URDAD as Quality-Driven Process [1]

Fritz Solms, Stefan Gruner and Cuen Edwards

URDAD-MDE subgroup of SSFM
Department of Computer Science
University Of Pretoria

fritz@solms.co.za

June 9, 2011

[1] Accepted as a regular paper at SOMET 2011, the 10 International Conference on Intelligent Software Methodologies, Tools, and Techniques, St Petersburg, 28 - 30 Sept 2011

Problem Specification

- Inferior requirements
  - Core contributor to poor software quality & high cost.
- Formal methods
  - Use mathematical modeling & formal logic to specify & verify requirements.
  - Incur high cost & skills requirements.
- Semi-formal methods
  - Constrain cost & skills requirements.
  - Degree of formalization of process & inputs/outputs.
- Model Driven Engineering (MDE)
  - Fall into class of semi-formal methods
  - Often no defined engineering process.
    - Ad-hoc processes often inefficient and non-predictable (estimation)
  - Design structures, quality & semantics often vary.
    - Increases cost & complexity of model validation, code generation, documentation generation, ...

Problem Specification

- Inferior requirements
  - Core contributor to poor software quality & high cost.
- Formal methods
  - Use mathematical modeling & formal logic to specify & verify requirements.
  - Incur high cost & skills requirements.
- Semi-formal methods
  - Constrain cost & skills requirements.
  - Degree of formalization of process & inputs/outputs.
- Model Driven Engineering (MDE)
  - Fall into class of semi-formal methods
  - Often no defined engineering process.
    - Ad-hoc processes often inefficient and non-predictable (estimation)
  - Design structures, quality & semantics often vary.
    - Increases cost & complexity of model validation, code generation, documentation generation, ...

Problem Specification

- Inferior requirements
  - Core contributor to poor software quality & high cost.
- Formal methods
  - Use mathematical modeling & formal logic to specify & verify requirements.
  - Incur high cost & skills requirements.
- Semi-formal methods
  - Constrain cost & skills requirements.
  - Degree of formalization of process & inputs/outputs.
- Model Driven Engineering (MDE)
  - Fall into class of semi-formal methods
  - Often no defined engineering process.
    - Ad-hoc processes often inefficient and non-predictable (estimation)
  - Design structures, quality & semantics often vary.
    - Increases cost & complexity of model validation, code generation, documentation generation, ...

Problem Specification

- Inferior requirements
  - Core contributor to poor software quality & high cost.
- Formal methods
  - Use mathematical modeling & formal logic to specify & verify requirements.
  - Incur high cost & skills requirements.
- Semi-formal methods
  - Constrain cost & skills requirements.
  - Degree of formalization of process & inputs/outputs.
- Model Driven Engineering (MDE)
  - Fall into class of semi-formal methods
  - Often no defined engineering process.
    - Ad-hoc processes often inefficient and non-predictable (estimation)
  - Design structures, quality & semantics often vary.
    - Increases cost & complexity of model validation, code generation, documentation generation, ...

Definitions

Definition

Quality is the degree to which a set of inherent characteristics fulfills requirements. [2]

[2] David Hoyle, *ISO 9000: 2000 Quality Systems Handbook*. 4th ed, 2000.

[3] P. G Petersen, et al., Software quality drivers and indicators. *System Sciences*, p210 –218 vol.2, 1989.

Definitions

Definition

Quality is the degree to which a set of inherent characteristics fulfills requirements. [2]

Definition

A *quality criterion* is an observable quality characteristic of the solution.

[2] David Hoyle, *ISO 9000: 2000 Quality Systems Handbook*. 4th ed, 2000.

[3] P. G Petersen, et al., Software quality drivers and indicators. *System Sciences*, p210 –218 vol.2, 1989.

Definitions

Definition

Quality is the degree to which a set of inherent characteristics fulfills requirements. [2]

Definition

A *quality criterion* is an observable quality characteristic of the solution.

Definition

A *quality measure* is a quantitative metric for a quality criterion.

[2] David Hoyle, *ISO 9000: 2000 Quality Systems Handbook*. 4th ed, 2000.

[3] P. G Petersen, et al., Software quality drivers and indicators. *System Sciences*, p210 –218 vol.2, 1989.

Definitions

Definition

Quality is the degree to which a set of inherent characteristics fulfills requirements. [2]

Definition

A *quality criterion* is an observable quality characteristic of the solution.

Definition

A *quality measure* is a quantitative metric for a quality criterion.

