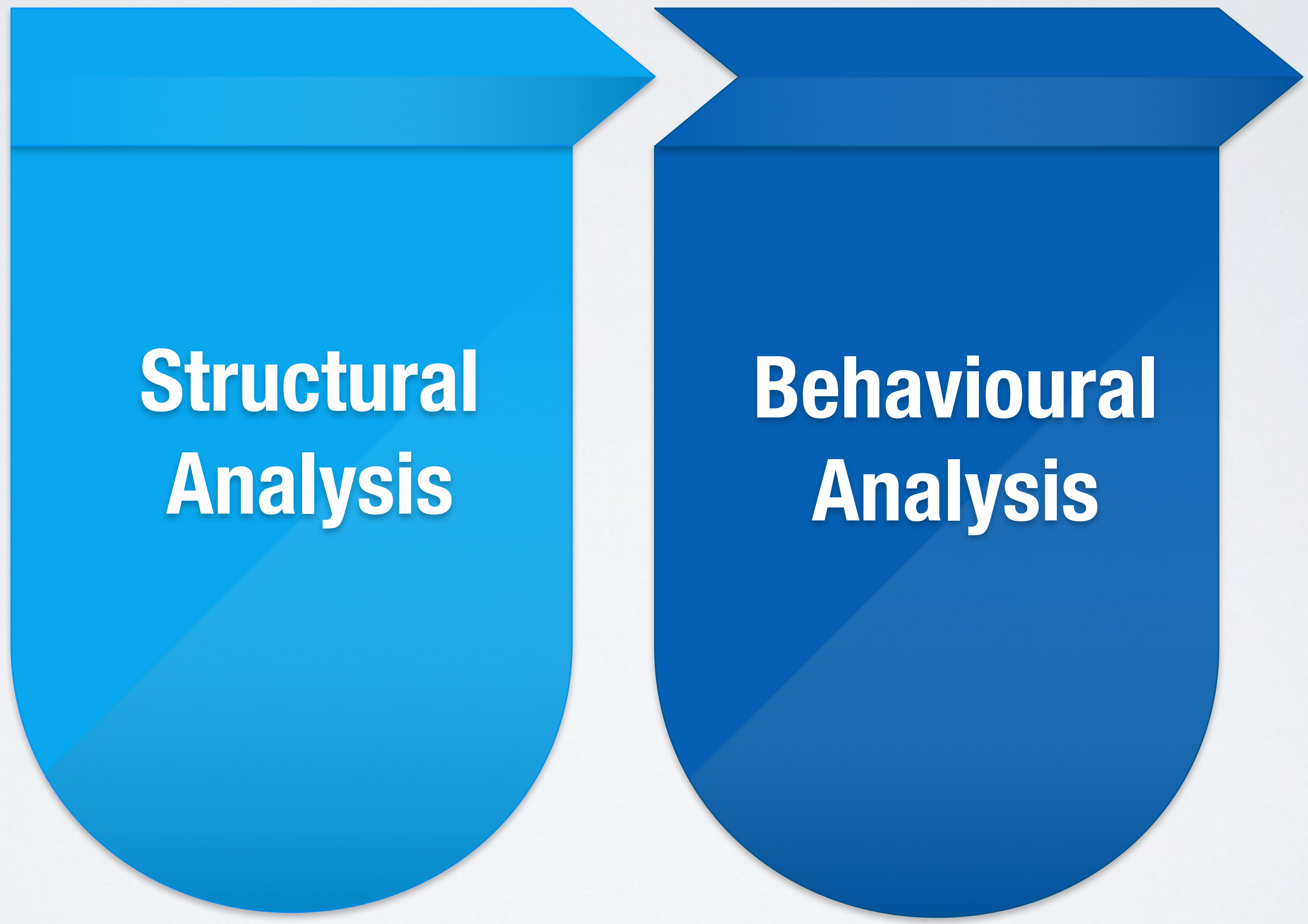
Structural Diagram



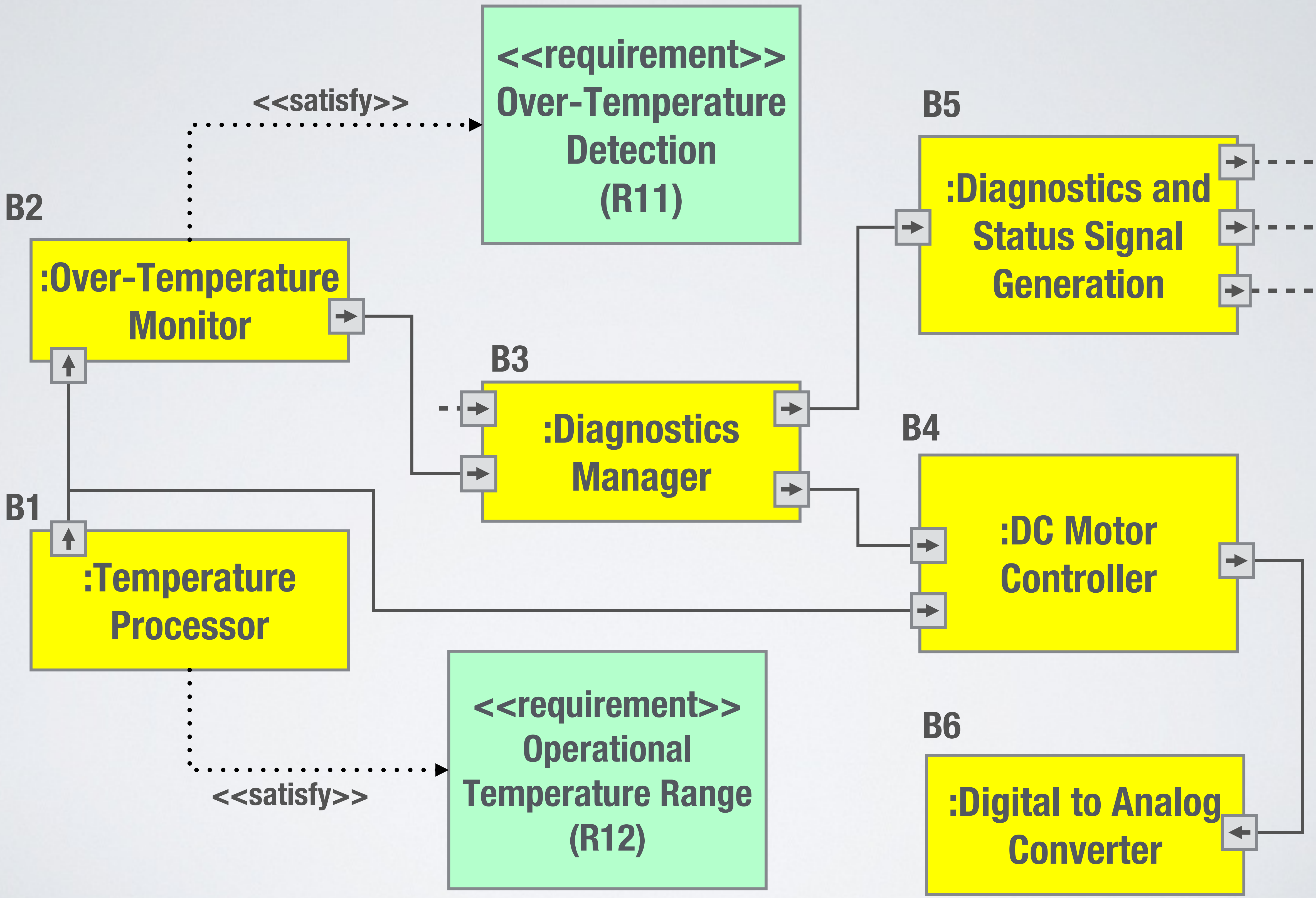
Behavioural Diagram



Compute Impacted Elements



Structural Diagram

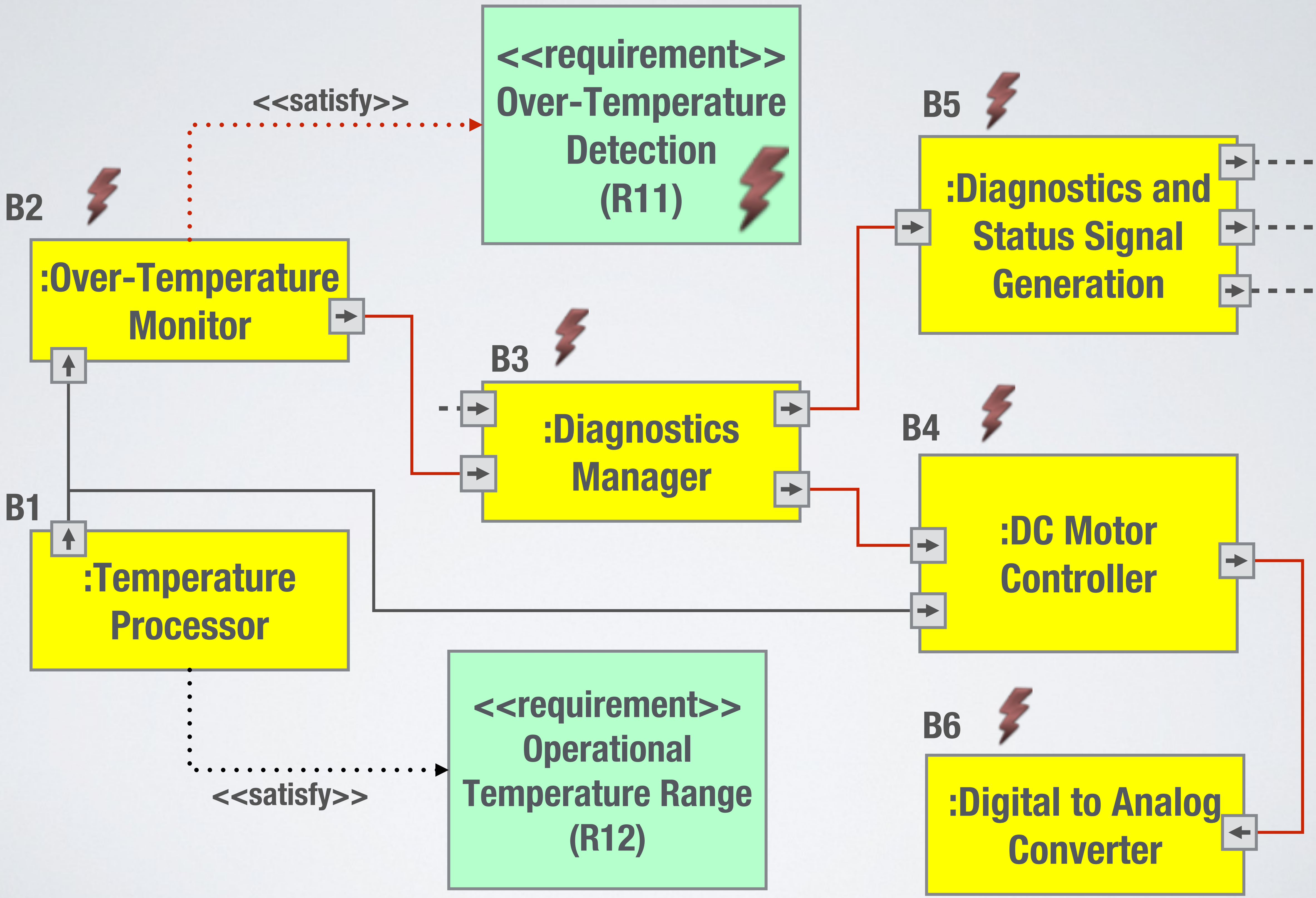


Structural Diagram

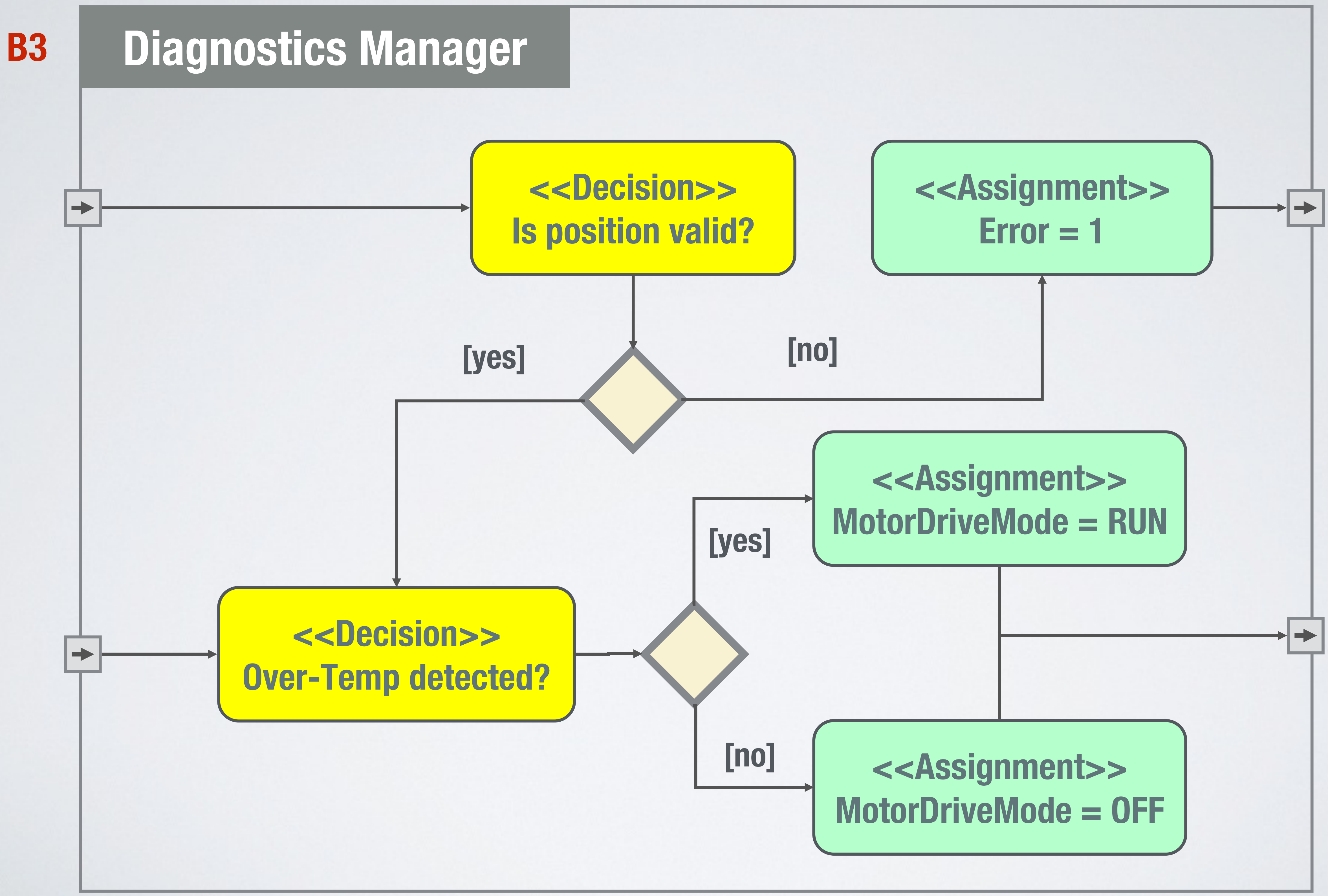
<<requirement>>

Change to R11: Change over temperature detection level to 147 C from 110 C.

