URDAD as a Quality-Oriented Analysis and Design Process for Model Driven Development

Fritz Solms
Solms Training, Consulting and Development
113 Barry Hertzog Ave, Emmarentia, Johannesburg, 2915, South Africa
fritz@solms.co.za

Riaan Klopper Dept. of Comp. Science, University of Pretoria, South Africa.
Riaan.Klopper2@standardbank.co.za

Stefan Gruner
Dept. of Comp. Science, University of Pretoria, South Africa.
sgruner@cs.up.ac.za

Derrick G. Koerie
Dept. of Comp. Science, University of Pretoria, South Africa.
dkourie@cs.up.ac.za

ABSTRACT

URDAD, the *Use Case, Responsibility Driven Analysis and Design* process is a process which generates the *Platform Independent Model* (PIM) of OMG's framework for model driven development, the *Model Driven Architecture* (MDA). The quality of a PIM is crucial since an inferior quality PIM compromises any further models and artifacts which are generated from the PIM through model transformation and artifact generation technologies. An URDAD generated PIM has a formally defined model structure. In this paper we endeavour to qualitatively assess to what extent an URDAD PIM realizes stake holder quality requirements for the PIM.

Keywords
URDAD, model quality, model driven development, UML ,
MDA

1. INTRODUCTION

Model Driven {Architecture/Engineering/Development} (MDA, MDE, MDD) [**?**, **?**, **?**], as an academic idea has has not seen a practical translation into the IT and software engineering industry to the extent which was initially envisaged.

Part of the problem might be the inherent difficulty of the model transformation process (i.e. refinement) on which the theory of MDD is based, which requires much deeper computer science knowledge than the average IT worker possesses. This dificulty is exacerbated by often not having a well defined model structure as input. In general the contents and structure of different UML models and it is difficult to develop useful transformation tools which can perform model refinement, implementation mapping, test and documentation generation across such widely varying input models.

Originally, tool support was envisaged as a way of circumventing this problem by equipping the semi-skilled IT worker with advanced theoretical knowledge that is embedded within "push-button" types of support systems. However, it soon turned out that the development of such support tools for MDSD is not a trivial task at all. Moreover, a support tool requires a flawless (i.e. syntactically correct and semantically consistent) input model before it can be further processed in an automated fashion—much like a compiler requires syntactically and semantically correct input in a high level language before it can generate target code.

Thus, we discover that even the initial, top-level models in the MDD pyramid tend to be of such poor quality that they cannot sensibly be further processed, even if the desired transformation tools were already available. In other words, the design of the initial models at the highest level of abstraction—from which, according to the theory of MDD, all other deliverables have to be generated—seems to be an intrinsically difficult and error-prone activity.

2. RESEARCH APPROACH

Amongst the many papers which have already addressed this problem from various perspectives, we have chosen [**?**] and [**?**] as the starting points of some our work. Those papers have described quality criteria, which we will refer to as quality *requirements*, that high-quality software models and software development processes should ideally fulfill (and which are typically not fulfilled in practice).

An empirical survey was done in order to determine what model quality requirements are important to which stake holders.

A non-empirical qualitative approach was used to determine if URDAD employes, and enforces all of the necessary design principles to generate a high quality model.

2.1 Building the case

In this paper we clarify who the stake holders are of these quality requirements of software models, as well as re-visit the actual quality requirements as defined by [**?**] and [**?**]. We only use this as a starting point, and build on these quality requirements by adding additional ones. The underlying design principles that aid in the quality requirements will also be discussed.

URDAD, a quality-oriented analysis and design process [**?**], which was designed especially in support of MDD, will be assessed based on the quality requirements, and the underlying design principles enforced by the step by step process.

The result of this study, is to have completed an assessment of the URDAD process from a theoretical perspective, as well as a quantitative assessment of the resultent model. We argue that URDAD succesfully passed both assessment assessments, and consequently, we conjecture that the application of the URDAD process will eventually lead to software models of better quality.

2.2 Paper Structure

The paper is structured as follows. In section **??** we will discuss *related work* from the perspective of URDAD, in other words, *similar approaches* (to software model design) aiming at *similar problems* (of software model quality).

In section **??** we introduce the various stake holders who have an interest in software models and model quality.