Definition

A *quality driver* is an activity which improves one or more process or model quality criteria. [3]

[2] David Hoyle, *ISO 9000: 2000 Quality Systems Handbook*. 4th ed, 2000.

[3] P. G Petersen, et al., Software quality drivers and indicators. *System Sciences*, p210 –218 vol.2, 1989.

Abstract

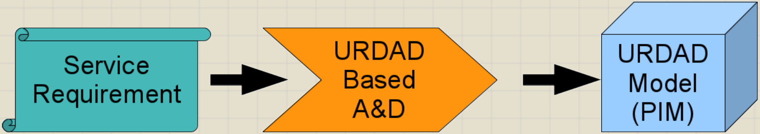
- URDAD is a semi-formal, service-oriented A&D methodology.
  - Generates technology neutral requirements model (PIM).
  - Methodology supported by metamodel & DSL.
- Contributions of this paper
  - We identify for each quality criterion
    - Set of quality drivers.
  - Show quality drivers used in URDAD.

Abstract

- URDAD is a semi-formal, service-oriented A&D methodology.
  - Generates technology neutral requirements model (PIM).
  - Methodology supported by metamodel & DSL.
- Contributions of this paper
  - We identify for each quality criterion
    - Set of quality drivers.
  - Show quality drivers used in URDAD.

URDAD

- Systematic methodology for technology-neutral A&D
  - service-oriented approach
  - generates MDA's PIM



URDAD as recursive analysis & design algorithm


```
class Urdad
{
  provideService(serviceRequirement):Service
  {
    serviceContract = negotiateContract(serviceRequirement)

    try
    {
      return serviceRegistry.getService(serviceContract)
    }
    catch (noRealizingServiceException)
    {
      service = designService(serviceContract)

      for (lowerLevelServiceRequirement : service.requiredServices)
        provideService(lowerLevelServiceRequirement)
    }
  }
}
```

URDAD analysis phase

```
class UrdadAnalysis
{
  negotiateContract(serviceRequirement):ServiceContract
  {
    for (stakeholder:identifyStakeHolders(serviceRequirement))
    {
      functionalRequirements = sourceFunctionalRequirements(
           stakeholder, serviceRequirement)
      qualityRequirements = sourceFunctionalRequirements(stakeholder,
            serviceRequirement)
    }
    negotiateConsistentRequirements()
    groupFunctionalRequirementsIntoServiceRequirements(
         functionalRequirements)
    for (functionalRequirement:functionalRequirements)
      defineCondition(functionalRequirement)
        // includes test & associated exception
    specifyDatastructuresForRequestAndResultClasses()
    assembleServiceContract()
    assignServiceContractToResponsibilityDomain()
    return serviceContract
  }
}
```

Semantic quality

Model quality impacted by quality of modeling language.

- Define semantics via metamodel or ontology.

Semantic quality

Model quality impacted by quality of modeling language.

- Define semantics via metamodel or ontology.

Qualities of modeling language:

- *Completeness*
  - Formal lang: power to express statements needed for URDAD.
    - All meaning to be conveyed can be conveyed.
  - Informally verified through
    - Analyze URDAD process & models for required semantics.
    - Empirically tested via example models.
- *Consistency*
  - Metamodel/ontology is instantiable
  - Verified: transform to ontology & assessed consistency using logical reasoner.
- *Complexity*
  - Assessed by counting classes, relationships & constraints.
  - Much lower than for UML (generic language).
    - UML: 16x more classes, 7x more relationships.

Semantic quality

Model quality impacted by quality of modeling language.

- Define semantics via metamodel or ontology.

Qualities of modeling language:

- *Completeness*
  - Formal lang: power to express statements needed for URDAD.
    - All meaning to be conveyed can be conveyed.
  - Informally verified through
    - Analyze URDAD process & models for required semantics.
    - Empirically tested via example models.
- *Consistency*
  - Metamodel/ontology is instantiable
  - Verified: transform to ontology & assessed consistency using logical reasoner.
- *Complexity*
  - Assessed by counting classes, relationships & constraints.
  - Much lower than for UML (generic language).
    - UML: 16x more classes, 7x more relationships.

Semantic quality

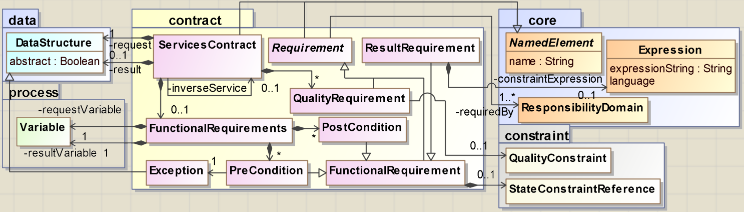
Model quality impacted by quality of modeling language.