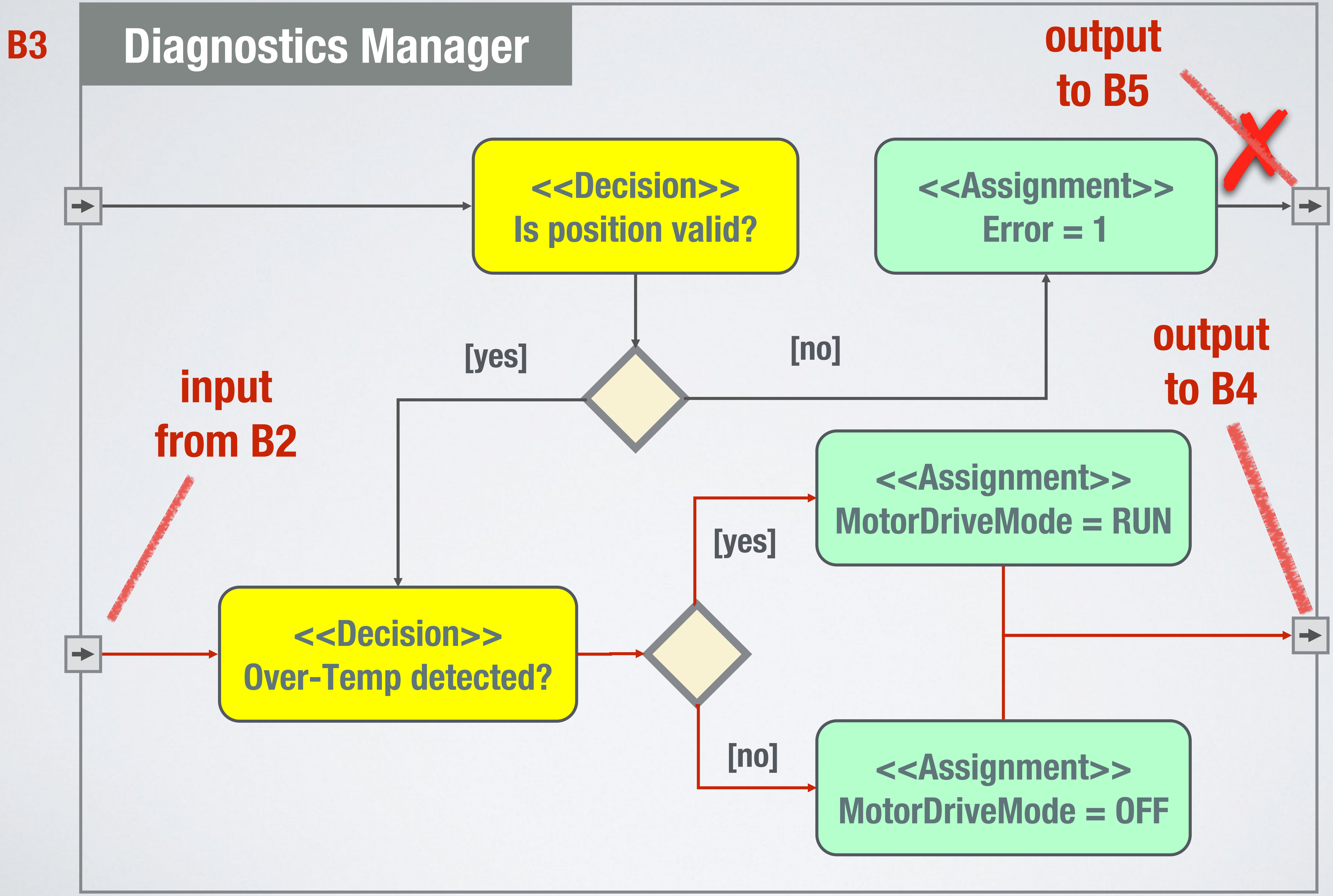
Structural Diagram



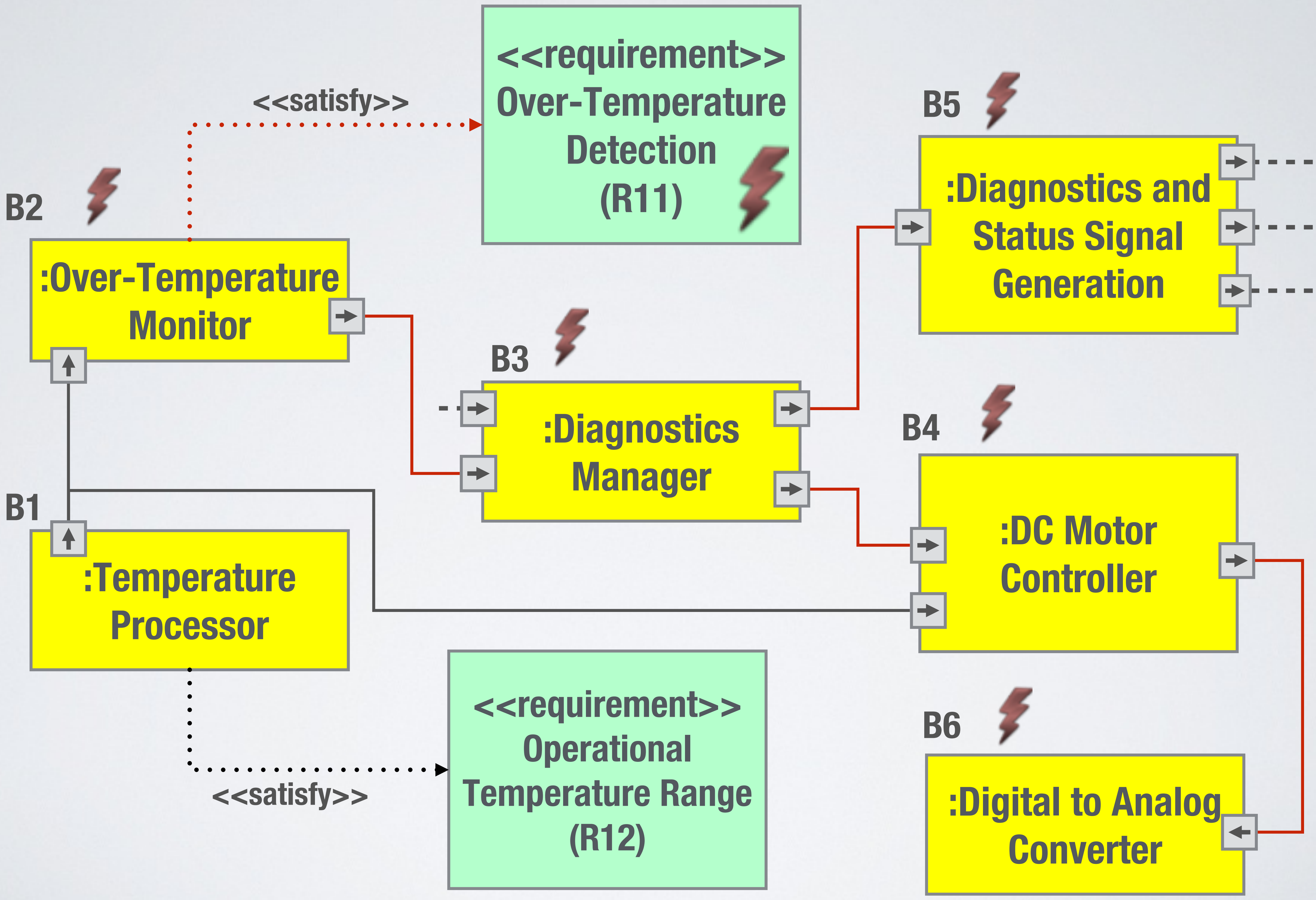
Behavioural Diagram



Behavioural Diagram



Structural Diagram



Rank Elements

Change to R11: Change over temperature detection level to 147 C from 110 C.

B2, B3, B4, B6

Natural Language Processing Analysis

B2
B6
B3
B4

Ranked according to likelihood of impact

Change Statements

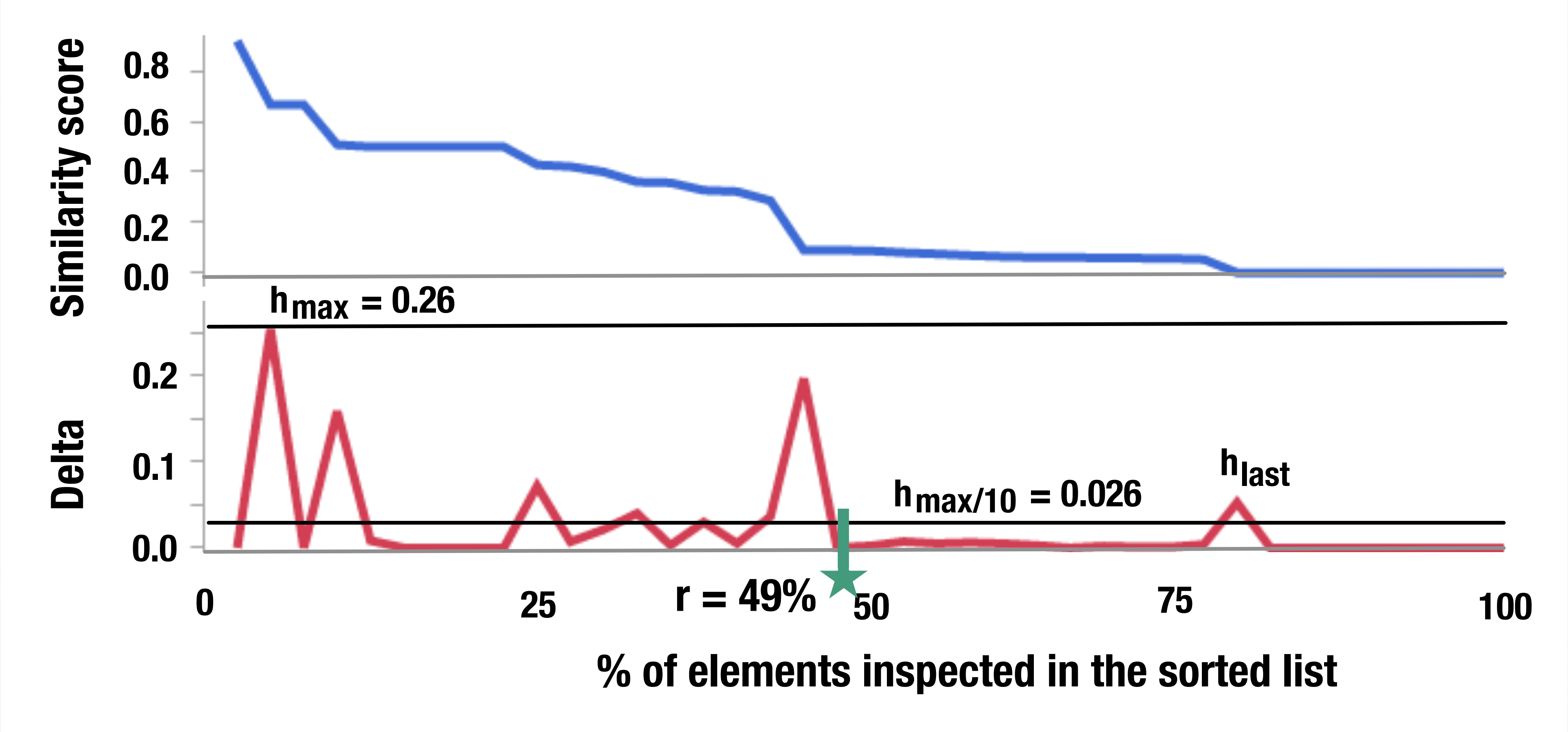
- **Informal inputs** from systems engineers regarding impact of changes
- **Example:** "Temperature lookup tables and voltage converters need to be adjusted"

Natural Language Processing

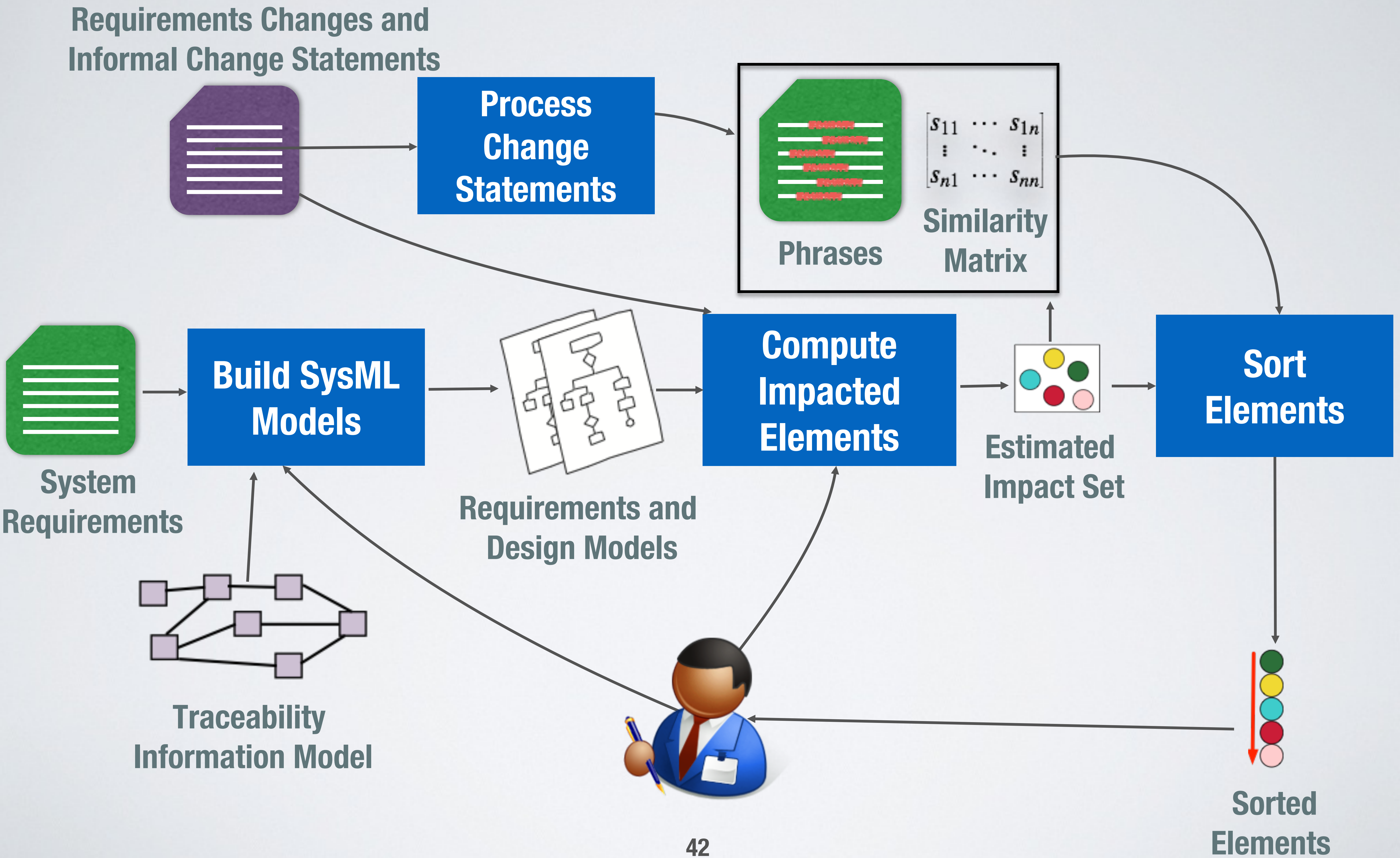
- Compute **similarity scores** (syntactic and semantic) **between model elements labels and change statements**
- Experimented with **pairwise combinations** of syntactic and semantic measures
- **Sort** the design elements obtained after structural and behavioral analysis based on the similarity scores
- Engineers inspect the **sorted lists** to identify impacted elements

Identifying a Subset to Inspect

- Pick the last significant peak in delta similarity between two successive elements
- Point beyond which the similarity scores can no longer adequately tell apart the elements



Overview

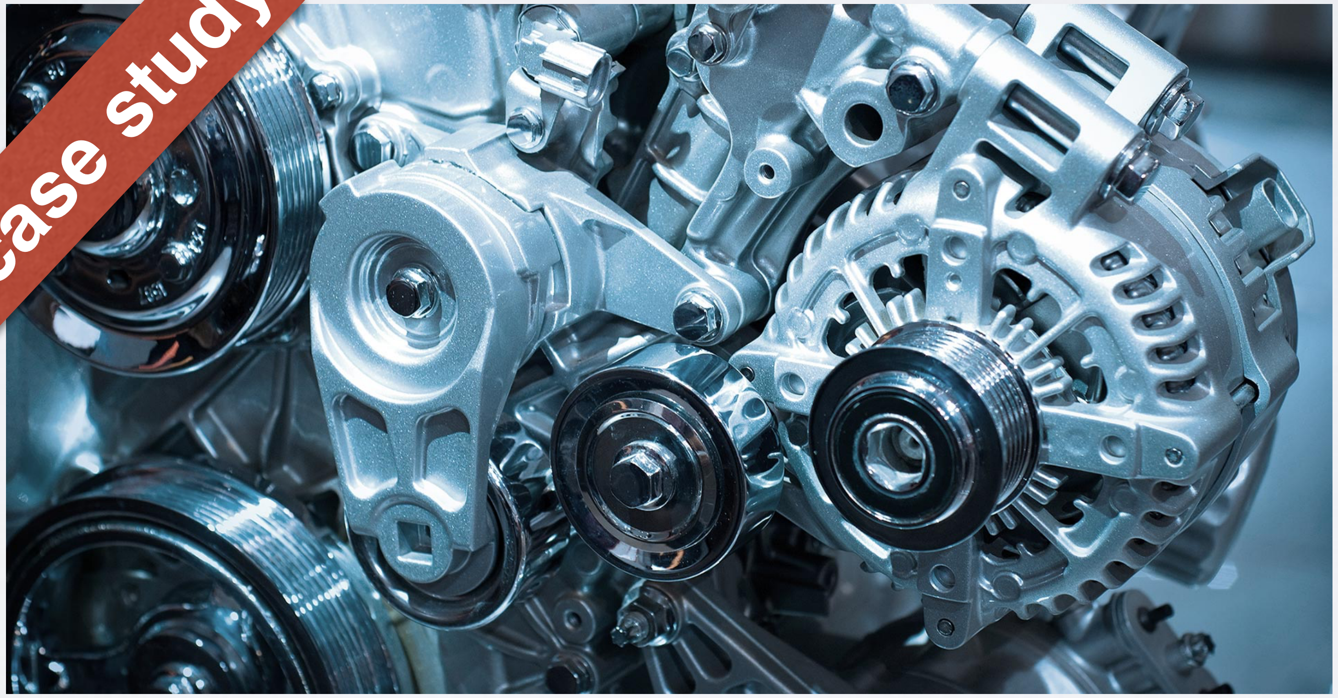

Requirements Changes and Informal Change Statements
Process Change Statements
Phrases
$\begin{bmatrix} s_{11} & \cdots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{n1} & \cdots & s_{nn} \end{bmatrix}$
Similarity Matrix
System Requirements
Build SysML Models
Requirements and Design Models
Compute Impacted Elements
Estimated Impact Set
Sort Elements
Traceability Information Model
Sorted Elements

