Summary of "An Empirical Study on the Potential Usefulness of Domain Models for Completeness Checking of Requirements"

1. Introduction and Problem Context

Requirements completeness—ensuring no critical information is missing—is a cornerstone of software engineering. However, **external completeness** (alignment with domain knowledge) is particularly challenging to verify because it relies on cross-referencing requirements with external sources like stakeholder knowledge or domain models. While domain models (e.g., UML class diagrams) are widely used to structure domain concepts, their empirical utility for detecting incompleteness in natural language (NL) requirements remains understudied. This paper addresses this gap by evaluating whether domain models can systematically reveal omissions in "shall"-style functional requirements, which are prevalent in industry but prone to ambiguity and gaps.

Key Concepts:

- **Internal Completeness**: Requirements are self-contained (e.g., no undefined terms).
- **External Completeness**: Requirements align with domain knowledge (e.g., no missing functions).
- **Domain Models**: Structured representations of domain concepts (classes, attributes, relationships) that serve as a reference for cross-validation.

The study hypothesizes that domain models can act as "sensors" for incompleteness: if a domain model element (e.g., a class or association) has no corresponding reference in the requirements, this signals a potential omission. However, the relationship between domain models and requirements is nuanced. For example:

- **Tacit Knowledge**: Domain models may include abstractions (e.g., "User") not explicitly mentioned in requirements but still valid.
- **Lexical Gaps**: Requirements might use different terminology than domain models (e.g., "dynamic reconfiguration" vs. "modifies" association).

---

2. Methodology: Industrial Case Studies

The study employs **three industrial case studies** to ensure diversity and realism:

1. **Case A**: A simulator module for aerospace applications (163 requirements).
2. **Case B**: A sensor platform for cyber-physical systems (212 requirements).
3. **Case C**: A content management system for safety assurance (110 requirements).

Domain Model Construction

- **Scoping**: To manage complexity, domain modeling focused on a randomly selected subset of 35 requirements per case. Noun phrases (NPs) from these requirements (e.g., "simulator," "configuration") were extracted and reviewed by experts to identify genuine domain concepts.
- **Refinement**: Experts expanded the models by adding abstractions (e.g., "Communication Interface") and relationships, guided by existing sketches and domain documentation. Models were iteratively refined to achieve **feasible completeness**—a pragmatic balance between detail and practicality.
- **Final Models**:

- Case A: 144 concepts, 41 attributes, 95 associations, 65 generalizations.
- Case B: 60 concepts, 7 attributes, 58 associations, 6 generalizations.
- Case C: 54 concepts, 17 attributes, 121 associations, 15 generalizations.

Identifying Omissions

Researchers classified omissions into:

1. **Unspecified Requirements**: Entire requirements missing.
2. **Under-Specified Requirements**: Incomplete details within requirements:
   - **Conditions** (e.g., "after simulation execution").
   - **Constraints** (e.g., "via FTP or web service").
   - **Objects** (e.g., omitting one item from a list like "transfer plan and configuration").

Interdependencies between omissions were documented:

- **Type 1**: At least one segment must remain (e.g., retaining one constraint if multiple exist).
- **Type 2**: Removing one segment forces removal of another (e.g., omitting a parent constraint requires omitting its subordinate).

Traceability and Simulation

- **Trace Links**: Experts manually mapped requirements to domain model elements. For example, in Case A, the requirement "The administrator shall dynamically reconfigure simulations" was traced to the *Administrator* class and the *modifies* association.
- **Monte Carlo Simulation**:
  - **Process**: Randomized removal of requirements/segments across 100 iterations.
  - **Metrics**: Tracked the percentage of **unsupported domain elements** (elements with no trace links after omissions).
  - **Surrogate Analysis**: Keyphrases (NPs/VPs) were used to predict sensitivity without domain models.

---

3. Key Findings

RQ1: Sensitivity of Domain Models to Omissions

- **Near-Linear Relationship**: Domain models showed a consistent, near-linear increase in unsupported elements as omissions grew. For example:
  - **Case A**: Omitting 10% of requirements (16/163) left ~23% of domain elements unsupported.
  - **Case B**: Omitting 10% (21/212) led to ~22% unsupported elements.
  - **Case C**: Omitting 10% (11/110) resulted in ~15% unsupported elements.
- **Unspecified vs. Under-Specified**:
  - Domain models were **4.4x more sensitive** to unspecified requirements. Omitting 5 entire requirements in Case A left 12.5% of elements unsupported vs. 2.3% for omitting 5 constraints.
  - **Reasons**:
    - Unspecified requirements remove entire concepts, while under-specified ones only affect details.

        - Repeated references to concepts (e.g., "simulator" in multiple requirements) reduced sensitivity in Case C.
- **Role of Tacit Elements**: Tacit domain elements (e.g., "User" in Case A) accounted for 20–30% of unsupported elements, highlighting the need for expert judgment to avoid false alarms.

RQ2: Predicting Sensitivity via Keyphrases

- **Strong Correlation**: Keyphrase analysis (NPs/VPs) achieved Pearson's $r > 0.96$ across all cases when predicting unsupported domain elements. For example:
    - Case A: 0.974 ($p < 0.0001$).
    - Case B: 0.980 ($p < 0.0001$).
    - Case C: 0.968 ($p < 0.0001$).
- **Practical Utility**: Organizations lacking domain models can use automated NLP tools to extract keyphrases and approximate sensitivity.

---

4. Implications and Limitations

Practical Insights

- **Prioritize Missing Requirements**: Domain models are most effective for detecting unspecified requirements. Analysts should focus reviews on domain elements with sparse references.
- **Balance Abstraction**: Overly abstract domain models (e.g., tacit elements like "User") may increase false alarms. Models should align with the granularity of requirements.
- **Leverage Automation**: Keyphrase extraction tools (e.g., Apache OpenNLP) can supplement manual tracing, reducing effort.

Threats to Validity

- **Internal**: Subjectivity in domain modeling (e.g., classifying a concept as a class vs. attribute) could affect results. Mitigated by using multiple experts and cross-case consistency.
- **Construct**: The study measures *potential* utility, not real-world analyst performance. A domain model's theoretical sensitivity doesn't guarantee practitioners can exploit it.
- **External**: Models were built under controlled, scoped conditions. Real-world models may lack completeness due to time/budget constraints.

Future Directions

1. **User Studies**: Test how analysts use domain models to detect omissions in practice.
2. **Non-Functional Requirements**: Extend the approach to security, performance, etc., using goal models or ontologies.
3. **Tool Support**: Develop NLP-aided tools to automate traceability and highlight unsupported domain elements.
4. **Scalability**: Validate the method on large-scale requirements (e.g., 1,000+ "shall" statements).

---

5. Conclusion

This study provides the first empirical evidence that **domain models**, particularly UML class diagrams, are highly sensitive to incompleteness in "shall"-style requirements. Their structured representation of domain concepts enables systematic cross-validation, with unspecified requirements (missing entirely) being 4.4x easier to detect than under-specified ones (incomplete details). While the results are promising, real-world adoption requires addressing practical challenges like tacit knowledge and scalability. For organizations, investing in domain modeling—or even lightweight keyphrase analysis—can significantly enhance requirements validation, reducing the risk of costly omissions. Future work should bridge the gap between theoretical sensitivity and practical usability, ensuring domain models become a staple in industrial requirements engineering.