- Define semantics via metamodel or ontology.

Qualities of modeling language:

- *Completeness*
  - Formal lang: power to express statements needed for URDAD.
    - All meaning to be conveyed can be conveyed.
  - Informally verified through
    - Analyze URDAD process & models for required semantics.
    - Empirically tested via example models.
- *Consistency*
  - Metamodel/ontology is instantiable
  - Verified: transform to ontology & assessed consistency using logical reasoner.
- *Complexity*
  - Assessed by counting classes, relationships & constraints.
  - Much lower than for UML (generic language).
    - UML: 16x more classes, 7x more relationships.

Example: Language elements for contract specification



Syntactic quality

Ensure statements made in model comply to syntax rules of metamodel.

- Important for model validation, code, test & documentation generation

Syntactic quality

Ensure statements made in model comply to syntax rules of metamodel.

- Important for model validation, code, test & documentation generation

Syntactic quality drivers

- Define concrete syntax for encoding of models.
  - Text-based or diagrammatic.
  - Bi-directional mapping between syntax & metamodel.
  - Enforces URDAD semantics & model structure.
- Generate validating editor for concrete syntax.
  - Done using MDA tool suite.
- Use model validators
  - Compliance to metamodel structure.
  - Adherance to metamodel constraints.

Example: Service contract specification (1/2)

```
ServiceContract enrollForPresentation
{
 FunctionalRequirements receiving Variable
      enrollForPresentationRequest ofType
      EnrollForPresentationRequest
 {
  PreCondition enrollmentPrerequisitesMet requiredBy (
       TrainingRegulator Student) raises
       EnrollmentPrerequisitesNotSatisfiedException checks constraint
        enrollmentPrerequisitesForPresentationMet with ValueOf
       enrollForPresentationRequest
  PostCondition enrollmentProcessPerformed requiredBy (Student
       Client TrainingRegulator) ensures constraint
       studentEnrolledForPresentation with ValueOf
       studentEnrolledRequest constructedUsing doSequential
  {
   create Variable studentEnrolledRequest ofType
        StudentEnrolledRequest
   set Query OCL:"studentEnrolledRequest.personIdentifier" equalTo
        Query OCL:"enrollForPresentationRequest.personIdentifier"
   set Query OCL:"studentEnrolledRequest.presentationIdentifier"
        equalTo Query OCL:"enrollForPresentationRequest.
        presentationIdentifier"
```

Example: Service contract specification (2/2)


```
  ...
  PostCondition invoiceIssued ...
 }
 Request DataStructure EnrollForPresentationRequest
 {
  has identification presentationIdentifier identifying Presentation
  has identification studentIdentifier identifying Person
  has identification clientIdentifier identifying LegalEntity
 }
 Result DataStructure EnrollForPresentationResult { ... }
}
```

Simplicity

Inverse measure of complexity.

- Important because reduces cost, risk & improves maintainability.

Simplicity

Inverse measure of complexity.

- Important because reduces cost, risk & improves maintainability.

Simplicity drivers

- Use DSL to provide compact, precise language.
  - Reduce model size & improves understandability.
- Ensure all process activities address functional requirements.
  - Enforced through metamodel.
- Enforce single responsibility principle
  - Assignment of services to responsibility domains.
- No duplication of statements
  - Only one way to specify things.

Model completeness

The extend to which the model has all elements required for the model use cases

- e.g. code, test & documentation generation.

Model completeness

The extend to which the model has all elements required for the model use cases

- e.g. code, test & documentation generation.

Model completeness drivers

- Structural completeness criteria
  - Certain minimal structure enforced through metamodel.
- Process completeness
  - All functional requirements addressed.
  - Enforced through metaodel constraint.
- No enforced completeness on levels of granularity.
  - Decoupled via services contracts.
  - Service provider need not be designed - could be plugged in.
- Process assistance for completeness via process steps with
  - defined inputs & outputs, and
  - defined process tasks.

Model Consistency

Consistency often problematic in UML models

- Different UML models structurally and even semantically very different.
- Consistency issues across diagrams (e.g. sequence, activity diagrams & state charts).

Model Consistency

Consistency often problematic in UML models

- Different UML models structurally and even semantically very different.
- Consistency issues across diagrams (e.g. sequence, activity diagrams & state charts).