Evaluation

DELPHI

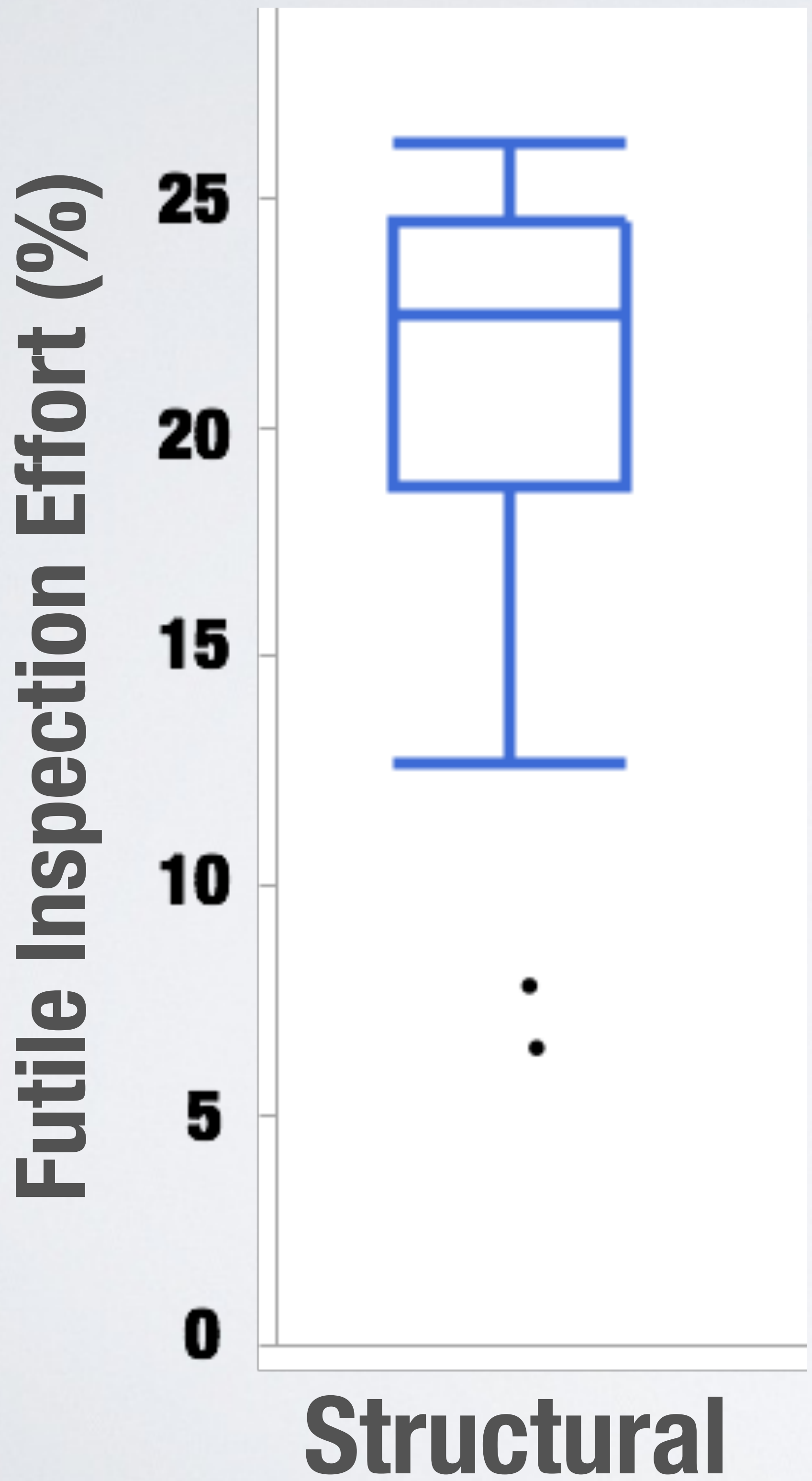
Innovation for the Real World

1 case study

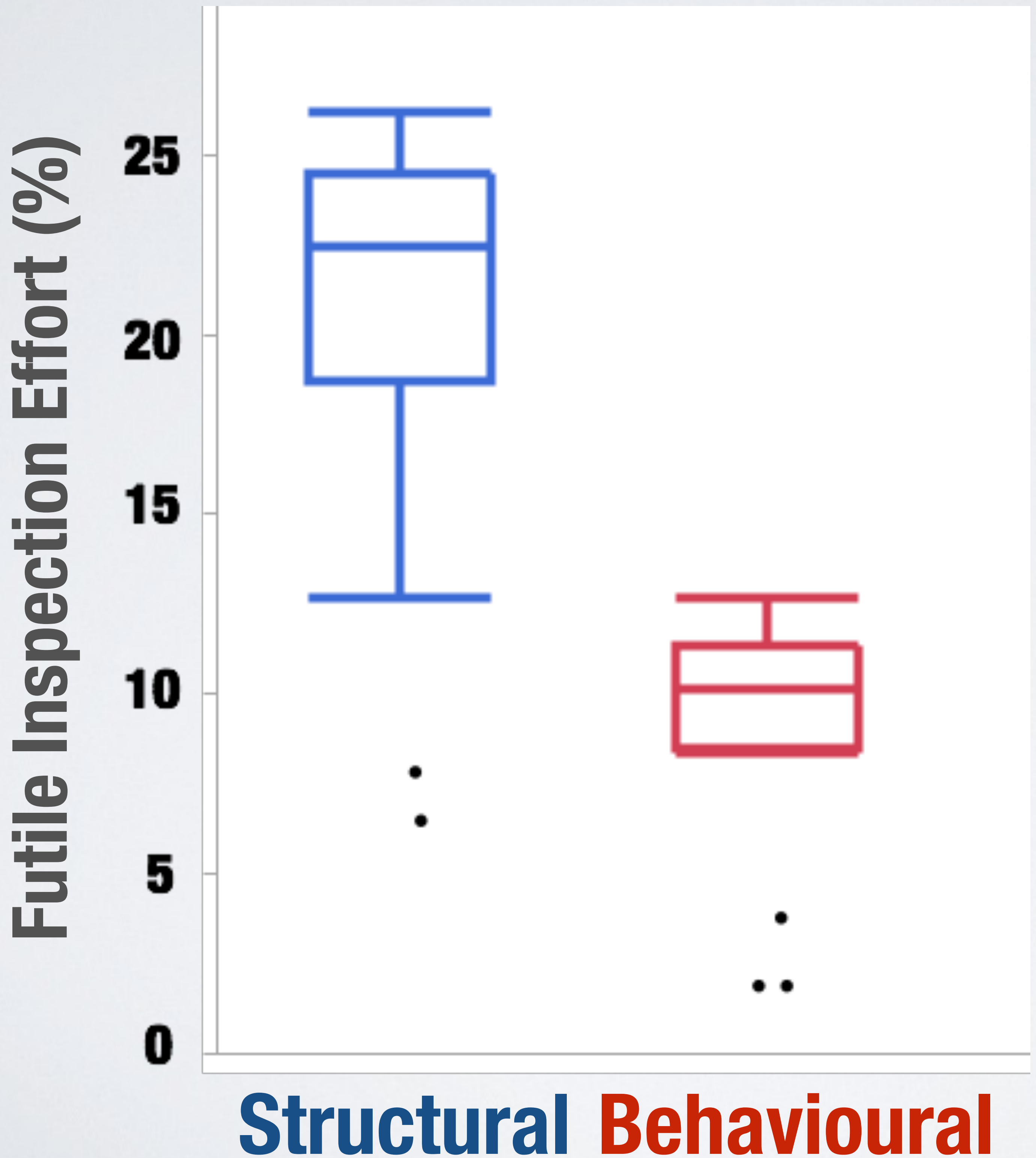


370 elements
16 change scenarios

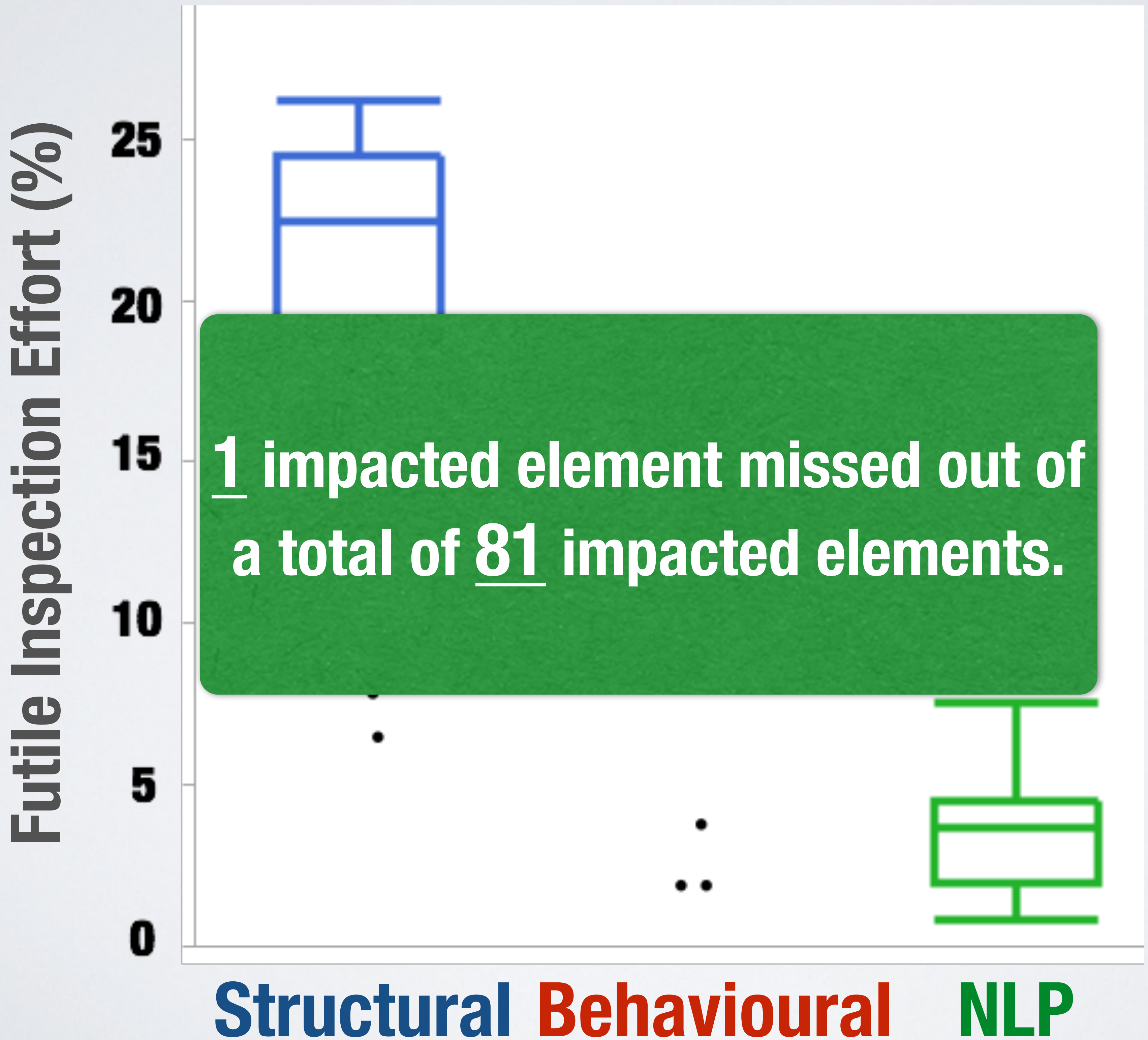
Effectiveness of Our Approach



Effectiveness of Our Approach



Effectiveness of Our Approach



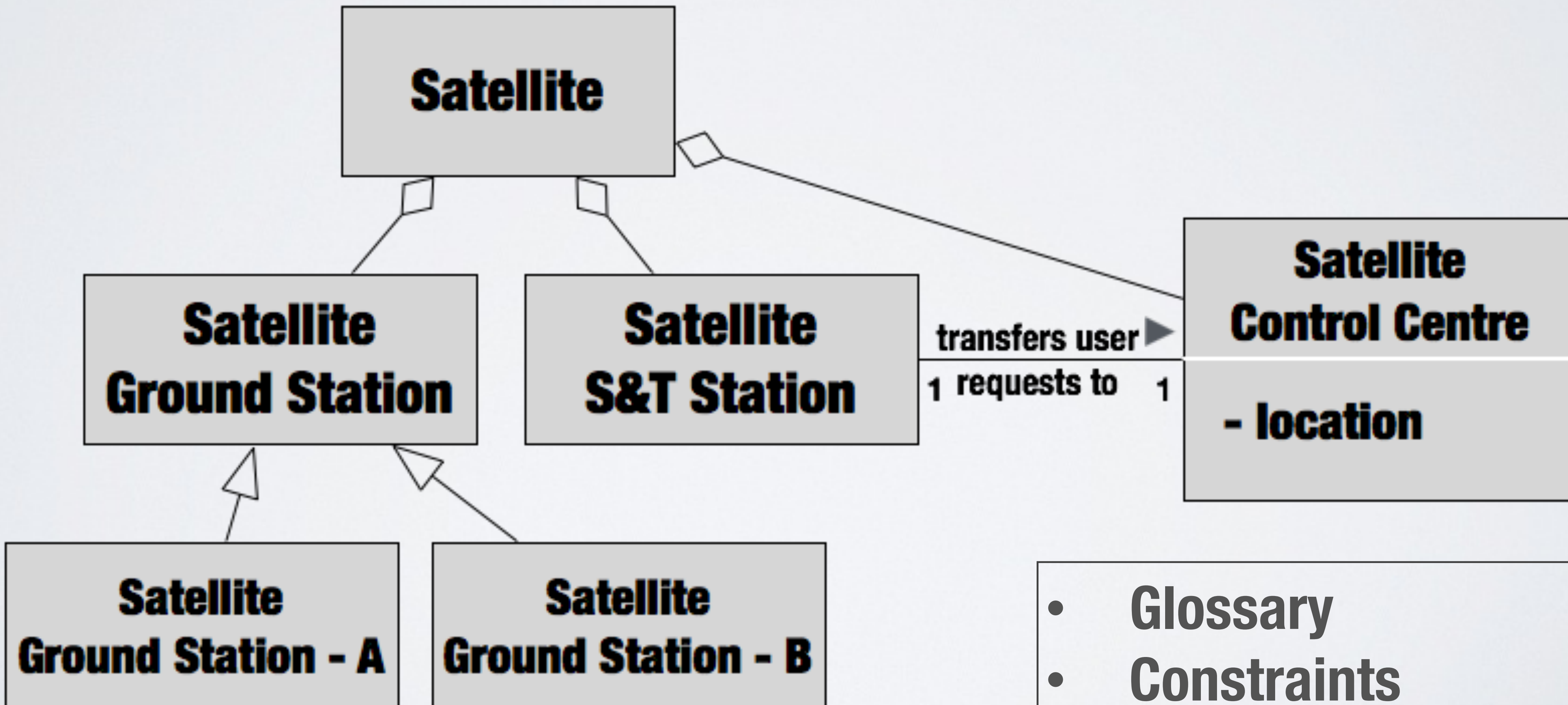
Extracting Domain Knowledge

Domain Knowledge

- All requirements depend, more or less explicitly, on domain knowledge
- Domain-specific concepts and terminology
- In practice: Not always consistent among all stakeholders
- Software engineers often have a superficial understanding of the application domain they target
- Extracting domain knowledge from requirements: Glossary, domain model …

Domain Models

A domain model is a representation of conceptual entities or real-world objects in a domain of interest