Section **??** briefly recapitulates the main features and characteristics of the URDAD process as further described in [**?**] and we then explain how the step by step URDAD process enforce the design principles mentioned.

In section **??** we compare the URDAD features against the previously mentioned quality requirements, including ([**?**] and [**?**]).

Finally, in section **??** we summarise our findings and sketch some future work in the context of this ongoing project.

3. CONTEXT AND RELATED WORK

There has been significant focus on the quality of software over the last few decades. Software projects were run without any formal way of approaching the software effort. As a result, software projects were notorious for failing, being late, over budget, or even delivering the wrong solution.

3.1 Software Engineering Challenges - A Historical Perspective

Brooks [?] argued that projects largely fail because of the inherent difficulties in software. He called it the *"essence"* of software, which includes complexity, conformity, changeability and invisibility. These inherent properties of software undoubtedly affects the quality of the end products produced by software engineering. It is true that many advances in software engineering merely addressed the "accidents" of software, and not the "essence". As will be seen, software modeling is an attempt to solve the perceived inherent difficulties of software, and as a result, addressing various quality issues.

There were numerous initial attempts at addressing this problem. Various methodologies were created to assist the software engineering effort, giving more guidance when building software, thereby increasing the quality of the delivered solution. Avison and Fitzgerald [?] cites The BCS Information Systems Analysis and Design Working Group's definition of a methodology to be a *"recommended collection of philosophies, phases, procedures, rules, techniques, tools, documentation, management, and training for developers of information systems"*.

All of these attempts were essentially process driven initiatives. The aim was to lay down pre-defined steps and rules to be followed, in a hope that it would increase the quality of software. Even-though formal methodologies and processes were a step in the right direction, it could not single handedly address the inherent difficulties in software, of which quality is one.

3.2 Advancements in Software Engineering

This was the beginning of new concepts, such as Agile Software Development, including *Extreme Programming (XP)* [?]. These new concepts and philosophies in software engineering redefined the way organisations viewed and managed project resources and approaches. Another new concept that was introduced was *Object Oriented Analysis and Design (OOAD)*, which raised the level of abstraction at which Analysis and Design is done, thereby bridging the gap between analysis and design (RK to reference).

Another concept that arose was the idea of doing actual programming at a much higher level of abstraction, and moving closer to the knowledge domain at which the end solution is aimed. The *Model Driven Architecture (MDA)* [?, ?] represents the concepts of this movement, and includes the discipline of software modeling. A practitioner should be able to *model* a solution in some predefined notation (such as the UML), and with appropriate tool support, he should be able to do code generation instead of intensive manual programming. This approach aims at bridging the gap between analysis and design even further, by incorporating some of the design decisions in the early analysis phases already, and separating the architectural decisions from the logical design.

As this is a relatively new concept in software development, the approaches and processes aimed at producing conceptual, executable software models are still evolving. There is a need for more substantial empirical evidence of the successful, and unsuccessful implementations of these approaches in the industry. Snelting [?] identified this need to test software methodologies and approaches in practice, and not to only regard it as academic advancement in software engineering. Proper empirical research will assist the evolution and maturing of new model driven approaches. There is an explicit need to be able to produce quality software models in practice, as well as being able to measure the quality in these models.

Although the measurement of software quality is important, the approaches aimed at producing quality in models are even more important from and organisational perspective. The better the quality at the early stages of the project, the more cost effective the project would be [?].

3.3 Related Work

There are various similiar intiatives that exist at the moment, trying to achieve the same goal, namely building high quality models in the context of the MDA. Some of them are mentioned next, in order to sketch the modeling landscape.

3.3.1 Empirical Analysis of Architecture and Design Quality

Empirical Analysis of Architecture and Design Quality (EmpAnADa) [?] is a project that aims at developing techniques to improve the quality of models. A quality model is proposed [?], as well as a supporting prototype to manage the quality model. The model is applied in various phases of the software development effort. The quality model not only measures the quality of the model, but also that of the software system that results from the model.

3.3.2 URDAD

URDAD, which is aimed at addressing the many known quality issues of modeling. We believe that URDAD can address these issues as will be seen in the rest of the paper. URDAD does not explicitly measure model quality. It defines rules and steps in order to create high quality UML models in line with the Platform Independent Model (PIM) requirements of the MDA and provides a model validation suite which assesses whether the resultant UML model has the structure and content required of an URDAD PIM. It is therefore process driven, while incorporating best practice principles and rules. This will be discussed in section **??**.

