Analyzing Natural-Language Requirements: Industrial Needs and Scalable Solutions

UU/SIKS Symposium on Natural Language in Requirements Engineering, 30 November 2017

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

- NL requirements come in a variety of forms
- NL requirements won’t go away
- Many and varying industrial needs
- NLP has made a huge leap forward in recent years
- Research leading to practical and scalable solutions
- Context factors, working assumptions
Outline

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

- Compliance with requirements templates
- Change impact analysis
- Domain knowledge extraction
- Requirements completeness assessment
- Requirements-driven testing
- Product lines and configuration
Checking Compliance with Templates
Representative Context

SES

your satellite company
Challenges

- Large projects (e.g., ESA)
- Hundreds of natural language requirements
- Tiers of requirements
- Many stakeholders
- Requirements capture a contract
- Requirements frequently change
Compliance with Templates

• Templates and guidelines address ambiguity and incompleteness in NL requirements
• Large number of requirements
• People tend not to comply with templates and guidelines, unless they are checked and enforced
• Scalable and accurate automation is needed
• Existing tools (DODT, RQA) require glossary or ontology
As soon as the visual notification is presented the SOT Operator shall launch the local S&T application as a separate process.
Approach

• **Text chunking:** identifies sentence segments (chunks) without performing expensive analysis
• **NLP parsing** only when needed
• Templates: **RUPP and EARS**, expressed as BNF grammars and then pattern matching rules
• **Practical:** No reliance on glossary, ontology …
• **Scalable:** Hundreds of requirements in a few minutes
Text Chunking

Process of decomposing a sentence into non-overlapping segments.

As soon as the visual notification is presented the SOT Operator shall launch the local S&T application as a separate process.

Noun Phrase (NP) Verb Phrase (VP) Subordinate Clause (SBAR)

Prepositional Phrase (PP) Adverbial Phrase (ADVP)
As soon as the visual notification is presented the SOT Operator shall launch the local S&T application as a separate process.
Evaluation

2 case studies

380 Requirements
380 Requirements

1 case study

110 Requirements

1 case study

890 Requirements
Results

- Absence of glossary has no significant impact on the accuracy of template conformance checking

- Avg. Recall - 94.3%

- Avg. Precision - 91.6%
Tool: RETA

http://sites.google.com/site/retanlp/
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
- Results: We can accurately pinpoint which requirements should be inspected for potential changes
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)
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.

- **Consider semantically-related phrases** that are not exact matches and **close syntactic variations** across requirements

- **Account for change rationale** expressed by user
Narcia

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

Phrase annotations
Similarity functions

S(x, y)

Requirements document

Sorted requirements

Boolean expression

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

SES
your satellite company

1 case study

158 Requirements
9 change scenrios

72 Requirements
5 Change Scenarios

novay

1 case study

27
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

- **Temperature Diagnostics**
  - `text` = "The CP controller shall provide temperature diagnostics."
  - `id` = "R1"

- **Over-Temperature Detection**
  - `text` = "The CP controller shall detect temperatures exceeding 110 °C."
  - `id` = "R11"

- **Operational Temperature Range**
  - `text` = "The CP controller shall be able to measure temperatures between -20 °C and 120 °C."
  - `id` = "R12"
Structural Diagram

<<requirement>>
Over-Temperature Detection (R11)

<<requirement>>
Operational Temperature Range (R12)

<<requirement>>
Digital to Analog Converter

<<satisfy>>
Temperature Processor

<<satisfy>>
Diagnostics Manager

Diagnostics and Status Signal Generation

DC Motor Controller

Over-Temperature Monitor

B1

B2

B3

B4

B5

B6
Diagnostics Manager

<<Decision>>
Is position valid?

[yes] [no]

<<Decision>>
Over-Temp detected?

[yes] [no]

<<Assignment>>
Error = 1

<<Assignment>>
MotorDriveMode = RUN

<<Assignment>>
MotorDriveMode = OFF
Compute Impacted Elements

Structural Analysis

Behavioural Analysis
Structural Diagram

<<requirement>>
Over-Temperature Detection (R11)

<<requirement>>
Operational Temperature Range (R12)

<<satisfy>>
Over-Temperature Monitor

Diagnostics and Status Signal Generation

DC Motor Controller

Digital to Analog Converter
Change to R11: Change over temperature detection level to 147 C from 110 C.
Structural Diagram

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

- Operational Temperature Range (R12)
- Over-Temperature Detection (R11)

- B1: Temperature Processor
- B2: Over-Temperature Monitor
- B3
- B4
- B5
- B6
Diagnostics Manager

**Behavioural Diagram**

- **Decision** Is position valid?
  - [yes]
  - [no]

- **Assignment** Error = 1

- **Decision** Over-Temp detected?
  - [yes]
  - [no]

- **Assignment** MotorDriveMode = RUN

- **Assignment** MotorDriveMode = OFF
Diagnostics Manager

<<Decision>>
Is position valid?