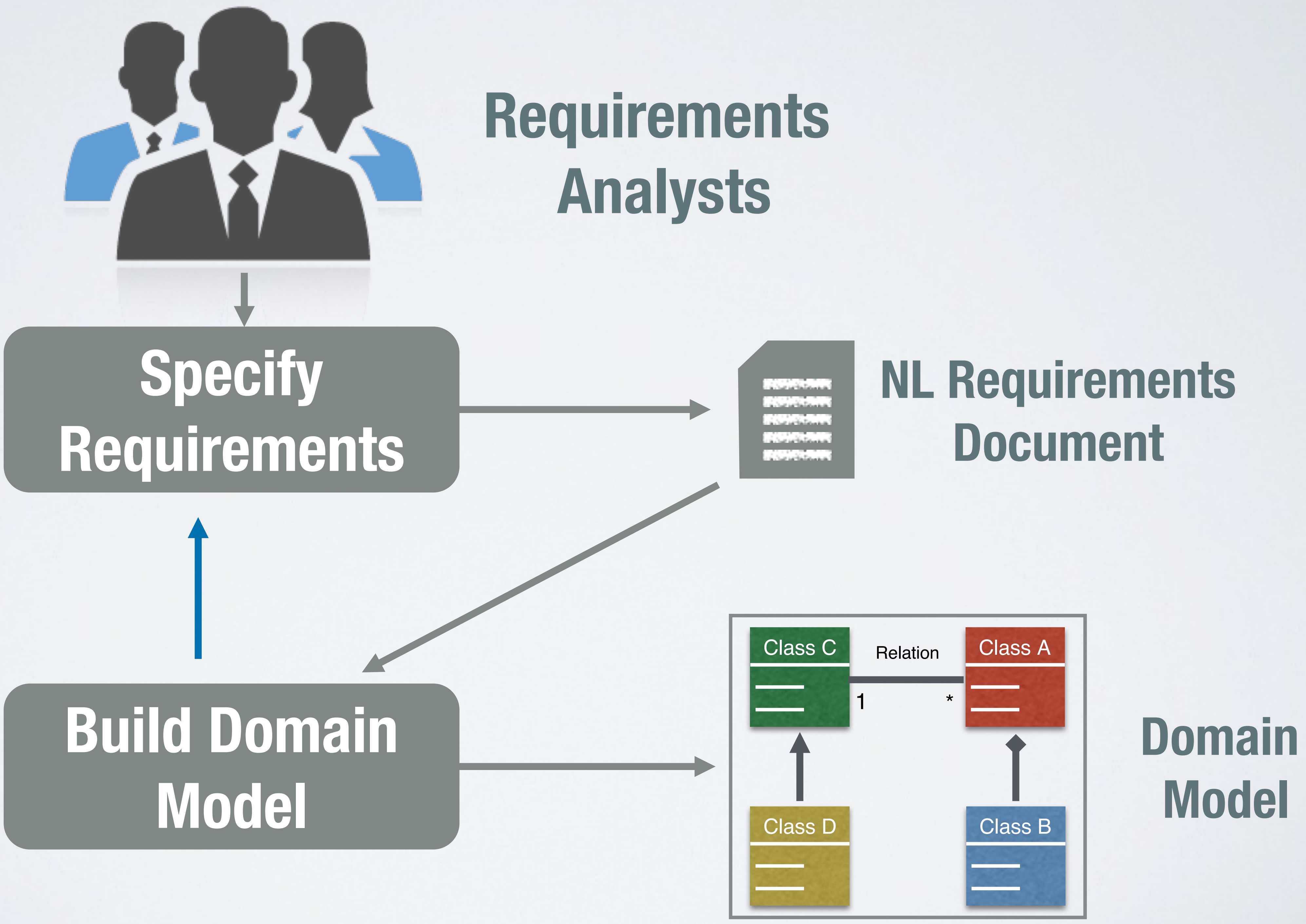


- Glossary
- Constraints

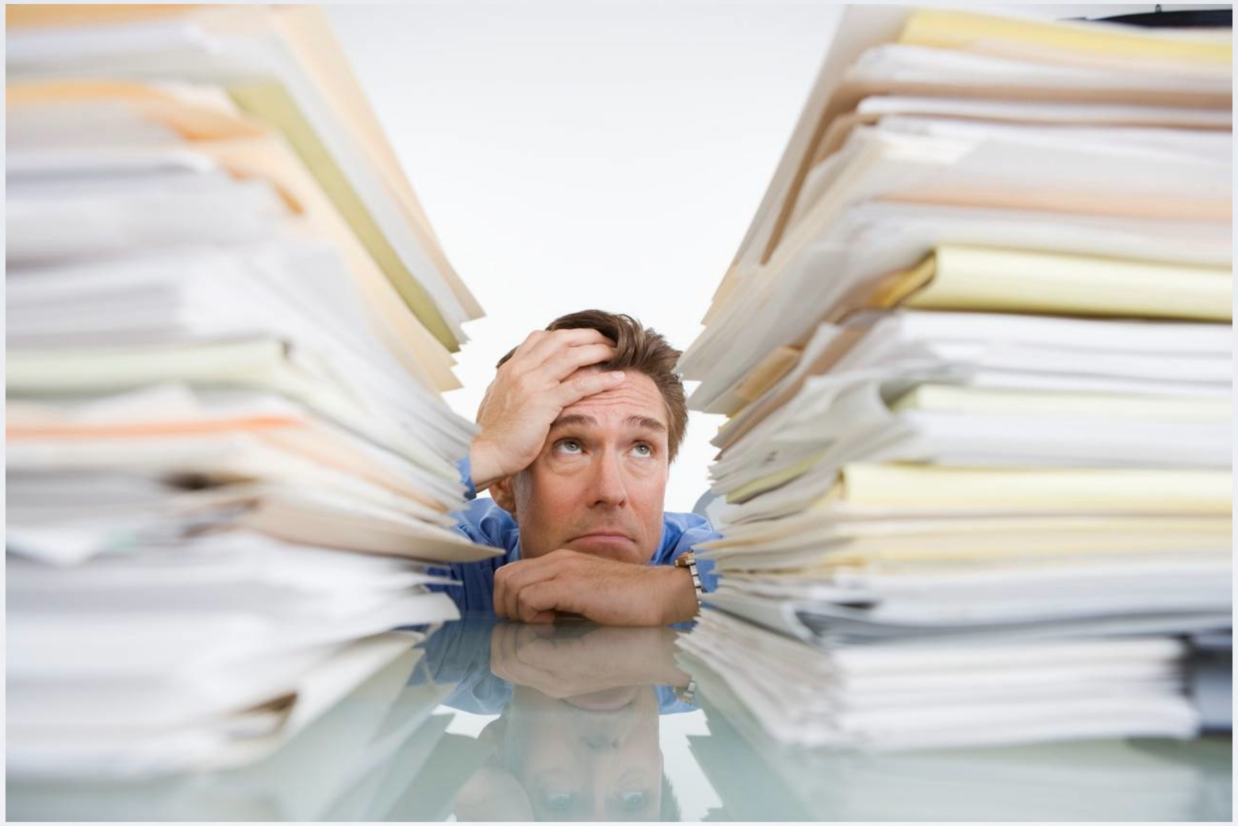
Motivation

- Representation of important domain concepts and their relations
  - Facilitate communication between stakeholders from different backgrounds
  - Help identify inconsistencies in terminology, etc.
  - Helps assess completeness of requirements
- In practice, domain models are not preceding the elicitation and writing of requirements

Context


Requirements Analysts
Specify Requirements
NL Requirements Document
Build Domain Model
Class C
Relation
Class A
1
*
Class D
Class B
Domain Model

Problem Definition

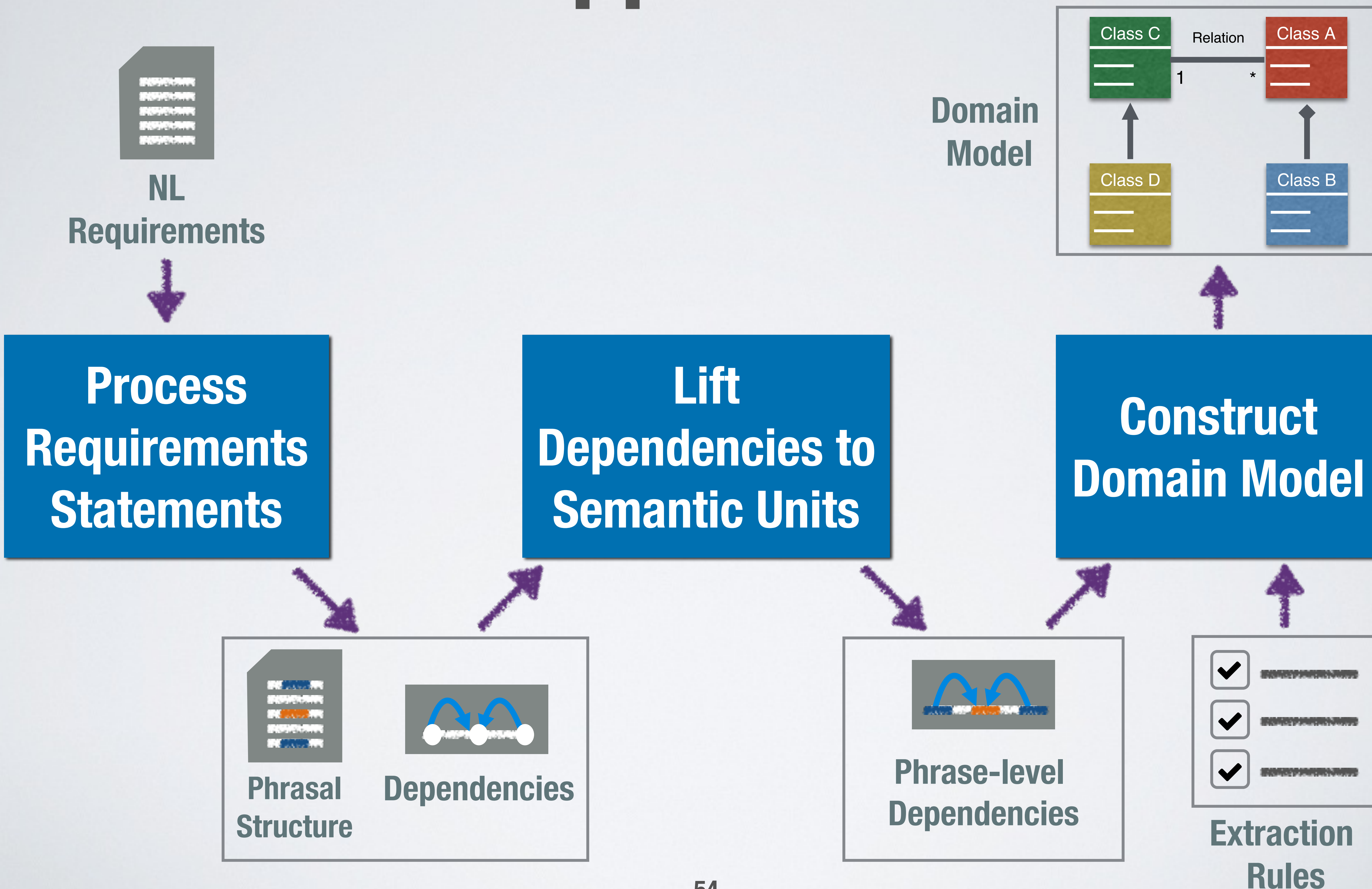


- Manually building domain models is **laborious**
- **Automated support is required** for building domain models

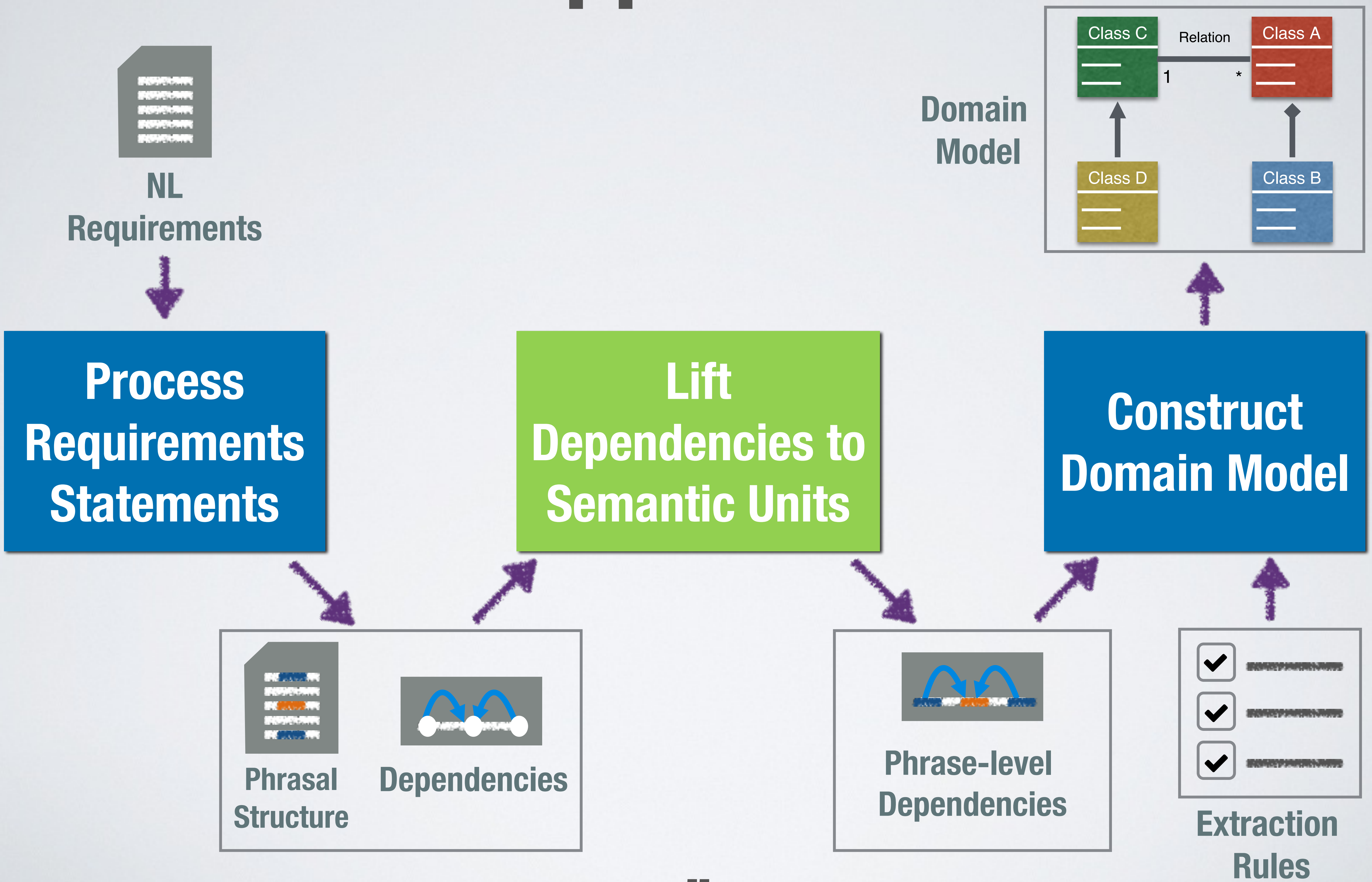
State of the Art

- **Multiple approaches exist** for extracting domain models or similar variants from requirements using **extraction rules**
  - Majority assume **specific structure**, e.g., restricted NL
  - Extraction of **direct relations only** but not indirect ones
  - **Limited empirical results** on industrial requirements

Approach

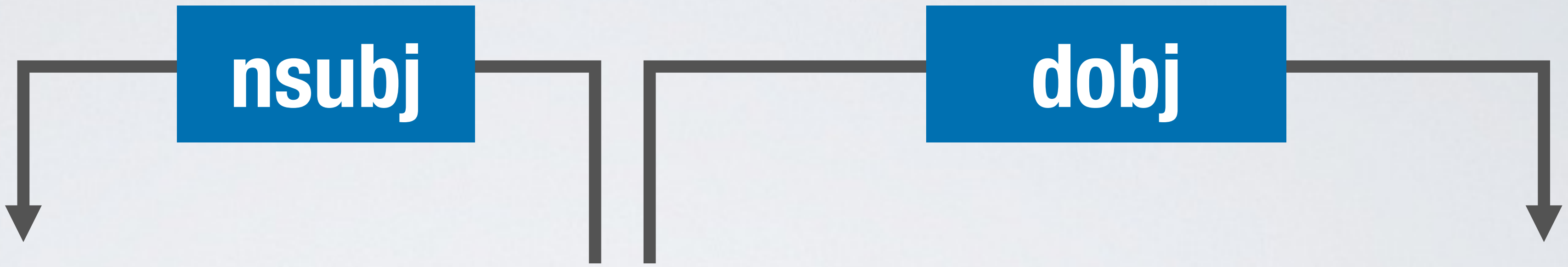

NL Requirements
Process Requirements Statements
Phrasal Structure
Dependencies
Lift Dependencies to Semantic Units
Phrase-level Dependencies
Construct Domain Model
Extraction Rules
Domain Model
Class C
Relation
Class A
1
*
Class D
Class B