Model consistency drivers

- Repeatable process with defined inputs, outputs & tasks for each process step.
- Enforced model structure & semantics through metamodel.
  - Does not allow duplicate specifications

Model Cohesion

Cohesion refers the extend to which structural realtionships map onto conceptual and functional relationships.

- Important for localized maintenance, easy finding of model elements, testability reusability and understandability.
- High model cohesion results in code with high cohesion.

Model Cohesion

Cohesion refers the extend to which structural realtionships map onto conceptual and functional relationships.

- Important for localized maintenance, easy finding of model elements, testability reusability and understandability.
- High model cohesion results in code with high cohesion.

Model cohesion drivers

- Responsibility localization
  - Contracts contain only services from same responsibility domain.
  - "Encouraged" by process.
- Services as cohesive, self-contained units
  - Statelessness enforced by metamodel.
  - Each service must address complete functional requirement at some level of granularity.

Model modifiability

Modifiability refers to the ease with which the model can be modified.

- Important for maintenance in context of change requests and refactorization.

Model modifiability

Modifiability refers to the ease with which the model can be modified.

- Important for maintenance in context of change requests and refactorization.

Modifiability drivers

- Decoupling via services contracts
  - Modifiability through decoupling.
  - "Enforced" by process & metamodel.
- Guided levels of granularity
  - Process includes step to check whether additional levels of granularity should be defined.
  - Requirements engineer verifies whether any services at any level of granularity can be combined into single, cohesive, higher-level service.
- Simplicity and hence its quality drivers also improve modifiability.

Reusability

Reusability refers to the ease with which model elements can be reused.

- Important for reducing development & maintenance cost & risk, as well as consistency.

Reusability

Reusability refers to the ease with which model elements can be reused.

- Important for reducing development & maintenance cost & risk, as well as consistency.

Reusability drivers

- All services realize services contracts
  - Modifiability through decoupling.
  - "Enforced" by process & metamodel.
- Optimized levels of granularity
  - Process includes step to check whether additional levels of granularity should be defined.
  - Requirements engineer verifies whether any services at any level of granularity can be combined into single, cohesive, higher-level service.
- Stateless, self-contained services.
- Cohesion and hence its quality drivers also improve discoverability and reusability.

Requirements traceability is important for design validation and estimation. Validation includes assessing sufficiency and necessity, i.e., assessing whether all requirements are met and whether all model elements are required. This has to be done across levels of granularity [**?**],whereby four types of traceability should be taken into account [**?**]: *satisfaction* links, *evolutional* links, *rationale* links, and *dependency*.

Traceability

Traceability refers to the ability to trace rationale, satisfaction, dependency and evolution.

- Important for estimation, design validation and maintenance.
- Validation for both, sufficiency and necessity.

Traceability

Traceability refers to the ability to trace rationale, satisfaction, dependency and evolution.

- Important for estimation, design validation and maintenance.
- Validation for both, sufficiency and necessity.

Traceability drivers

- Include satisfaction, dependency and rationale links in model.
- Rationale links
  - only indirectly through link of process activity to functional requirement & functional requirement to stakeholder.
- Satisfaction links
  - between services and service contracts.
- Dependency links throughout.
  - e.g. dependencies between services across levels of granularity.
- Evolutionary links through version control.

Example: Service specification (1/2)

```
Service enrollForPresentationImpl realizes enrollForPresentation
     receiving Variable enrollForPresentationRequest ofType
     EnrollForPresentationRequest
{
 use checkStudentSatisfiesEnrollmentPrerequisites toAddress (
      enrollmentPrerequisitesMet)
 use issueInvoice toAddress (financialPrerequisitesSatisfied
      invoiceIssued)
 ...
 Process doSequential
 {
  create Variable
      checkStudentSatisfiesEnrollmentPrerequisitesRequest ofType
      CheckStudentSatisfiesEnrollmentPrerequisitesRequest
  set Query OCL:"enrollForPresentationRequest.studentIdentifier"
      equalTo Query OCL:"checkEnrollmentPrerequisitesRequest.
      studentIdentifier"
  set Query OCL:"enrollForPresentationRequest.presentationIdentifier
      " equalTo Query OCL:"checkEnrollmentPrerequisitesRequest.
      presentationIdentifier"
  ...
```

