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?

Change impact?

Other Artifacts

NL Requirements

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

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

1. Process requirements statements
2. Apply change
3. Identify differences
4. Specify propagation condition
5. Sort requirements based on relevance to change

Requirements document

Phrase annotations

Similarity functions $S(x, y)$

Sorted requirements

Boolean expression

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

1 case study

158 Requirements
9 change scenrios

72 Requirements
5 Change Scenarios
Effectiveness of Our Approach

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

Case-A

Case-B
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

«requirement»
Temperature Diagnostics

text = "The CP controller shall provide temperature diagnostics."

id = "R1"

«requirement»
Over-Temperature Detection

text = "The CP controller shall detect temperatures exceeding 110 °C."

id = "R11"

«requirement»
Operational Temperature Range

text = "The CP controller shall be able to measure temperatures between -20 °C and 120 °C."

id = "R12"
Structural Diagram

- **Over-Temperature Monitor**
- **Temperature Processor**
- **Diagnostics Manager**
- **Diagnostics and Status Signal Generation**
- **DC Motor Controller**
- **Digital to Analog Converter**

<<requirement>> Over-Temperature Detection (R11)
<<requirement>> Operational Temperature Range (R12)

<<satisfy>>
<<satisfy>>
Behaviouiral Diagram

Diagnostics Manager

<<Decision>>
Is position valid?

<<Assignment>>
Error = 1

<<Decision>>
Over-Temp detected?

<<Assignment>>
MotorDriveMode = RUN

<<Assignment>>
MotorDriveMode = OFF
Compute Impacted Elements

Structural Analysis

Behavioural Analysis
Over-Temperature Monitor

Diagnostics and Status Signal Generation

Digital to Analog Converter

DC Motor Controller

Temperature Processor

Over-Temperature Detection (R11)

Operational Temperature Range (R12)

<<requirement>>

<<requirement>>

<<satisfy>>

<<satisfy>>
Change to R11: Change over temperature detection level to 147°C from 110°C.
Diagnostics Manager

<<Decision>>
Is position valid?

[yes] [no]

<<Decision>>
Over-Temp detected?

[yes] [no]

<<Assignment>>
Error = 1

<<Assignment>>
MotorDriveMode = RUN

<<Assignment>>
MotorDriveMode = OFF
Diagnostics Manager

<<Decision>>
Is position valid?

<<Assignment>>
Error = 1

<<Decision>>
Over-Temp detected?

<<Assignment>>
MotorDriveMode = RUN

<<Assignment>>
MotorDriveMode = OFF

Behavourial Diagram

input from B2

output to B4

output to B5

B3
Structural Diagram

- Over-Temperature Monitor
- Diagnostics Manager
- DC Motor Controller
- Digital to Analog Converter

- Operation Temperature Range (R11)
- Operational Temperature Range (R12)

<<requirement>>
<<satisfy>>
Change to R11: Change over temperature detection level to 147 °C from 110 °C.

Natural Language Processing Analysis

Ranked according to likelihood of impact

B2, B3, B4, B6
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

Build SysML Models

Process Change Statements

Phrases

Similarity Matrix

Requirements and Design Models

Compute Impacted Elements

Estimated Impact Set

Sort Elements

System Requirements

Traceability Information Model

Sorted Elements

42
Evaluation

DELPHI
Innovation for the Real World

1 case study

370 elements
16 change scenarios
Effectiveness of Our Approach

Futile Inspection Effort (%)

Structural
Effectiveness of Our Approach

![Box plot comparing Structural and Behavioural Futile Inspection Effort (%)]

- **Futile Inspection Effort (%)**
  - Structural
  - Behavioural
Effectiveness of Our Approach

1 impacted element missed out of a total of 81 impacted elements.
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

Specify Requirements

Build Domain Model

Requirements Analysts

NL Requirements Document

Domain Model

Class A

Class B

Class C

Class D
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

Lift Dependencies to Semantic Units

Construct Domain Model

Domain Model

Class A
Class B
Class C
Class D

1

Relation

Phrasal Structure
Phrasal Dependencies

Phrase-level Dependencies

Extraction Rules
Approach

NL Requirements

Process Requirements Statements

Phrasal Structure Dependencies

Lift Dependencies to Semantic Units

Phrase-level Dependencies

Construct Domain Model

Domain Model

Class A

Class B

Class C

Class D

Relation 1

Extraction Rules

Class A

Class B

Class C

Class D

Relation 1

Extraction Rules
The system operator shall initialize the simulator configuration.
The system operator shall initialize the simulator configuration.
**Approach**

- **NL Requirements**
  - Process Requirements Statements
  - Phrasal Structure
  - Dependencies

- **Lift Dependencies to Semantic Units**
  - Phrase-level Dependencies

- **Construct Domain Model**
  - Domain Model
  - Class A
  - Class B
  - Class C
  - Class D

- 23 Extraction Rules
The simulator shall send log messages to the database via the monitoring interface.
How useful is our approach?