Approach

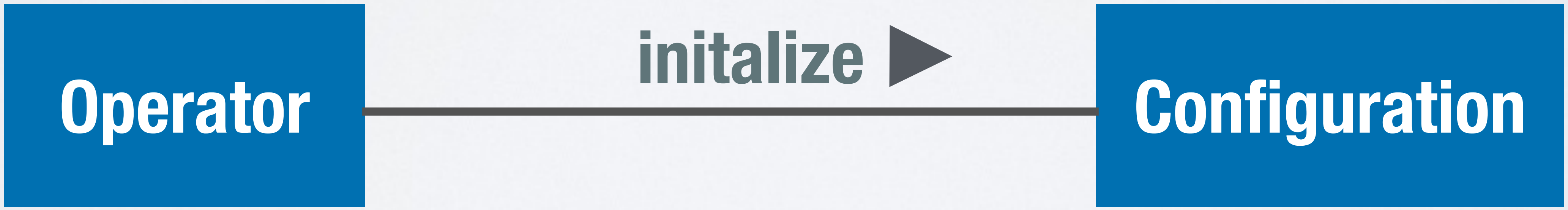


NL
Requirements
Process Requirements Statements
Phrasal Structure
Dependencies
Lift Dependencies to Semantic Units
Phrase-level Dependencies
Construct Domain Model
Extraction Rules
Domain Model
Class C
Relation
Class A
1
*
Class D
Class B

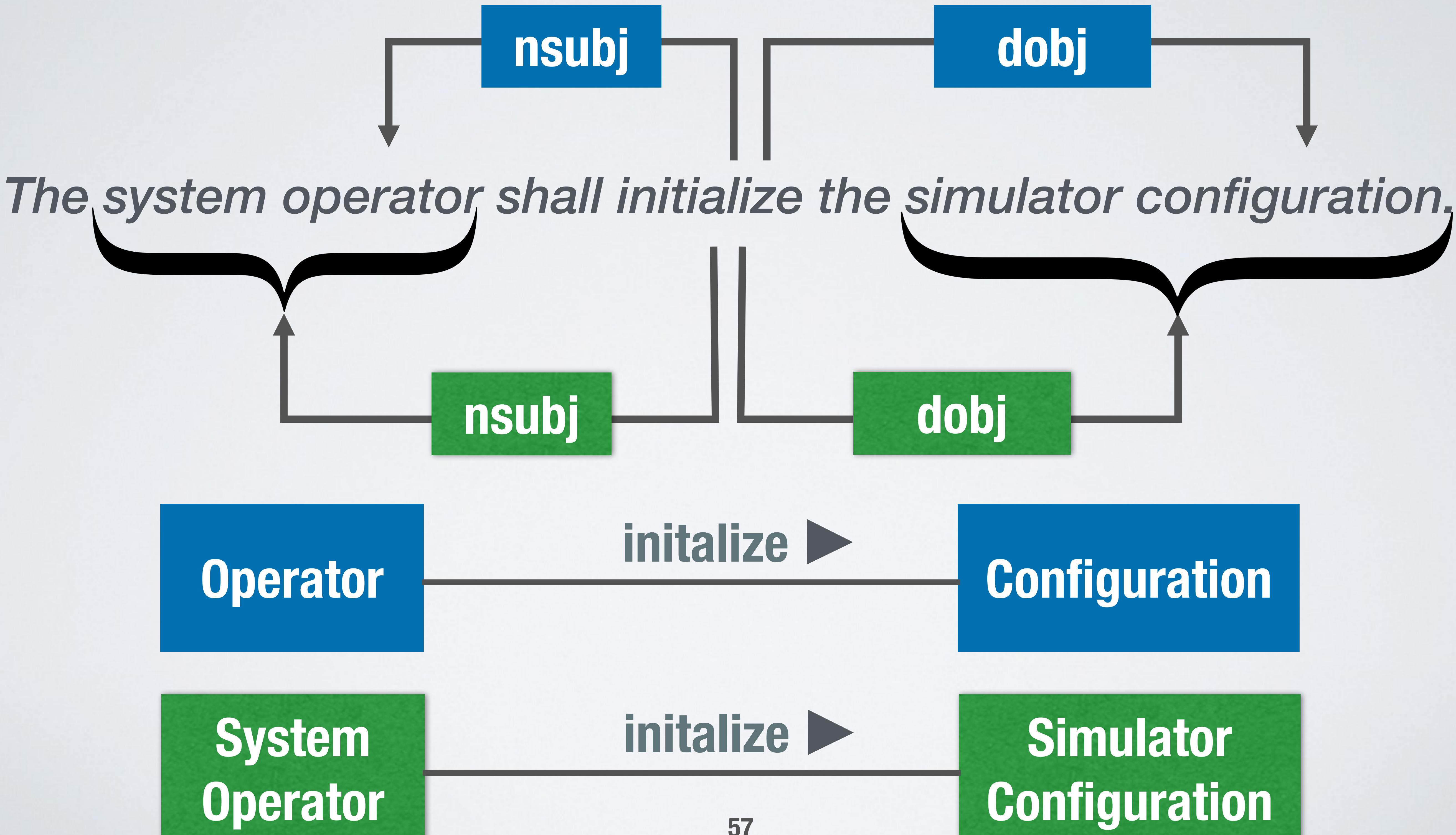
Grammatical Dependencies



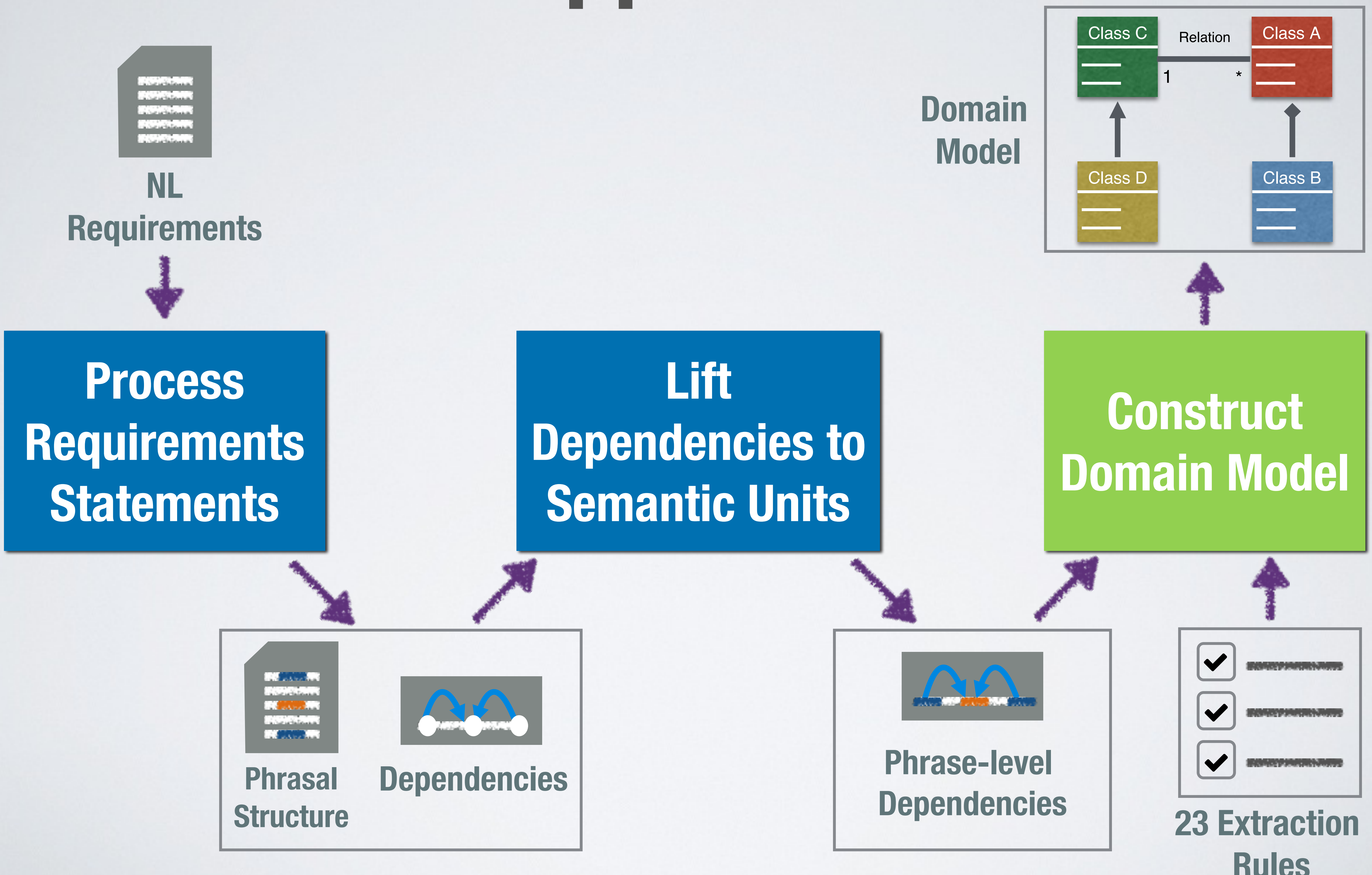
The system operator shall initialize the simulator configuration.



Lift Dependencies to Semantic Units

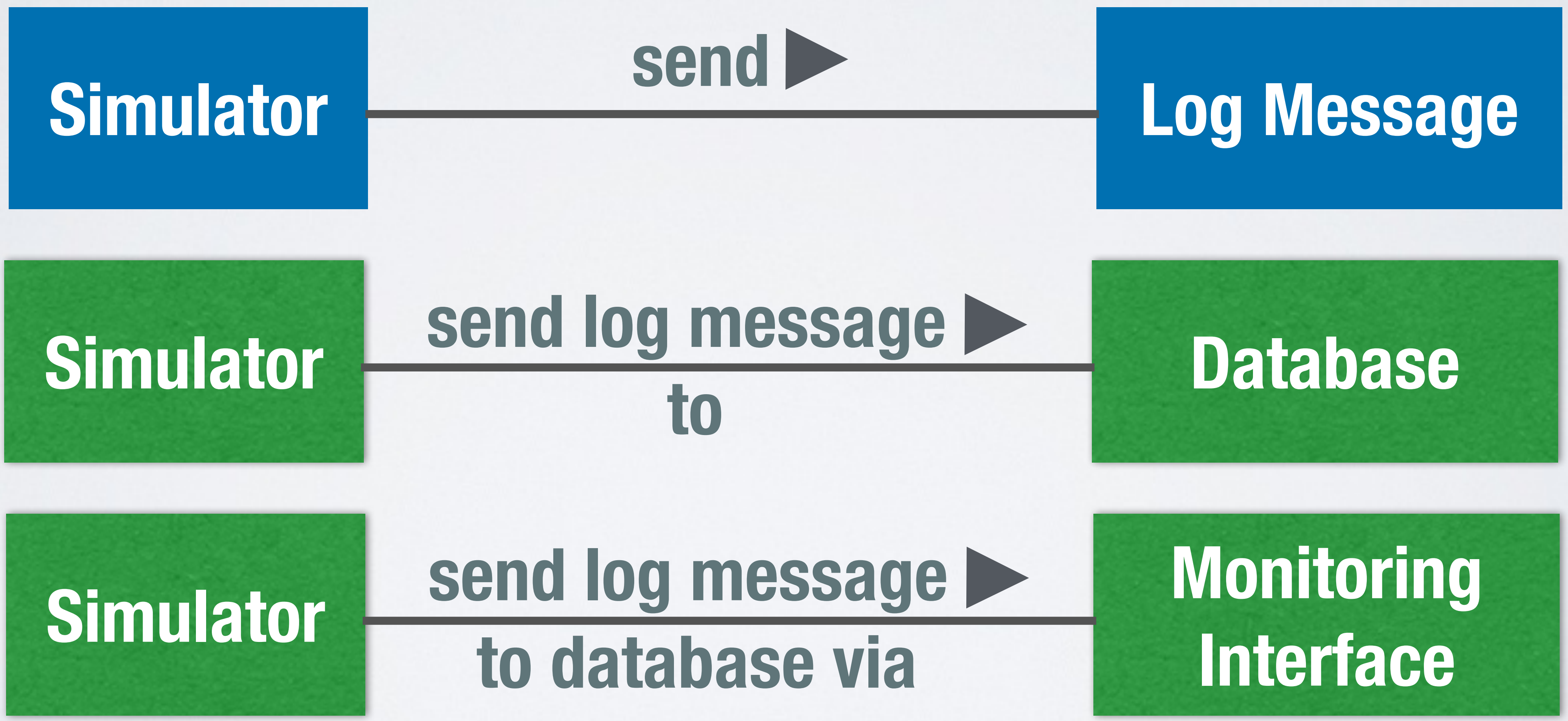


Approach


NL Requirements
Process Requirements Statements
Lift Dependencies to Semantic Units
Construct Domain Model
Domain Model
Class C
Relation
Class A
1
*
Class D
Class B
Phrasal Structure
Dependencies
Phrase-level Dependencies
23 Extraction Rules

Link Paths – Indirect Relations

The simulator shall send log messages to the database via the monitoring interface.



How useful is our approach?

SES your satellite company

1 case study

50 Requirements
213 Relations

- **Interview survey** with experts
- **Correctness and Relevance** of each relation
- **Missing relations** in each requirement

Results

Correctness: 88% (avg.)

Incorrectness largely explained by NLP errors

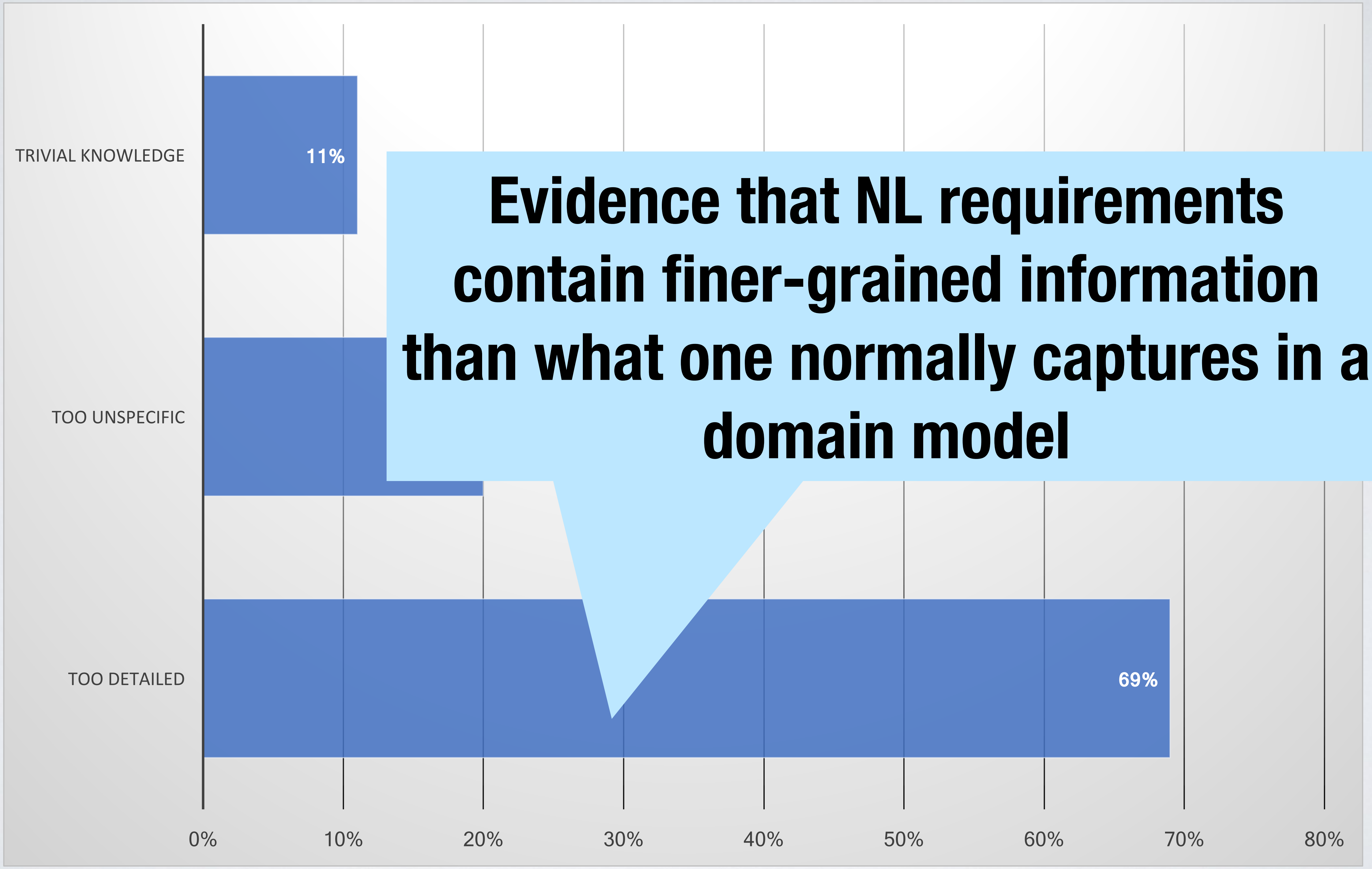
Relevance: 37% (avg.)