Example: Service specification (2/2)


```
requestService checkStudentSatisfiesEnrollmentPrerequisites with
     checkStudentSatisfiesEnrollmentPrerequisitesRequest yielding
     Variable checkStudentSatisfiesEnrollmentPrerequisitesResult
     ofType CheckStudentSatisfiesEnrollmentPrerequisitesResult
choice
{
 if Constraint enrollmentMeetsPrerequisitesMet OCL:"
      checkStudentSatisfiesEnrollmentPrerequisitesResult.
      enrollmentPrerequisitesMet = true" doSequential
 {
  ...
  requestService issueInvoice with issueInvoiceRequest yielding
       Variable issueInvoiceResult ofType IssueInvoiceResult
  {
   on FinancialPrerequisitesNotSatisfiedException raiseException
        FinancialPrerequisitesNotSatisfiedException
  }
  ...
  returnResult enrollForPresentationResult
 }
 else raiseException EnrollmentPrerequisitesNotSatisfiedException
}
}
```

Testability

Testability refers to the ease with which the model or its implementation mapping can be tested.

- Important for model & code validation.

Testability

Testability refers to the ease with which the model or its implementation mapping can be tested.

- Important for model & code validation.

Teatability drivers

- Fully specified services contracts
  - In service-oriented paradigm, services can only be tested by
    - Extracting information about environment using other services.
    - Assessing constraints on obtained information.
- Metamodel
  - Contract has constraint as either pre- or post-condition.
  - Same state constraint can be pre- and post- condition for different services.

Testability

Testability refers to the ease with which the model or its implementation mapping can be tested.

- Important for model & code validation.

Teatability drivers

- Fully specified services contracts
  - In service-oriented paradigm, services can only be tested by
    - Extracting information about environment using other services.
    - Assessing constraints on obtained information.
- Metamodel
  - Contract has constraint as either pre- or post-condition.
  - Same state constraint can be pre- and post- condition for different services.

Example: State constraint specification


```
StateConstraint studentEnrolledForPresentation receiving Variable
    enrollForPresentationRequest ofType
    EnrollForPresentationRequest
{
 stateAssessmentProcess doSequential
 {
  create Variable getEnrollmentsRequest ofType GetEnrollmentsRequest
  set Query OCL:"getEnrollmentsRequest.presentationIdentifier"
      equalTo Query OCL:"enrollForPresentationRequest.
      presentationIdentifier"
  requestService getEnrollments with getEnrollmentsRequest yielding
      Variable getEnrollmentsResult ofType GetEnrollmentsResult
 }
 Constraint OCL:"getEnrollmentsResult.enrollments.includes (
     enrollForPresentationRequest.personIdentifier)"
}
```

Summary of quality drivers in URDAD

Quality-driver	Model qualities									
			Pragmatic model qualities							
	Semantic	Syntactic	Simplicity	Completeness	Modifiability	Consistency	Decoupling	Cohesion	Reusability	Traceability
Define metamodel or ontology	✓	✓	✓	✓	✓	✓	✓			✓
Define concrete DSL grammars		✓	✓		✓					
Define levels of granularity			✓		✓				✓	✓
Decouple services via contracts			✓		✓		✓		✓	✓
Single reponsibility principle			✓		✓			✓	✓	✓
Testable pre- & post-conditions				✓	✓	✓				
Localize controll logic			✓		✓		✓	✓	✓	✓
Include traceability links			✓	✓	✓	✓				✓

Internal consistency of methodology

- URDAD = analysis & design methodology used to design services
  - Apply process to design service of performing analysis & design for service
    - must regenerate itself
    - if it doesn't, then not internally consistent
    - if it does, it does show that URDAD is a good methodology, but only that it is consistent
- Applying URDAD to design service-oriented A&D methodology regenerates
  - process, and
  - metamodel.

Internal consistency of methodology

- URDAD = analysis & design methodology used to design services
  - Apply process to design service of performing analysis & design for service
    - must regenerate itself
    - if it doesn't, then not internally consistent
    - if it does, it does show that URDAD is a good methodology, but only that it is consistent
- Applying URDAD to design service-oriented A&D methodology regenerates
  - process, and
  - metamodel.

Summary

- Linked quality drivers to quality criteria.
- Demonstrated how quality drivers used in URDAD process.
- When using URDAD to design A&D process, one can regenerate URDAD with its metamodel.

Summary

- Linked quality drivers to quality criteria.
- Demonstrated how quality drivers used in URDAD process.
- When using URDAD to design A&D process, one can regenerate URDAD with its metamodel.

Summary

- Linked quality drivers to quality criteria.
- Demonstrated how quality drivers used in URDAD process.
- When using URDAD to design A&D process, one can regenerate URDAD with its metamodel.