<<Decision>>
Over-Temp detected?

<<Assignment>>
Error = 1

<<Assignment>>
MotorDriveMode = OFF

<<Assignment>>
MotorDriveMode = RUN

Behavioural Diagram

input from B2

output to B5

output to B4
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)
- Operational Temperature Range (R12)

<<satisfy>>

B1

B2

B3

B4

B5

B6
Change to R11: Change over temperature detection level to 147°C from 110°C.

B2, B3, B4, B6

Natural Language Processing Analysis

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

• Computing **similarity scores** for model elements by applying NLP techniques to measure similarity (syntactic and semantic) between model elements labels and change statements.

• **Sorting** 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.
Approach

1. **Build SysML Models**
   - System Requirements
   - Traceability Information Model

2. **Process Change Statements**
   - Requirements and Design Models

3. **Compute Impacted Elements**
   - Phrases
   - Similarity Matrix
   - Estimated Impact Set

4. **Sort Elements**
   - Sorted Elements

Steps:
- Requirements Changes and Informal Change Statements
- Build SysML Models
- Process Change Statements
- Compute Impacted Elements
- Sort Elements
Evaluation

1 case study

370 elements
16 change scenarios
Effectiveness of Our Approach

Futile Inspection Effort (%)

Structural
Effectiveness of Our Approach

![Box plot showing the distribution of futile inspection effort for structural and behavioural categories.](image-url)
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
- Capturing domain knowledge: **Glossary, domain model**
Glossary Extraction and Clustering
Terminology

- Usually multiple stakeholders, organizations …
- Inconsistent terminology
  - Multiple terms for same concepts
    - element / component / object
  - Multiple representations of same keywords
    - status of Ground Station Interface component
    - Ground Station Interface component’s status
    - Interface component status
Requirements Glossary

- Glossaries help mitigate ambiguities
  - consistent terminology
  - improves communication among stakeholders

- Glossaries are, in practice, rarely (fully) defined before requirements are written
Approach

Identification of Candidate Terms

Candidate Terms

Similarity Calculation

Similarity Matrix

Clustering

Combination and Filtering Heuristics

Similarity Measure

Clusters

Clustering Parameter(s)
**R1** - STS shall supply GSI monitoring information (GSI input parameters and GSI output parameters) to the STS subcontractor.

**R2** - When GSI component’s status changes, STS shall update the progress of development activities.
Evaluation of Glossary Terms

2 case studies
380 Requirements
138 Requirements

1 case study
110 Requirements
Results

Our Approach

△Recall > 20%

VS

JATE
TOPIA
TextRank
TermoStat
TermRaider

No clustering
Results

Our Approach vs

Precision ~

JATE
TOPIA
TextRank
TermoStat
TermRaider
Clustering Evaluation

 SES
 your satellite company

 OPENCOSS

 • Interview Survey

 2 case studies

 Safety-Critical Software

 1 case study

 20 clusters each case study

 27 clusters
How useful is our approach?

- I find this cluster helpful for identifying the related terms for a glossary term.
  - 89.6% (strongly agreed / agreed)

- As the result of seeing this cluster, I can define a glossary term more precisely than I originally had in mind.
  - 88% (strongly agreed / agreed)

- I find this cluster helpful for identifying the variations (synonyms) of a glossary term.
  - 61% (strongly agreed / agreed)
  - 28% (not relevant)
Domain Model Extraction
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
A domain model is a representation of conceptual entities or real-world objects in a domain of interest.
Context

Specify Requirements

Build Domain Model

Requirements Analysts

NL Requirements Document

Domain Model

Class A

Class B

Class C

Class D

Relation 1
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

Dependencies

Phrase-level Dependencies

Extraction Rules
Approach

NL Requirements

Process Requirements Statements

Lift Dependencies to Semantic Units

Construct Domain Model

Phrasal Structure Dependencies

Phrase-level Dependencies

Domain Model

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

1. NL Requirements
2. Process Requirements Statements
3. Lift Dependencies to Semantic Units
4. Construct Domain Model

- Phrasal Structure
- Dependencies
- Phrase-level Dependencies
- 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

1 case study

50 Requirements
213 Relations
Results

Correctness - 90% (avg.)

Relevance - 36% (avg.)