Breakdown of the remaining 63%

12% are incorrect
51% are correct but superfluous

Missed: 10% (avg.)

Statistics for Superfluousness



Evidence that NL requirements contain finer-grained information than what one normally captures in a domain model

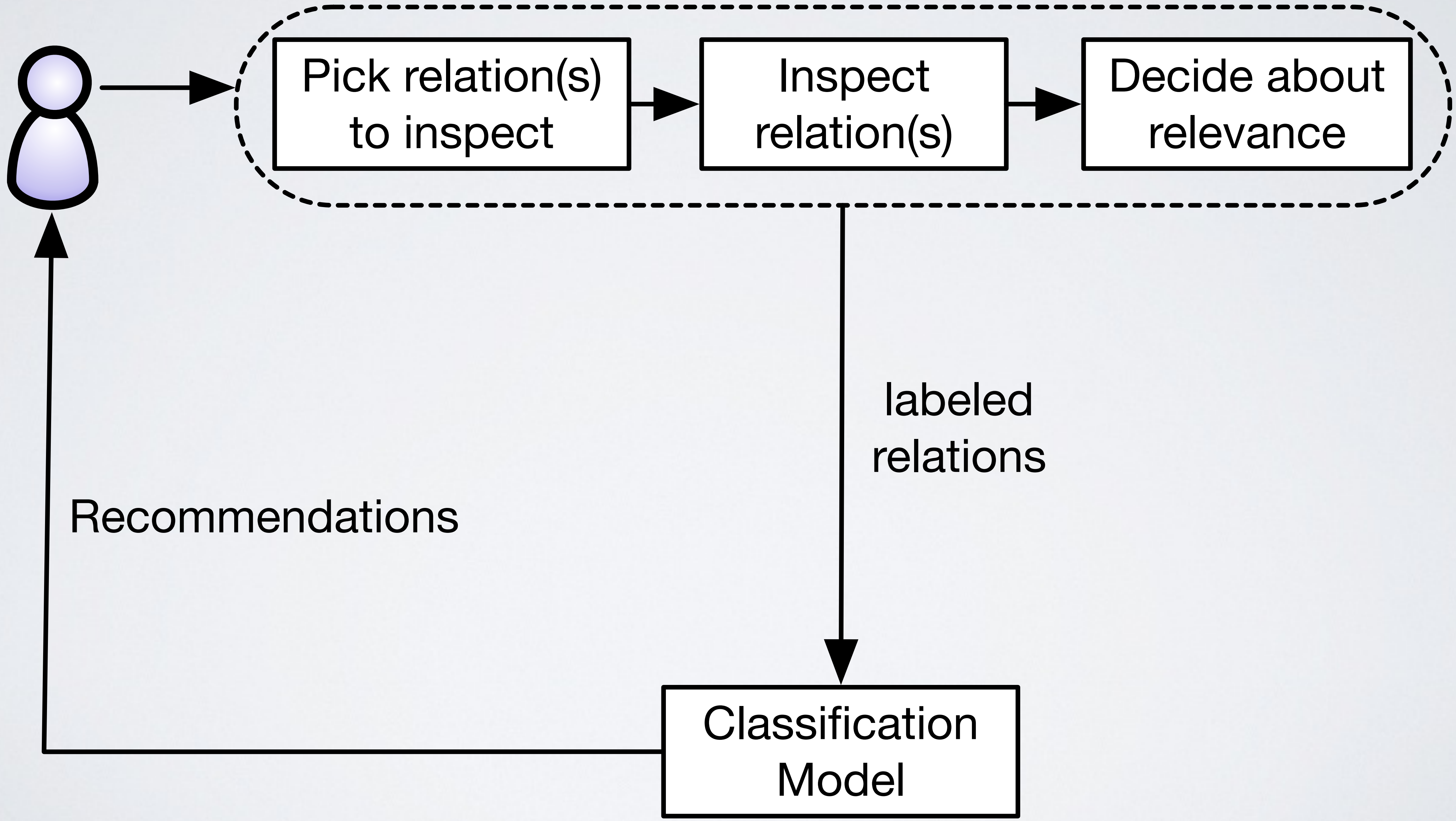
Can we improve the relevance of model extraction results?

Active Learning

- **Definition:** Machine learning paradigm in which a learning technique interactively requests inputs from an external source in order to improve the accuracy of the machine learning model.

- **Application:** We process analysts' feedback, and dynamically apply the logic gleaned from the feedback for reducing superfluous information.

Active Learning Feedback Loop



Example Features (1/2)

Label-independent (Never updated):

- Type of the relation: Association, Aggregation, Generalization
- The extraction rule that produced the relation
- Number of tokens in the relation's end points (concepts)

Example Features (2/2)

Label-dependent (Updated with new relations):

- Number of relevant relations (in the training set) extracted from the same requirement as the given relation
- Number of relevant relations in the training set that share one end concept with the given relation
- Number of relevant relations in the training set that share both end concept with a given relation

Effectiveness of Detecting Superfluous Relations

- 96% of the recommendations made are correct
  - The approach is unlikely to throw the analyst off-course.
- 45% of the superfluous relations are automatically marked
  - Potentially significant savings
- We do not need a large seed training set: 30-40 relations

Requirements-Driven Testing

Traceability

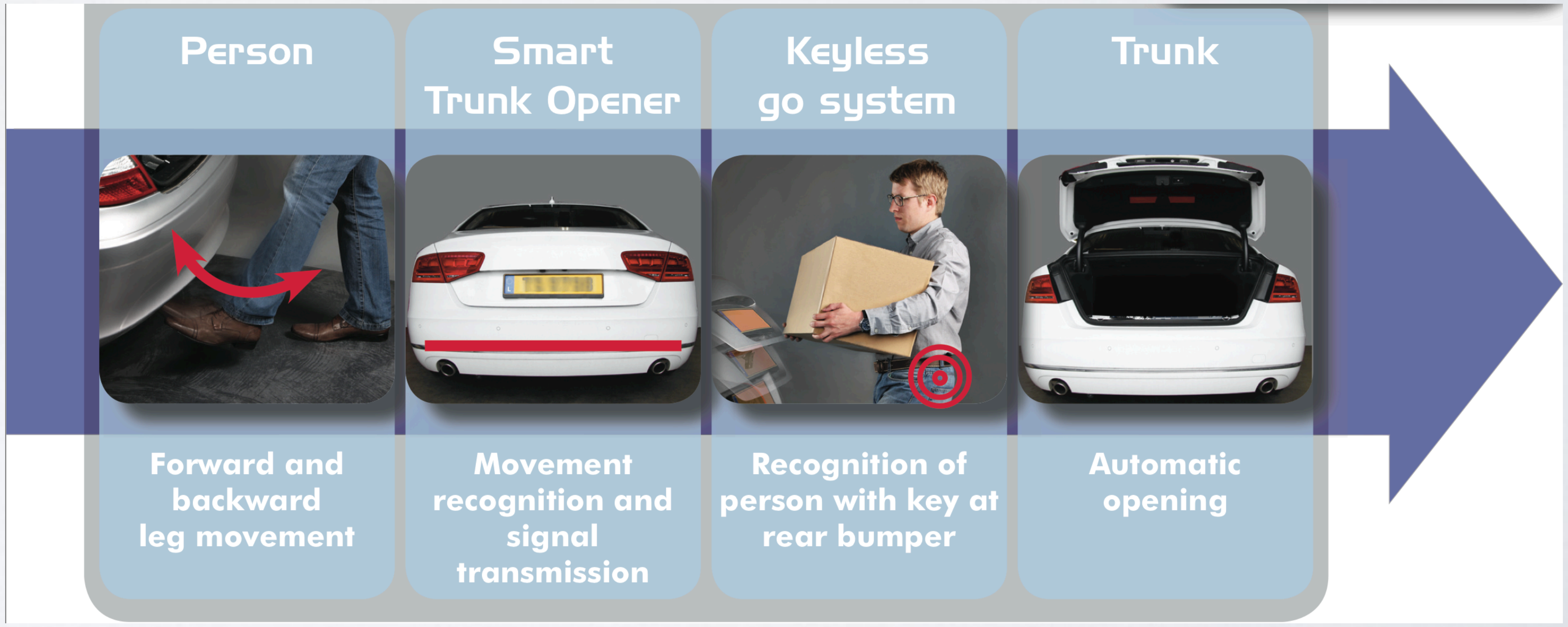
- In many domains, various types of traceability are required
- For example, in automotive (ISO 26262), traceability between requirements and system tests: requirements-driven testing
- Many requirements, many tests, therefore many traces …
- Automation is required

Context

IEE develops real-time embedded systems:

- Automotive **safety sensing systems**
- Automotive **comfort & convenience** systems, e.g., Smart Trunk Opener

International Electronics & Engineering (IEE)



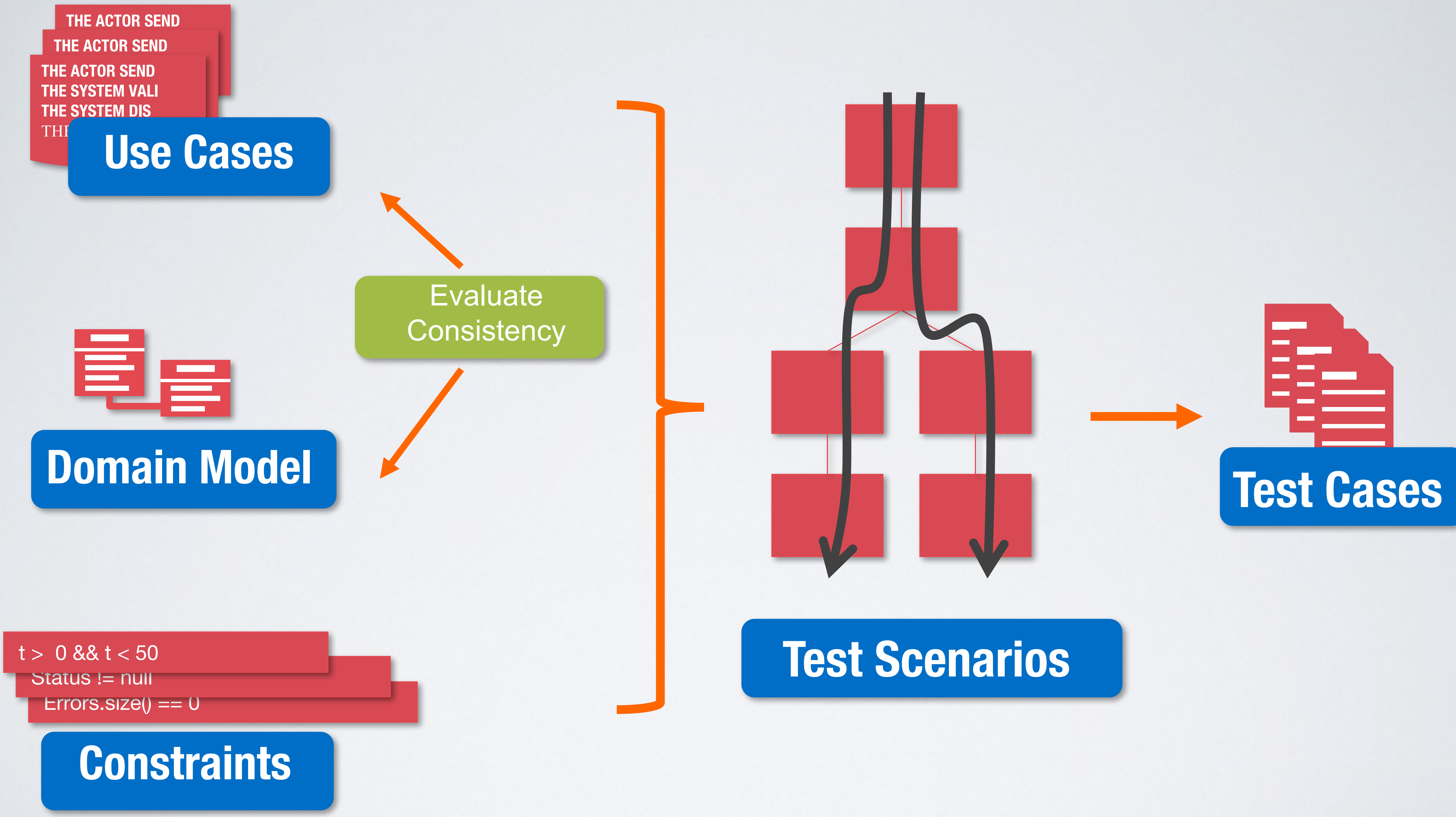
Objectives

- Support generation test cases from requirements
- Capture and create traceability information between test cases and requirements
  - Requirements are captured through use cases
  - Use cases are used to communicate with customers and the system test team
  - Complete and precise behavioral models are not an option: too difficult and expensive (no model-based testing)

Strategy

- **Analyzable** use case specifications
- Automatically extract test model from the use case specifications using **Natural Language Processing**
- **Minimize modeling**, domain modeling only
- **No behavioral modeling**

UMTG



THE ACTOR SEND
THE ACTOR SEND
THE ACTOR SEND
THE SYSTEM VALI
THE SYSTEM DIS
Use Cases
Evaluate Consistency
Domain Model
t > 0 && t < 50
Status != null
Errors.size() == 0
Constraints
Test Scenarios
Test Cases

Restricted Use Case Modeling: RUCM

- RUCM is based on a (1) template, (2) restriction rules, and (3) specific keywords constraining the use of natural language in use case specifications
- RUCM reduces ambiguity and facilitates automated analysis of use cases
- Conformance is supported by a tool based on NLP

RUCM

[Yue et al. TOSEM'13]

Use Case Name: Identify Occupancy Status

Actors: AirbagControlUnit

Precondition: The system has been initialized

. . .

Basic Flow

1. The seat SENDS occupancy status TO the system.

Postcondition: The occupant class for airbag control has been sent.

