Metrics for Model Driven Requirements Development

Brian Berenbach, Gail Borotto
ICSE 2006
Presenter: Dongwon Kang
Contents

- Introduction
- Overview of the approach
- Model driven requirements process
- Project progress metrics
- Quality metrics
- Project execution
- Issues from using commercial tools
- Conclusion
- Discussion
Introduction

- In Siemens Corporation, a UML model-driven requirement development process is used
  - Project management can be improved through traceability utilized from the hierarchically composed model
- Metrics can be extracted from UML models, and they can be used for project management
  - cf.) Extracting metrics from text-based requirements is relatively harder
- In this paper,
  - Propose metrics from UML models for project progress and quality management
Overview of the approach

Requirement development process using UML model

Progress metric
By identifying the rate of completed model elements

Quality metric
By checking constraints and identifying defects number

Use in project management
Measure the progress of a project
Enable immediate feedback from quality metrics
- To improve quality
Model driven requirements process (1/6)

**Drawing initial use case context diagram**
- New car features
- Subdivide features

**Deriving concrete use cases from features**
- Power window
  - Driver operate window
  - Passenger operate window

**Specifying concrete use cases**
- Derive processes

**Deriving testable requirements**
- Write detail information for each leaf requirement

**Requirement development process**
- Formal model review
- Formal requirement review

- Actor
- Abstract use case \(\rightarrow\) feature
- Concrete use case \(\rightarrow\) process or activity
- boundary object
- Leaf functional requirements
- Leaf non-functional requirements

Make sequence or activity diagram for a main concrete use-case in diagrams
- Express business process using included, extending use cases
- Find related objects (e.g., boundary object)
Model driven requirements process (2/6)

- Drawing initial use case context diagram
  - Start with actors and only one feature
    - As a starting point for traceability
      - Feature: service provided by the system that fulfills one or more stakeholder needs
        - Expressed as abstract use case
Model driven requirements process (3/6)

- Deriving concrete use cases from features
  - Express features with related a set of processes (activities)
Model driven requirements process (4/6)

- Specifying concrete use cases
  - Subdivide each use case and make activity or sequence diagram for the main use case of a diagram
    - Included or extending use cases are expressed as messages
    - Boundary classes are identified from sequence diagrams

![Diagram of use cases and messages](image)
Deriving testable requirements

- Testable requirements are shown at terminals
  - Terminal use case is expressed using specific stereotypes
    - Functional requirements as gears
      - Can be expressed using activity or state diagrams
    - Non-functional requirements as oil cans
  - Write detailed information for each terminal requirement

- Used for test plans later
Model driven requirements process (6/6)

- **Formal model review**
  - Step 1: Check model completeness
    - Check status of each artifact
      - Status information is added to each artifact by a tool (DesignAdvisor)
  - Step 2: informal model review
  - Step 3: formal model review with stakeholders
    - Mainly focus on models, not requirements with descriptions

- **Formal requirement review**
  - Taken with stakeholders
    - Terminal (leaf) use case is used as a unit of review
Project progress metrics (1/2)

- **Use case status**
  - Indicator representing the number of completed use cases
    - Used by getting the percentage of “accepted” use cases
  - Before setting a value “accepted” to a class, make sure that constraints are satisfied
    - Constraints includes
      - All included, extending or child use cases should be accepted
      - No errors should be detected
    - Constraints are automatically detected by tool (DesignAdvisor)

- **Use case diagram status**
  - Indicator representing all the contained use cases are accepted
Project progress metrics (2/2)

- **Missing boundary**
  - Provide a rough estimate UI requirements not been defined yet
    - Based on the rule that all actors have to communicate with a concrete use case through a boundary

- **Dangling use case**
  - Defined as a leaf use case not marked as terminal
  - Gross estimate of the subject areas not been covered
Quality metrics (1/2)

- Represented as constraints
  - Concrete use case not defined
    - Non-leaf concrete use cases should be described with one or more sequence, activity, or state diagrams
  - Hidden artifact
    - Artifacts appear in the model, but not in diagrams
  - Class not instanced
    - Business object does not appear on a sequence or activity diagram
Quality metrics (2/2)

- Represented as constraints (Cont’d)
  - Circular dependency
    - Use case make circular relationship
      - Ex) A includes B, B includes C and C includes A
  - Use case completeness
    - Any included or extending use cases would be shown on a sequence or activity diagram associated with the parent use case
  - Illegal association
    - Extending or interface association between abstract and concrete artifact
  - Mixed use case relationship
    - Included or extending use cases for a parent use case should all concrete or all abstract
Project execution (1/2)

- Daily-based modeling process

Diagram:

- Begin Modeling Session
  - Completed Artifact Descriptions in Spreadsheet
    - Import Previously completed artifact descriptions
      - Generate Completion and Error Metrics
        - Team repair model Errors and/or assign to analysts to be repaired after modeling session
          - Model Incomplete Areas
            - Extract Artifact Spreadsheets
              - Incomplete Artifact Descriptions in Spreadsheet
                - Assign Analysts to Complete Artifact Descriptions
                  - End Modeling Session
Project execution (2/2)

- Merits for using metrics of model driven requirements development
  - Ease of monitoring due to automation of measurement
    • Immediate feedback on quality of work
    • Improved quality and reduced rework
  - Programmatic marking of diagrams and formal definition of acceptance increase productivity
    • Prevent to descend the diagram trees to hunt for incomplete items
Issues from using commercial tools (1/3)

- CASE 1: Using ROSE and RequisitePro
  - Policy
    - Created requirement repository from inside the model
      - Enabled by complete integration between two tools
  - Advantage
    - Possible to create requirement in RequisitePro without leaving ROSE
  - Problems
    - Parent and child are not able to have different types
      - E.g. a requirement with lower level requirements of type security, performance, etc.
    - Name of use case becomes description in RequisitePro
      - Description changes in RequisitePro automatically change models in ROSE
Issues from using commercial tools (2/3)

CASE 2: Using ROSE and DOORS

- Policy
  - Draw model in ROSE and import the model from DOORS
- Advantage
  - “One time” dump from the model to DOORS works well
- Problems
  - Relationship of models is broken
    - If a use case is included in two other use cases, only one can be its parent in the database
Issues from using commercial tools (3/3)

- CASE 3: Using ROSE and Caliber
  - Policy
    - Modeling after significant number of requirements have been defined
      - Model was used to verify the existing requirements
  - Advantage
    - Find out missing features and requirements from existing requirements
Conclusion

- Proposed progress and quality metrics for requirement development
  - Use of metrics improved productivity and quality of requirement development
  - Availability of measurement on an on-demand basis changed the way teams worked
    • Frequent model review (daily basis)

- Future work
  - Tool support for the process needs to be reinforced
Discussion

- Advantage
  - Proposed new metrics for measuring progress and quality of software projects
    - Especially, *progress* metrics has not been introduced to the best of my knowledge
  - Model-based requirement development process is proposed
    - Easy to understand the process and maintain requirements
Discussion

- Shortcomings
  - Lack of quantitative validation of metrics
    - Especially, the usefulness of progress metrics is not proved
      - Even not showed how to use the metrics
  - Problems of proposed metrics
    - Progress metrics
      - Before all use cases are identified, it’s not possible to estimate progress
      - Can be used just for estimation of remained review effort
    - Quality metrics
      - Rules can’t be complete
      - Semantic defects can’t be detected
      - e.g.) conflict between requirements