Missed Relations - 8%
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
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 (Model-based testing)
Strategy

• **Analyzable** use case specifications

• Automatically extract test model from the use case specifications ([Natural Language Processing](#))

• **Minimize modeling**, domain modeling only

• **No behavioral modeling**
Errors.size() == 0

Status ! = null

Constraints:
- t > 0 && t < 50
- Errors.size() == 0

Use Cases

Domain Model

Evaluate Consistency

Test Scenarios

Test Cases

UMTG
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** AirbagControlUnit.
Elicit Use Cases

RUCM
Use Cases

Identify Constraints

Constraint descriptions

TEMPERATURE IS LOW
STATUS IS VALID
ERRORS ARE ABSENT

Evaluate Consistency

Model the Domain

Domain Model

Missing Entities

Specify Constraints

OCL constraints

Generate Test Cases

Test Cases

Generate Scenarios and Inputs

Object Diagrams

Test Scenarios

Mapping Table
1. **Elicit Use Cases**
2. **Model the Domain**
3. **Evaluate Consistency**
   - Domain Model
   - RUCM
4. **Identify Constraints**
   - Constraint descriptions
   - TEMPERATURE IS LOW
   - STATUS IS VALID
   - ERRORS ARE ABSENT
5. **Specify Constraints**
   - OCL constraints
   - $t > 0 \land t < 50$
   - Status != null
   - Errors.size == 0
6. **Generate Scenarios and Inputs**

*Based on Natural Language Processing*
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.
“no error has been detected”

```
OCL Constraint
Error.allInstances() ->forall( i | i.isDetected = false)
```

“the occupant class for airbag control was derived.”

```
OCL Constraint
BodySenseSystem.allInstances() ->forall( b | b.OccupantClassForAirbag = Child OR b.OccupantClassForAirbag = Adult )
```
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 sends it.
4. The system sends the occupant class for airbag control to the airbag control unit.
5. The system sends the occupant class for seat belt reminder to the seat belt control unit.
6. The system waits for the next execution cycle.
Postcondition: The occupant class for airbag control and the occupant class for seat belt reminder have been sent.

Domain Entities:
- OccupantStatus
  - OccupantClassForAirbagControl
  - OccupantClassForSeatBeltReminder
- AirbagControl
- ClassificationFilter
- Sensor
- System

88
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

• 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
Generating OCL 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
Semantic Role Labeling (SRL)

“no error has been detected”

A1 \downarrow \text{verb}

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

A1 \downarrow \text{verb}

A0: actor that performs an activity

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

“The system detects temperature errors”

\text{A0 \downarrow \text{verb}} \downarrow \text{A1}

\text{TemperatureError.allInstances() \rightarrow forAll( } i \mid i.\text{isDetected} = \text{true) }

A1 \downarrow \text{verb}
Effectiveness: scenarios covered

It is hard for engineers to capture all the possible scenarios involving error conditions.
Supporting Product Lines and Requirements Configuration in Use-Case Driven Development
Configuring Requirements

- Many software systems are part of product families targeting **varying needs among multiple customers**
- Requirements typically need to be tailored or configured for each customer
- Because of interdependencies among such decisions, this is often error-prone and complex
- **How do we support this with natural language requirements?**
IEE develops real-time embedded systems:

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

STO Provides **automatic and hands-free access** to a vehicle’s trunk (based on a **keyless entry system**).
IEE Requirements Engineering

Use Case-Driven Development

Use Case Diagram

Domain Model

Use Case Specifications

<table>
<thead>
<tr>
<th>Use Case ID</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name</td>
<td>Deposit check</td>
</tr>
<tr>
<td>Actors</td>
<td>Customer</td>
</tr>
<tr>
<td>Description</td>
<td>Deposit cash without using ATM card by using E-Card system.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>1. The Customer has an activated E-Card user name and password.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>1. Customer account balance is increased by amount of the deposit check.</td>
</tr>
<tr>
<td>Normal Flow</td>
<td>1. Open the application.</td>
</tr>
<tr>
<td>2. Login into the application.</td>
<td></td>
</tr>
<tr>
<td>3. Choose the account.</td>
<td></td>
</tr>
<tr>
<td>4. Choose the transaction then deposit check service.</td>
<td></td>
</tr>
<tr>
<td>5. Enter the amount of money of the check and submit it.</td>
<td></td>
</tr>
<tr>
<td>6. Receiving the barcode.</td>
<td></td>
</tr>
<tr>
<td>7. Scan the barcode.</td>
<td></td>
</tr>
<tr>
<td>8. Insert the check into the ATM machine.</td>
<td></td>
</tr>
<tr>
<td>10. Log off from the application.</td>
<td></td>
</tr>
<tr>
<td>Alternative Flows</td>
<td>8a. If the customer did not receive the barcode.</td>
</tr>
<tr>
<td>4. Customer will dial on the get barcode button.</td>
<td></td>
</tr>
<tr>
<td>5. Bank sends a new barcode.</td>
<td></td>
</tr>
<tr>
<td>6. Use case resumes on step 8 of normal flow.</td>
<td></td>
</tr>
<tr>
<td>9a. If the ATM did not accept the check.</td>
<td></td>
</tr>
<tr>
<td>3. Restart the check into the ATM.</td>
<td></td>
</tr>
<tr>
<td>4. Use case resumes on step 9 of normal flow.</td>
<td></td>
</tr>
<tr>
<td>Exceptions</td>
<td>8a. In step 8 of the normal flow, if the customer cannot scan the barcode.</td>
</tr>
<tr>
<td>4. Transaction is disapproved.</td>
<td></td>
</tr>
<tr>
<td>5. Customer scan the barcode correctly.</td>
<td></td>
</tr>
<tr>
<td>6. Use case resumes on step 9 of normal flow.</td>
<td></td>
</tr>
</tbody>
</table>
Dealing with Multiple Customers