2. INCLUDE USE CASE Classify occupancy status.

3. The system VALIDATES THAT the occupant class for airbag control is valid.

4. The system SENDS the occupant class for airbag control TO AirbagControlUnit.

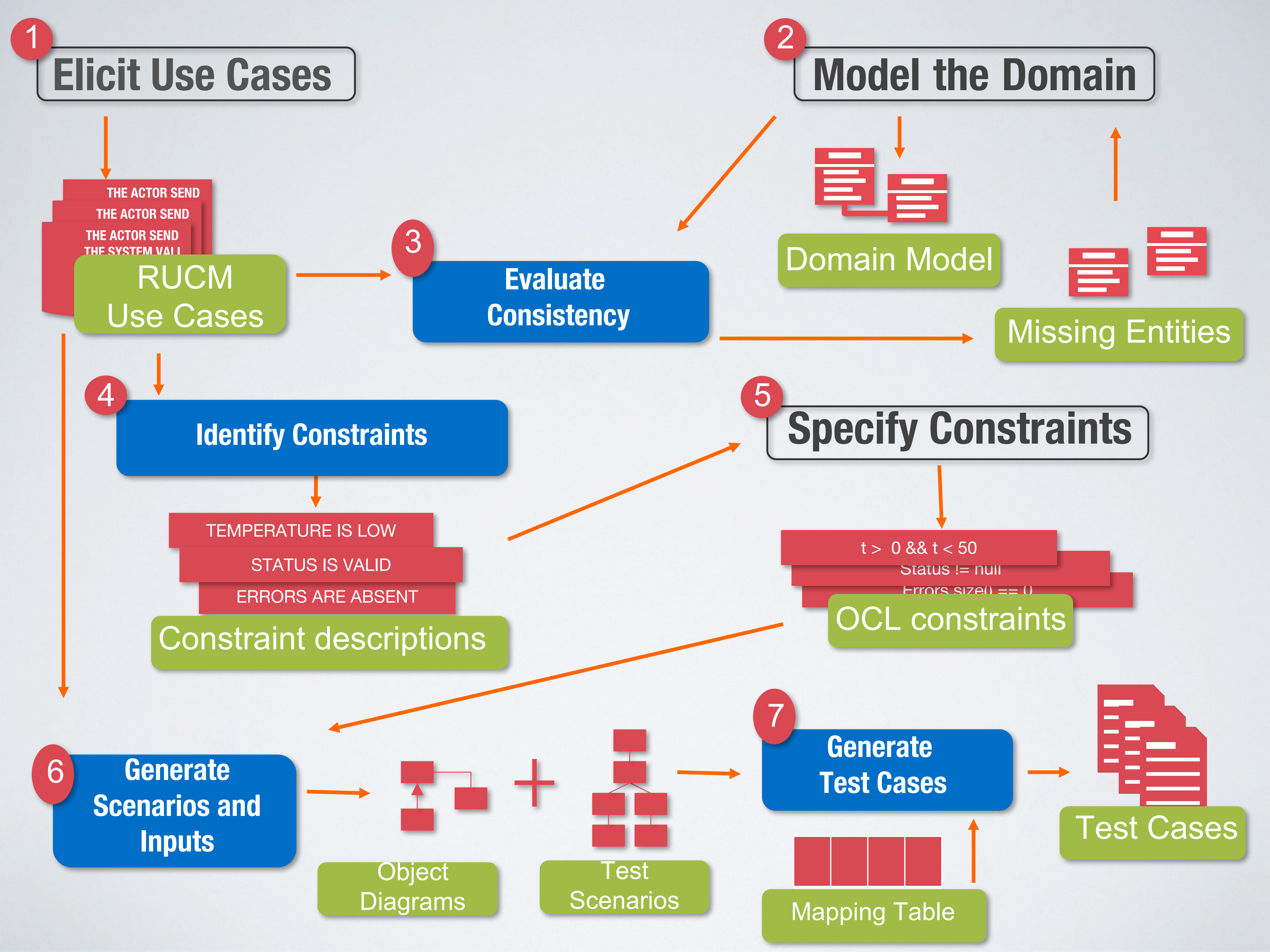
Specific Alternative Flow

RFS 3

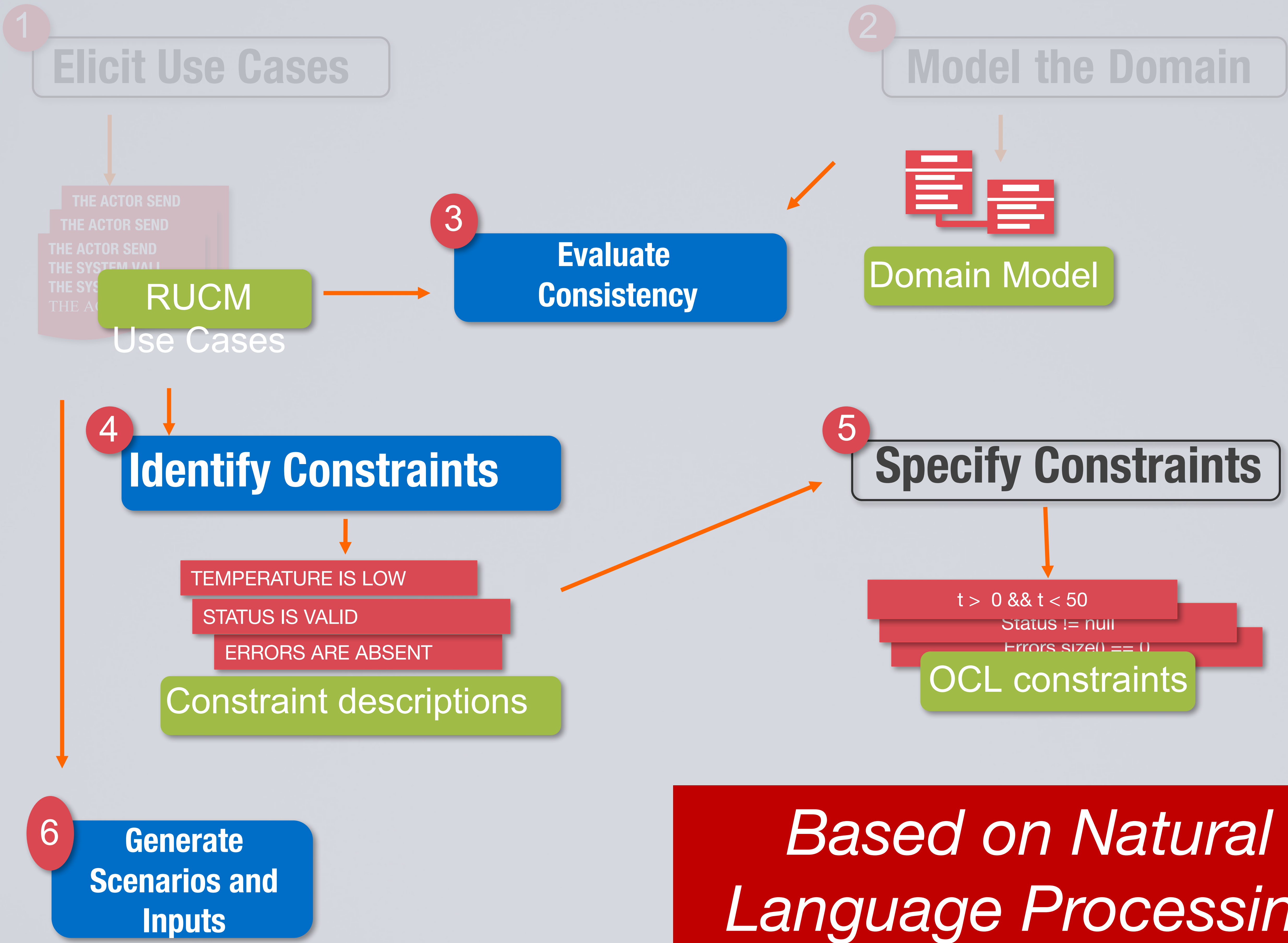
Postcondition: The previous occupant class for airbag control has been sent.

1. IF the occupant class for airbag control is not valid THEN

2. The system SENDS the previous occupant class for airbag control TO



1
Elicit Use Cases
THE ACTOR SEND
THE ACTOR SEND
THE ACTOR SEND
RUCM
Use Cases
2
Model the Domain
Domain Model
Missing Entities
3
Evaluate
Consistency
4
Identify Constraints
TEMPERATURE IS LOW
STATUS IS VALID
ERRORS ARE ABSENT
Constraint descriptions
5
Specify Constraints
t > 0 && t < 50
Status != null
OCL constraints
6
Generate
Scenarios and
Inputs
Object
Diagrams
Test
Scenarios
7
Generate
Test Cases
Mapping Table
Test Cases



Based on Natural Language Processing

Basic Flow

1. The seat **SENDS** occupancy status **TO** the system. → INPUT STEP

   DOMAIN ENTITY

2. **INCLUDE USE CASE** Classify occupancy status. → INCLUDE STEP

3. The system **VALIDATES THAT** → CONDITIONAL STEP

   the occupant class for airbag control is valid and

   CONSTRAINT

   the occupant class for seat belt reminder is valid.

   CONSTRAINT

4. The system **SENDS** the occupant class for airbag control **TO** AirbagControlUnit. → OUTPUT STEP

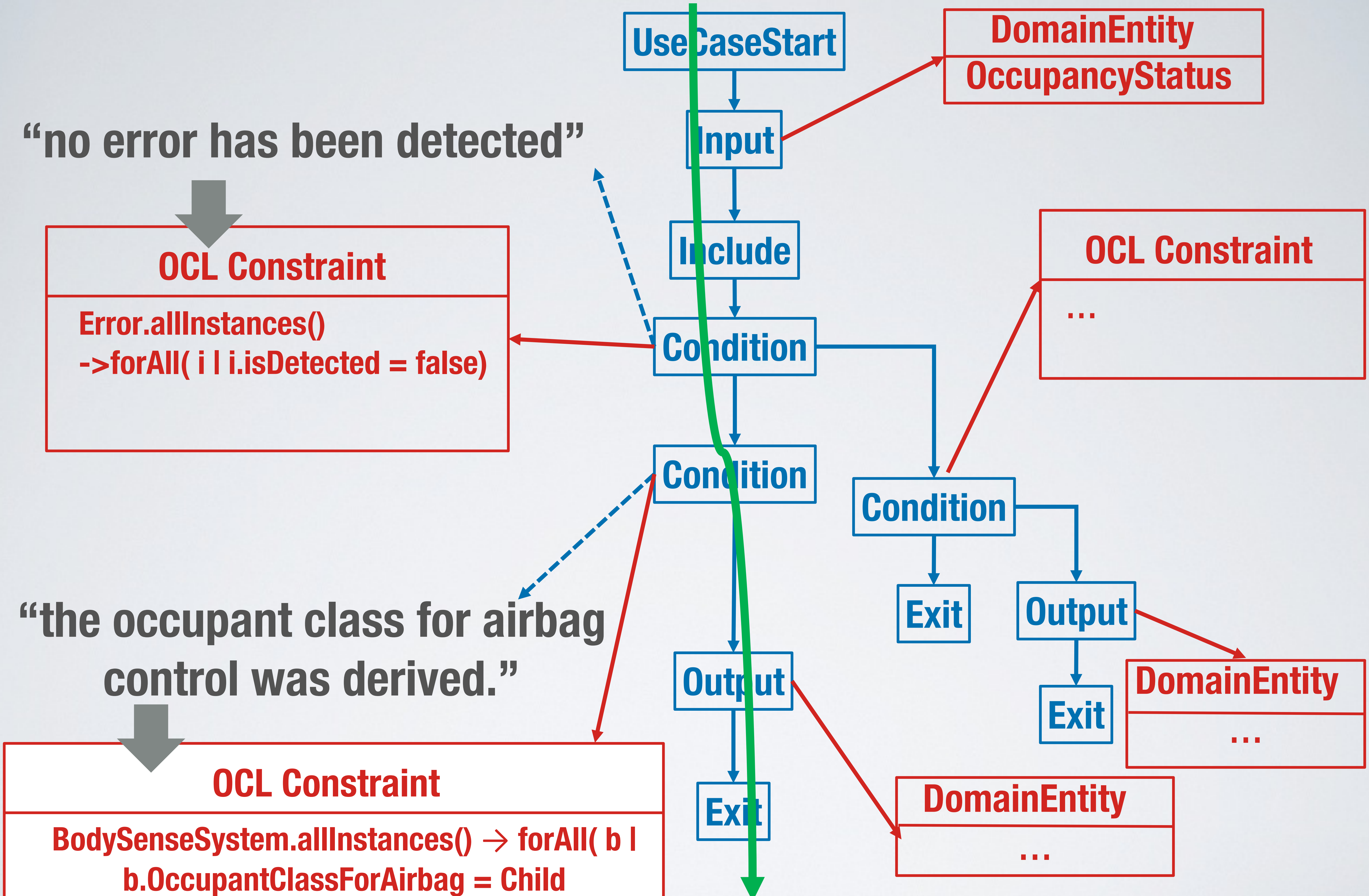
   DOMAIN ENTITY

5. The system **SENDS** the occupant class for seat belt reminder **TO** SeatBeltControlUnit. → OUTPUT STEP

   DOMAIN ENTITY

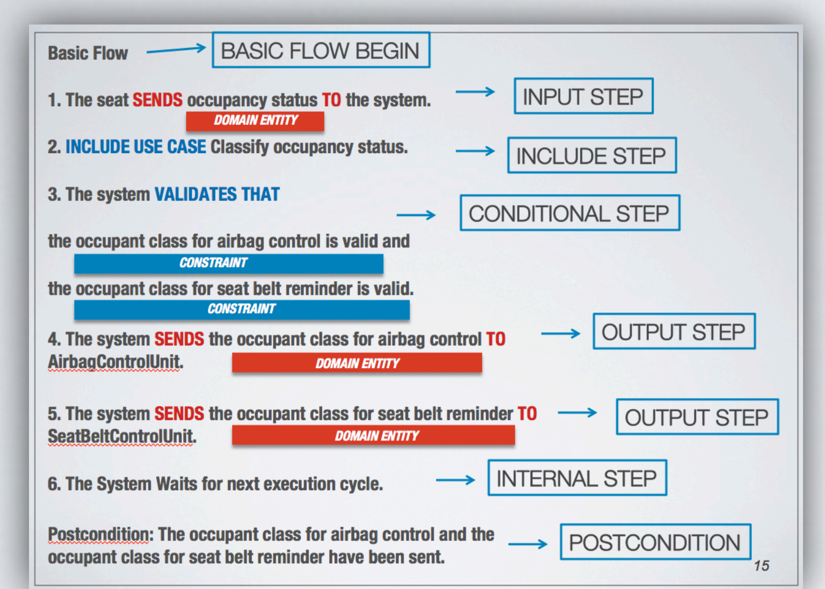
6. The System Waits for next execution cycle. → INTERNAL STEP

Postcondition: The occupant class for airbag control and the occupant class for seat belt reminder have been sent. → POSTCONDITION



Constraint solving: UML2Alloy

Evaluate Model Consistency


Basic Flow → BASIC FLOW BEGIN

1. The seat SENDS occupancy status TO the system. → INPUT STEP
DOMAIN ENTITY

2. INCLUDE USE CASE Classify occupancy status. → INCLUDE STEP

3. The system VALIDATES THAT → CONDITIONAL STEP

the occupant class for airbag control is valid and
CONSTRAINT

the occupant class for seat belt reminder is valid.
CONSTRAINT

4. The system SENDS the occupant class for airbag control TO AirbagControlUnit. → OUTPUT STEP
DOMAIN ENTITY

5. The system SENDS the occupant class for seat belt reminder TO SeatBeltControlUnit. → OUTPUT STEP
DOMAIN ENTITY

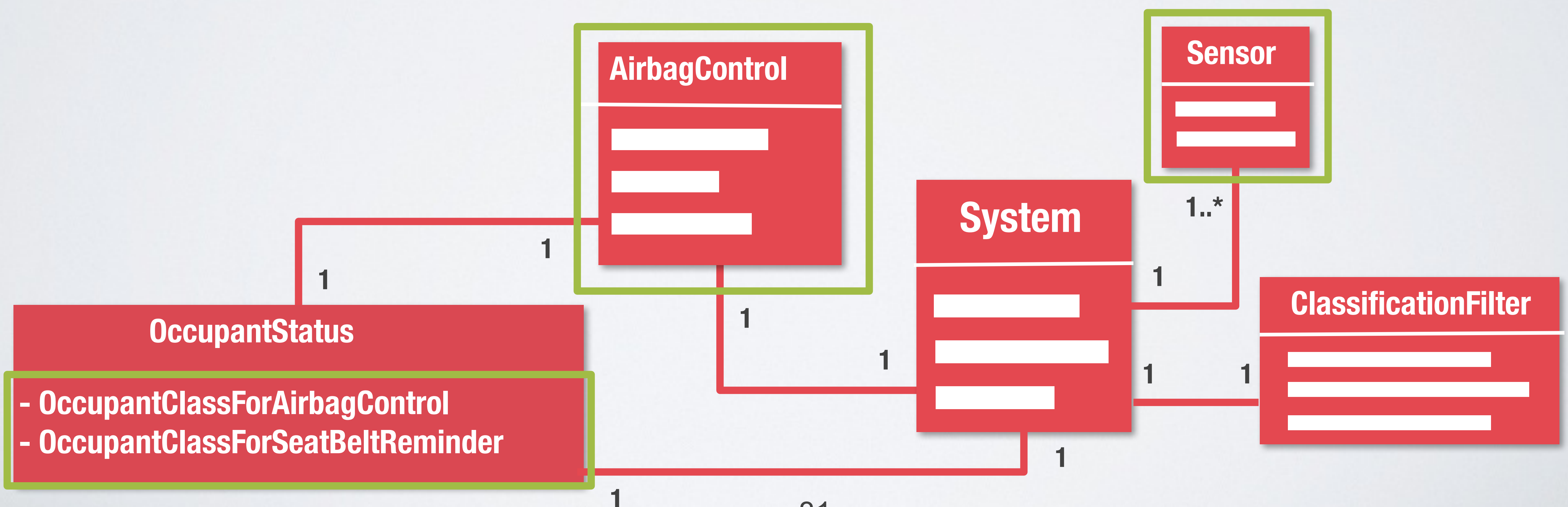
6. The System Waits for next execution cycle. → INTERNAL STEP

Postcondition: The occupant class for airbag control and the occupant class for seat belt reminder have been sent. → POSTCONDITION

