Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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 approaches
  - Problems:
    - Often no defined engineering process.
    - Design structures, quality & semantics often vary.

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.

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

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

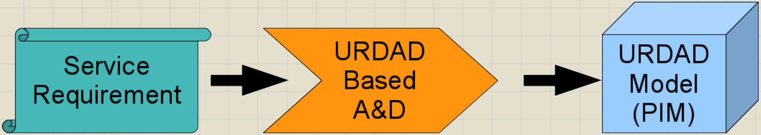
Summary

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 drivers

Model quality impacted by quality of modeling language.

- Define semantics via metamodel or ontology.

Semantic quality drivers

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.

Semantic quality drivers

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.

Semantic quality drivers

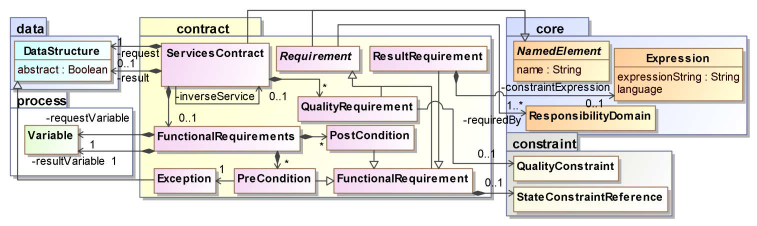
Model quality impacted by quality of modeling language.

- 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 drivers

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

Syntactic quality drivers

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

- Define concrete syntax for encoding of models.
  - Text-based or diagrammatic.
  - Bi-directional mapping between syntax & metamodel.
  - Enforces URDAD semantics & model structure.

Syntactic quality drivers

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

- 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.

Syntactic quality drivers

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

- 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.
    - Valid if constructed via concrete syntax.
  - Adherance to metamodel constraints.
    - Not all metamodel constraints enforced by concrete syntax.

Example: Service contract specification (1/2)

```
ResponsibilityDomain RequirementsEngineering
{
  ServiceContract specifyService
  {
    FunctionalRequirements receiving Variable specifyServiceRequest
         ofType SpecifyServiceRequest
    {
      PreCondition serviceHasStakeholders requiredBy (Client) raises
           NoStakeholdersException checks Constraint
           ServiceHasStakeholders
      PreCondition stakeholderRequirementsConsistent requiredBy (
           Client Implementation Testing) raises
           InconsistentStakeholderRequirementsException checks
           Constraint RequirementsConsistent
      PostCondition serviceContractSpecified requiredBy (Client
           Implementation Testing) ensures Constraint
           ServiceContractSpecified
      PostCondition processSpecified requiredBy (Client
           Implementation) ensures Constraint ProcessSpecified
    }
    ...
```

Example: Service contract specification (2/2)

```
    ...
    QualityRequirement traceability requiredBy (ProcessDesign
         ProjectManagement Development)
    Request DataStructure SpecifyServiceRequest
    {
      has Component serviceRequirements ofType _ServiceRequirements
    }
    Result DataStructure SpecifyServiceResult
    {
      has Component serviceContract ofType _ServiceContract
      has Component service ofType _Service
    }
  }
  Exception NoStakeholdersException {}
  Exception InconsistentStakeholderRequirementsException {}
}
```

Simplicity quality driver

Inverse measure of complexity.

- Use DSL to provide compact, precise language.
    - Reduce model size & improves understandability.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

Simplicity quality driver

Inverse measure of complexity.

- Use DSL to provide compact, precise language.
  - Reduce model size & improves understandability.
- Ensure all process activities address functional requirements.
  - Enforced through metamodel.

Simplicity quality driver

Inverse measure of complexity.

- 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.

Simplicity quality driver

Inverse measure of complexity.

- 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 drivers

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

- Structural completeness criteria
    - Enforced through metamodel.

Model completeness drivers

- Structural completeness criteria
    - Enforced through metamodel.
- Process completeness
    - All functional requirements addressed.
    - Enforced through metaodel constraint.

Model completeness drivers

- Structural completeness criteria
  - 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.

Model completeness drivers

- Structural completeness criteria
  - 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 Drivers

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

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 Drivers

- Responsibility localization
    - Contracts contain only services from same responsibility domain.
    - “Encouraged” by process.

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.

Modifiability Drivers

- Decoupling via services contracts
    - Modifiability through decoupling.
    - “Enforced” by process & metamodel.

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.

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 Drivers

- All services realize services contracts
  - Modifiability through decoupling.
  - “Enforced” by process & metamodel.

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.

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.

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.

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.
- Linkage between service and contract it realizes aids service provider discoverability.

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.
- Linkage between service and contract it realizes aids service provider discoverability.
- Enforced adapter layer.

Traceability Drivers

Traceability important for validation & estimation

- Validation for both, sufficiency and necessity.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

Traceability Drivers

Traceability important for validation & estimation

- Validation for both, sufficiency and necessity.

Quality drivers

- Process requires service identification for each functional requirement.

Traceability Drivers

Traceability important for validation & estimation

- Validation for both, sufficiency and necessity.

Quality drivers

- Process requires service identification for each functional requirement.
- Metamodel provides
  - Traceability of services across levels of granularity.
  - Traceability from service to functional requirement realized.
  - Traceability from functional requirement to stakeholder who requires it.

Example: Service specification

```
Service specifyServiceImpl realizes specifyService receiving
     Variable specifyServiceRequest ofType SpecifyServiceRequest
{
  use identifyStakeholders toAddress (serviceHasStakeholders)
  use performAnalysis toAddress (serviceContractSpecified)
  use designProcess toAddress (processSpecified)

  Process doSequential
  {
    create Variable identifyStakeholdersRequest ofType
         IdentifyStakeholdersRequest
    set Query OCL:"identifyStakeHoldersRequest.serviceRequirements"
         equalTo Query OCL:"specifyServiceRequest.serviceRequirements
         "
    requestService identifyStakeholders with
         identifyStakeholdersRequest yielding Variable
         identifyStakeholdersResult ofType IdentifyStakeholdersResult
          on NoStakeholdersException raiseException
         NoStakeholdersException

    create Variable specifyServiceContractRequest ofType
         SpecifyServiceContractRequest
    ...
```

Testability Drivers

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

- 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.

Testability 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 ServiceContractSpecified receiving Variable
     serviceRequirements ofType _ServiceRequirements
{
    StateAssessmentProcess doSequential
    {
create Variable serviceContractAvailable ofType Boolean
set Query OCL:"serviceContractAvailable" equalTo Constant "true"

create Variable provideServiceContractRequest ofType
     ProvideServiceContractRequest
set Query OCL:"provideServiceContractRequest.serviceRequirements"
     equalTo
Query OCL:"serviceRequirements"
requestService provideServiceContract with
     provideServiceContractRequest
yielding Variable provideServiceContractResult ofType
     ProvideServiceContractResult
on NoServiceContractAvailableException
set Query OCL:"serviceContractAvailable" equalTo Constant "false"
    }
    Constraint OCL:"serviceContractAvailable = true"
}
```

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

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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.

Summary

- Linked quality drivers to quality criteria.
- Demonstrated how quality drivers used in URDAD process.

Problem Specification

Definitions

Abstract

URDAD

Quality drivers embedded in URDAD

Internal consistency

Summary

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.