STO Requirements from Customer A (Use Case Diagram and Specifications, and Domain Model) evolves to STO Test Cases for Customer A (clone-and-own) and then modifies to STO Requirements from Customer B (Use Case Diagram and Specifications, and Domain Model). This process continues with STO Requirements from Customer C (Use Case Diagram and Specifications, and Domain Model) evolving to STO Test Cases for Customer C (clone-and-own) and then modifying to STO Requirements from Customer D (Use Case Diagram and Specifications, and Domain Model).
Product Line Approach

- A Product Line approach tailored to practice and with minimal overhead (adoption)
- Restricted and analyzable use case specifications (RUCM)
- No feature modeling!
- Variability modeling in use case diagrams and specifications
- Automated configuration guidance for configuring requirements with each customer
- Automated generation of product-specific use case models based on decisions
Use Cases And Domain Model

Identify Commonalities and Variabilities

Customer A for Product X

Configurator

Customer B for Product X

Customer C for Product X

Product-Line Use Cases And Domain Model

evolves

reconfigure

reconfigure

reconfigure

reconfigure

configure
Example Variability Extension

- **Keyword:** INCLUDE VARIATION POINT: ...
- **Inclusion of variation points in basic or alternative flows of use cases:**

<table>
<thead>
<tr>
<th>Use Case: Identify System Operating Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Flow</strong></td>
</tr>
<tr>
<td>1. The system VALIDATES THAT the watchdog reset is valid.</td>
</tr>
<tr>
<td>2. The system VALIDATES THAT the RAM is valid.</td>
</tr>
<tr>
<td>3. The system VALIDATES THAT the sensors are valid.</td>
</tr>
<tr>
<td>4. The system VALIDATES THAT there is no error detected.</td>
</tr>
<tr>
<td><strong>Specific Alternative Flow</strong></td>
</tr>
<tr>
<td>RFS 4</td>
</tr>
<tr>
<td>1. INCLUDE VARIATION POINT: Storing Error Status.</td>
</tr>
<tr>
<td>2. ABORT.</td>
</tr>
</tbody>
</table>
Results

- Tool Support (PUMConf): [https://sites.google.com/site/pumconf/](https://sites.google.com/site/pumconf/)

- NLP is a key instrument

- Positive feedback from engineers, both about the modeling approach and configuration tool

- They confirmed they benefited from:
  - **Understanding the commonalities** and differences across product requirements
  - **Automated guidance** in a configuration that is often complex, i.e., many (interdependent) decisions
Discussion
RE Applications

- Requirements to support a shared understanding among many 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 testing
- Requirements to support change control
- Requirements to support product-line configuration
Forms of Requirements

• **Natural language** statements, complying or not with templates
• **Use case stories**, following various templates
• **Use case specifications**, possibly structured and restricted
• **Mixing models and NL**, e.g., class and activity diagrams
The best form of requirements depends on context, but in most cases significant information is captured in natural language.
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
Automation is required to justify the cost of rigorous requirements engineering
In most cases, we don’t have practical and scalable automation solutions
Conclusions

• NLP technology now provides many opportunities for automation
• But more attention to NL requirements analysis is needed in research
• Many applications, diversity of contexts and types of requirements
• Account for practicality and scalability
• More (reported) industrial experiences, as working assumptions play a key role
Acknowledgements

- Mehrdad Sabetzadeh
- Chetan Arora
- Fabrizio Pastore
- Chunhui Wang
- Arda Goknil
- Ines Hajri
- Shiva Nejati
Analyzing Natural-Language Requirements:
Industrial Needs and Scalable Solutions

UU/SIKS Symposium on Natural Language in Requirements Engineering, 30 November 2017

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

• [Submitted] C. Wang et al., Automated Generation of Constraints from Use Case Specifications to Support System Testing
Product Families and Configuration

- **[MODELS 2015]** I. Hajri et al., Applying Product Line Use Case Modeling in an Industrial Automotive Embedded System: Lessons Learned and a Refined Approach

- **[SoSYM 2016]** I. Hajri et al., A Requirements Configuration Approach and Tool for Use Case-Driven Development

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