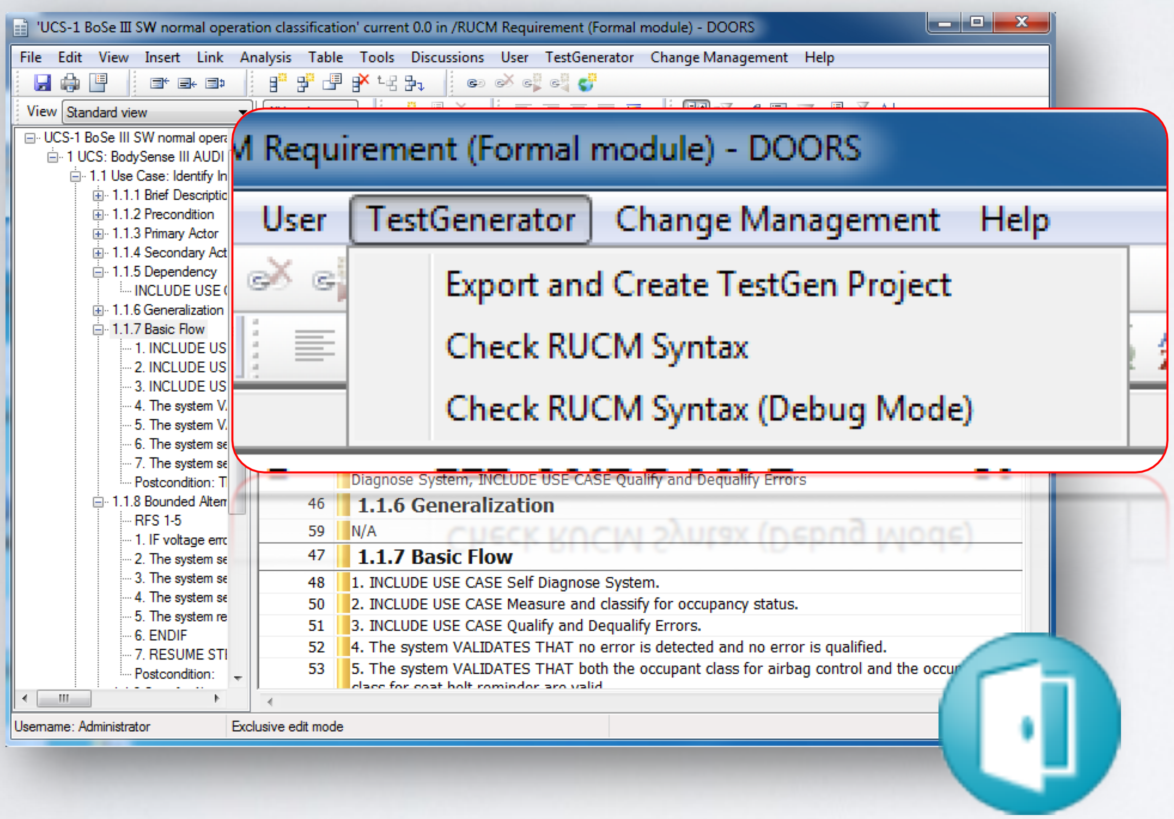
15

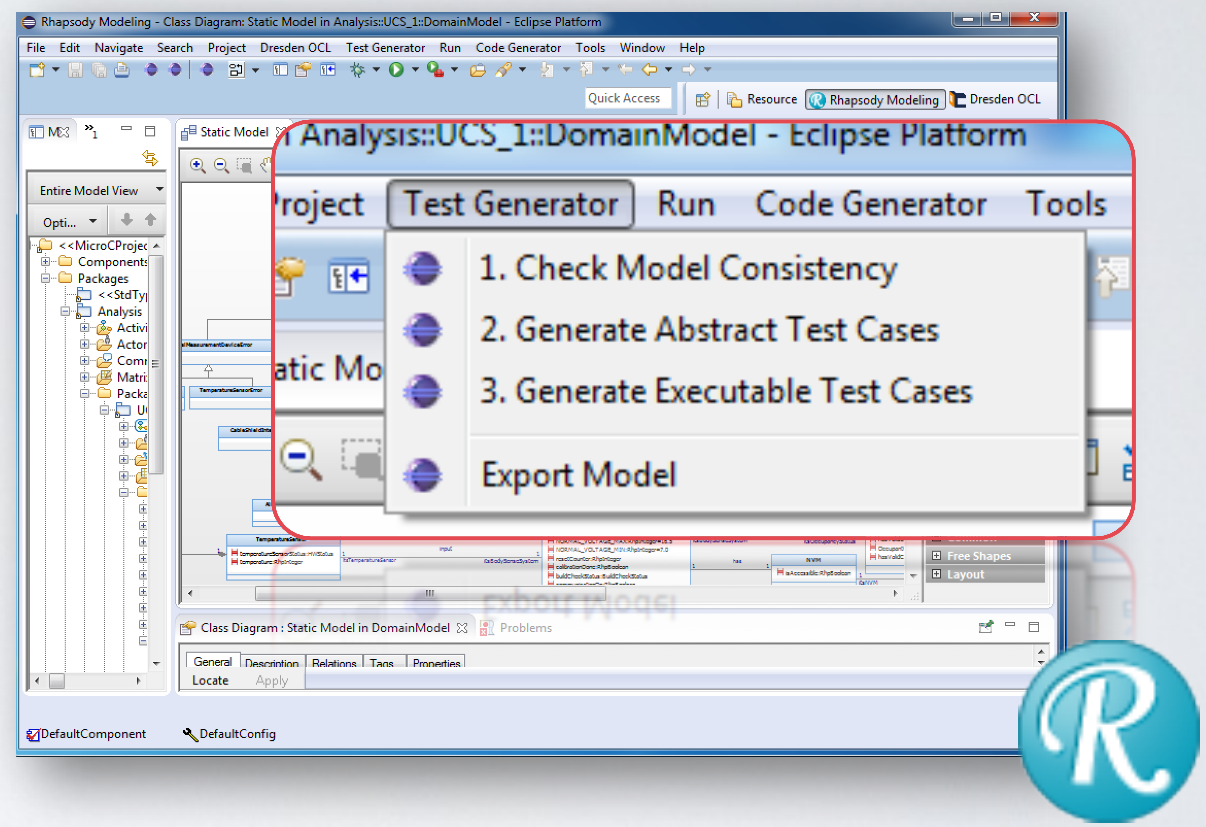
Tagged Use Case

Domain Entities



Toolset integrated with IBM DOORS and Rhapsody





https://sites.google.com/site/umtgTestGen/

Case Study

- BodySense, embedded system for detecting occupancy status in a car



- Evaluation:
  - Cost of additional modelling (Constraints)
  - Effectiveness in terms of covered scenarios compared to current practice at IEE
  - Keep in mind changes and repeated testing

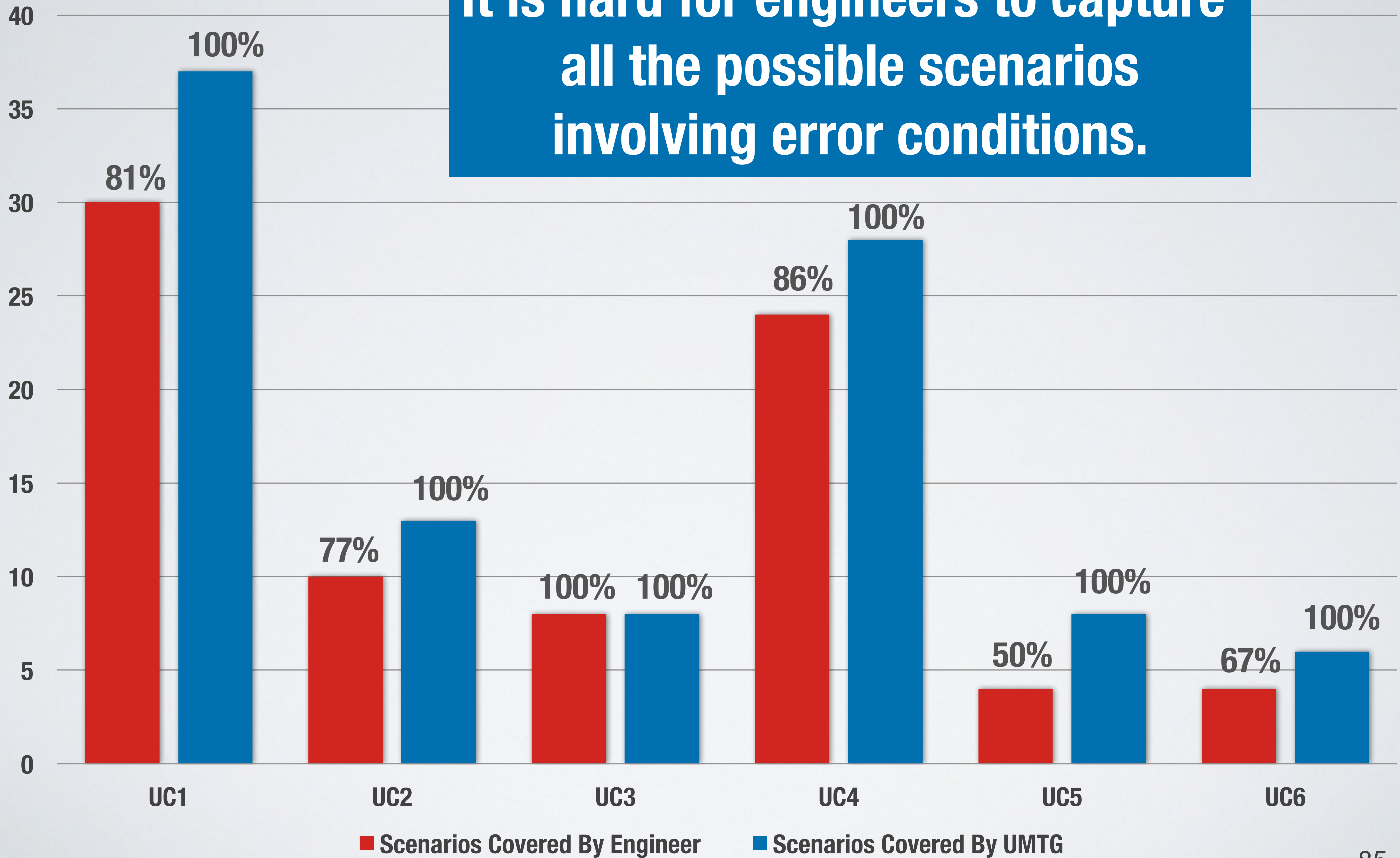
Costs of Additional Modeling

Use Case	Steps	Use Case Flows	OCL Constraints
UC1	50	8	9
UC2	44	13	7
UC3	35	8	8
UC4	59	11	12
UC5	30	8	5
UC6	25	6	12

5 to 10 minutes to write each constraints
=> A maximum of 10 hours in total

Effectiveness: scenarios covered

It is hard for engineers to capture all the possible scenarios involving error conditions.



Generating OCL Constraints

- Constraints may be a challenge in practice
- NLP: Semantic Role Labeling
- Determine the role of words in a sentence (e.g., affected actor)
- Match words with corresponding concepts in the domain model
- Generate an OCL formula based on patterns

Semantic Role Labeling (SRL)

“no error has been detected”

A1 verb

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

A1 verb

A0: actor that performs an activity

A1: actor that is affected by the activity described in a sentence

“The system detects temperature errors

A0 verb A1

TemperatureError.allInstances()->forAll(i | i.isDetected = true)

A1 verb

Empirical Evaluation

- Case study: BodySense, embedded system for detecting occupancy status in a car

- Evaluation:
  - Automatically generate the OCL constraints required to automatically derive executable test cases
  - Automatically generate executable test cases

OCL generation: Precision and Recall

- 88 OCL constraints to be generated
- OCLGen generates 69 constraints
  - 66 correct, only 3 incorrect
- Very high precision

$$\text{precision} = \frac{\#\text{ Correctly generated constraints}}{\#\text{ Generated constraints}} = \frac{66}{69} = 0.97$$

- High Recall

$$\text{recall} = \frac{\#\text{ Correctly generated constraints}}{\#\text{ Constraints to be generated}} = \frac{66}{88} = 0.75$$

Results: Limiting Factors

- **Imprecise specifications**
  - “The system VALIDATES THAT the temperature is valid“

```
BodySense.allInstances()->forAll( i | i.temperature < 200 )
```

- **Inconsistent terminology**
  - “The system VALIDATES THAT the occupancy status is valid“

```
BodySense.allInstances()->forAll( i | i.occupancyStatus <> Empty )
```

Security Testing

Misuse Case Specifications

Security Use Case Specifications

Automated Generation of Executable Test Cases

Security Vulnerability Testing

Addressing the identification of system vulnerabilities

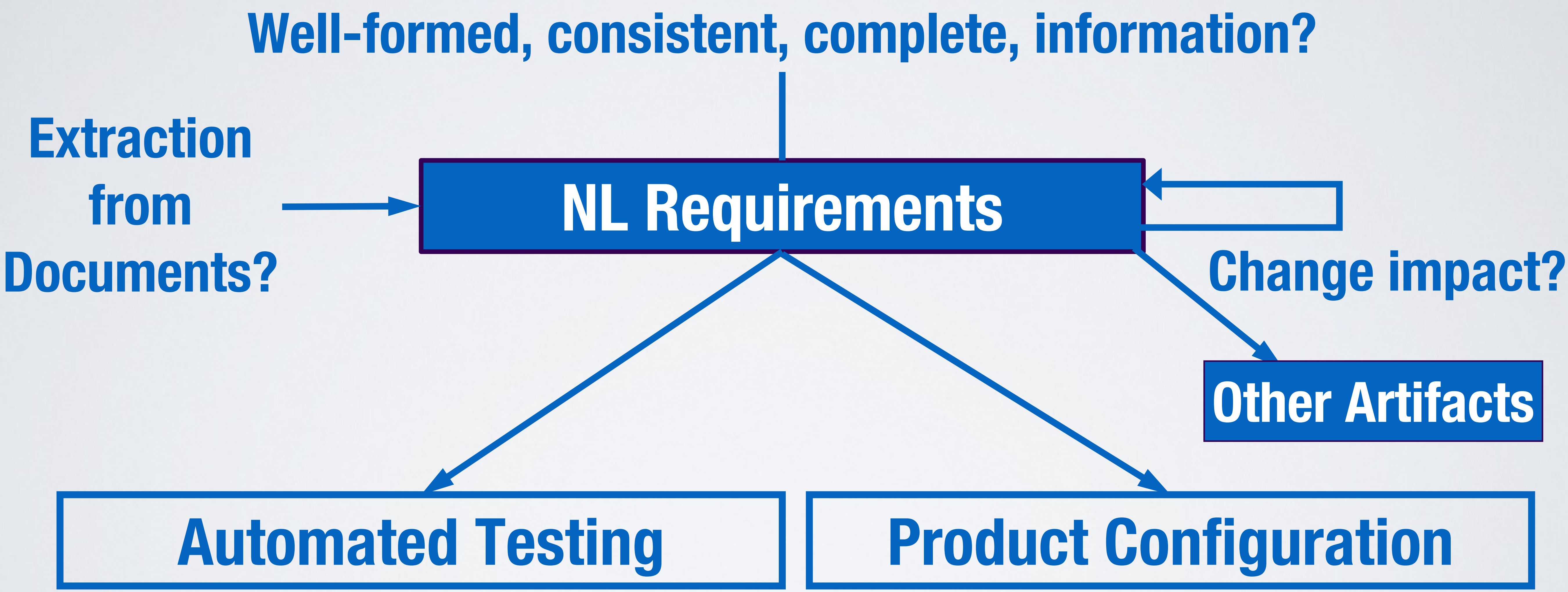
Security Functional Testing

Validating whether the specified security properties are implemented correctly

Benefits of automated generation:

- Automated generation reduces development costs
- Ensures coverage and traceability
- Compliance with standards and regulations

Summary


Well-formed, consistent, complete, information?
Extraction from Documents?
NL Requirements
Change impact?
Other Artifacts
Automated Testing
Product Configuration

The Complexity of our World

- Many applications, diversity of contexts, and types of NL requirements
- Variety of very different working assumptions
  - Form of requirements, e.g., RUCM
  - Change information
  - Modeling practice, e.g., domain models
  - Scale, e.g., embedded automotive versus satellite ground control systems

The Road Ahead

- Practical solutions are possible based on combining advanced NLP and (often) machine learning.
- We must account for practicality and scalability at the outset, not as an afterthought.
- We need more (reported) industrial experiences, as working assumptions play a key role.

Automated Analysis of Natural-Language Requirements:

Industrial Needs and Opportunities

AIRE'2018 @ RE'2018

Lionel Briand
Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT)
University of Luxembourg, Luxembourg

References

Analysis of Natural Language Requirements

- [TSE 2017] C. Arora et al., Automated Extraction and Clustering of Requirements Glossary Terms
- [MODELS 2016] C. Arora et al., Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation
- [RE 2015] C. Arora et al., Change Impact Analysis for Natural Language Requirements: An NLP Approach
- [TSE 2015] C. Arora et al., Automated Checking of Conformance to Requirements Templates using Natural Language Processing

Requirements-Driven (Security) Testing

- [ISSTA 2015] C. Wang et al., Automatic Generation of System Test Cases from Use Case Specifications
- [ICST 2017] C. Wang et al., System Testing of Timing Requirements based on Use Cases and Timed Automata
- [ICST 2018] C. Wang et al., Automated Generation of Constraints from Use Case Specifications to Support System Testing
- [ISSRE 2018] P. Mai et al., A Natural Language Programming Approach for Requirements-based Security Testing

References

Product Families and Configuration

- [SoSYM 2018] I. Hajri et al., Configuring Use Case Models in Product Families
- [JSS 2018] I. Hajri et al., Change Impact Analysis for Evolving Configuration Decisions in Product Line Use Case Models

Impact Analysis

- [FSE 2016] S. Nejati et al., Automated Change Impact Analysis between SysML Models of Requirements and Design
- [RE 2015] C. Arora et al., Change Impact Analysis for Natural Language Requirements: An NLP Approach