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

50 Requirements
213 Relations
Results

Correctness: 88% (avg.)

Relevance: 37% (avg.)

Missed: 10% (avg.)

Incorrectness largely explained by NLP errors

Breakdown of the remaining 63%

12% are incorrect
51% are correct but superfluous
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

1. Pick relation(s) to inspect
2. Inspect relation(s)
3. Decide about relevance

Recommendations

Classification Model

labeled relations
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
International Electronics & Engineering (IEE)

IEE develops real-time embedded systems:
• Automotive **safety sensing systems**
• Automotive **comfort & convenience** systems, e.g., Smart Trunk Opener
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

- **Analyzeable** use case specifications
- Automatically extract test model from the use case specifications using **Natural Language Processing**
- **Minimize modeling**, domain modeling only
- **No behavioral modeling**
Errors.size() == 0

Status != null

t > 0 && t < 50

Constraints

Test Scenarios

Test Cases

UMTG

Use Cases

Domain Model

Evaluate Consistency
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.
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
2. Model the Domain
3. Evaluate Consistency
4. Identify Constraints
5. Specify Constraints
6. Generate Scenarios and Inputs

TEMPERATURE IS LOW
STATUS IS VALID
ERRORS ARE ABSENT

OCL constraints:
- $t > 0 \&\& t < 50$
- $\text{Status} \neq \text{null}$
- $\text{Errors.size} == 0$

Based on Natural Language Processing
Basic Flow

1. The seat **SENGS** occupancy status **TO** the system. **DOMAIN ENTITY**

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

3. The system **VALIDATES THAT**

   the occupant class for airbag control is valid and **CONSTRAINT**

   the occupant class for seat belt reminder is valid. **CONSTRAINT**

4. The system **SENGS** the occupant class for airbag control **TO** 
   AirbagControlUnit. **DOMAIN ENTITY**

5. The system **SENGS** the occupant class for seat belt reminder **TO** 
   SeatBeltControlUnit. **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**
“no error has been detected”

OCL Constraint

Error.allInstances() \rightarrow\forall i \ (i\text{.isDetected} = false)

“the occupant class for airbag control was derived.”

OCL Constraint

BodySenseSystem.allInstances() \rightarrow\forall b \ (b\text{.OccupantClassForAirbag} = \text{Child} \ \text{OR} \ b\text{.OccupantClassForAirbag} = \text{Adult} \ )

Constraint solving: UML2Alloy
Evaluate Model Consistency

Tagged Use Case

Domain Entities

Airbag Control
Classification Filter
Sensor

Occupant Class for Airbag Control
Occupant Class for Seat Belt Reminder

Basic Flow
1. The seat SENDS occupancy status TO the system.
2. INCLUDE USE CASE Classify occupancy status.
3. The system VALIDATES THAT:
   the occupant class for airbag control is valid and
   the occupant class for seat belt reminder is valid.
4. The system SENDS the occupant class for airbag control TO
   AirBagControlUnit.
5. The system SENDS the occupant class for seat belt reminder TO
   SeatBeltControlUnit.
6. The System Waits for next execution cycle.
Postcondition: The occupant class for airbag control and the
occupant class for seat belt reminder have been sent.

AirbagControl
OccupantStatus
- OccupantClassForAirbagControl
- OccupantClassForSeatBeltReminder

System
ClassificationFilter
Sensor

81
Toolset integrated with IBM DOORS and Rhapsody

[Image with screenshots of software interfaces]

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

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Steps</th>
<th>Use Case Flows</th>
<th>OCL Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC1</td>
<td>50</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>UC2</td>
<td>44</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>UC3</td>
<td>35</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>UC4</td>
<td>59</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>UC5</td>
<td>30</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>UC6</td>
<td>25</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>

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”

Error.allInstances().filterAll(i | i.isDetected == false)

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

“The system detects temperature errors”

TemperatureError.allInstances().filterAll(i | i.isDetected == true)

“A0: actor that performs an activity

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

Benefits of automated generation:

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

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
Summary

Well-formed, consistent, complete, information?

Extraction from Documents?

NL Requirements

Automated Testing

Product Configuration

Change impact?

Other Artifacts
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
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

- [ISSST 2015] C. Wang et al., Automatic Generation of System Test Cases from Use Case Specifications
References

Product Families and Configuration

- [SoSYM 2018] I. Hajri et al., Configuring Use Case Models in Product Families

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