4. PIM STAKE HOLDERS AND THEIR QUALITY REQUIREMENTS

We take the approach that there is no absolute quality, but that quality is a measure of the extend to which the stake holder's quality requirements are fulfilled. Thus, before we can assess the quality of a (business) model, we need to identify the stake holders who have an interest in the model and then we need to elicit their quality requirements for the model.

The stake holders who have an interest in the PIM include:

- *Client/Business* which gets the return on investment from the product.
- *(Business) Analysts* who are responsible for performing the stake holder requirements analysis and the technology neutral (business) process design itself.
- *Architecture* which is responsible for designing a suitable infrastructure hosting the functionality (business processes) in such a way that it enables the organization/system to realize its vision and mission.
- *Implementation* which is responsible for performing the model transformations and implementation mapping. (In a business this includes developers who develop the automated implementation mapping as well as managers who train their staff to perform certain business process steps manually).
- *Quality assurers* who are responsible for assessing the extend to which the model realizes the stake holder's functional and non-functional requirements and reporting of any defects.
- *Operations* which is responsible for executing / overseeing the execution of the (business) processes and ensuring that the services are rendered to the user's satisfaction.

Table **??** shows the PIM qualities required by these stake holders. The results are based on an abstraction of an empirical study which included interviews with the various stakeholders in the development process.

Table 1: Stake holders and their quality requirements.

quality	Client/Business	Business Analysis	Architecture	Implementation	Quality assurance	Operations
completeness		x	x	x	x	x
consistency		x	x	x	x	x
simplicity/understandability		x		x	x	x
modifiability	x	x				
reusability		x		x		
testability		x		x	x	
traceability		x		x		
cohesion		x		x		

5. URDAD

URDAD, the *Use-Case, Responsibility Driven Analysis and Design* process provides an algorithmic analysis and design process with the following characteristics:

1. Both, requirements and design are step-wise refined.
2. URDAD specifies the steps of the design methodology, indicating the activities, the inputs and the outputs for each step, thus rendering the process repeatable and predictable.
3. The services contracts for the service providers required at any level of granularity are generated, thus enforcing a technology-neutral approach as well as pluggability and testability at any level of granularity.
4. Work flow logic at any level of granularity is factored out of the service providers for that level of granularity, enforcing decoupling of role players across levels of granularity.
5. URDAD provides an explicit approach to fixing the levels of granularity.
6. URDAD explicitly aims to generate a technology- and architecture-neutral design to represent the MDA's PIM.

Since the technology-neutral model should be in the problem domain, the modeling itself should be done by domain experts and *not* by implementation technology specialists. In the case of enterprise system development, the domain experts are typically business analysts. URDAD requires domain experts *across* responsibility domains to collectively contribute to a technology-neutral domain model. Thus, while the analysis and design of a particular service is typically done by the domain specialists whose focus and responsibility lies in the domain of that service, lower level services used in the process are usually analyzed and designed by other domain specialists who focus on their particular responsibility domain.

The process itself enforces established design principles which are known to improve the quality of designs. This is done through specific activities like fixing the level of granularity, enforcing the single responsibility principle, enforcing the definition of services contracts, and so on.

In addition to providing an analysis and design process, URDAD fixes the structure and the contents of the technology-neutral model (PIM) generated by the process. This improves repeatability, enhancing the simplicity of the PIM, and thus also its utility. For example model transformation as well as code, test and documentation generation are all significantly simpler due to the known structure of the input model.

We first dicuss the design principles/activities enforced by the process and the qualities they support. Then we look at the URDAD process itself before discussing the structure of an URDAD PIM.

5.1 Design principles

Table **??** shows the stake holders and their functional requirements. Table **??** shows the design principles/activities which feed those qualities into the PIM. These are largely well known design principles which are simply enforced by the URDAD process. For example, the single responsibility principle improves simplicity and understandability, localizes maintenance around changes to particular responsibility thereby improving modifiability, is a core driver behind reusability and improves cohesion.

5.2 The URDAD process

The URDAD methodology is shown in figure **??**. [1] URDAD interprets a UML use case formally as a requirement for a service. For that service, URDAD requires that first one identifies the stake holders and their requirements specified in the form of a services contract. The services contract is represented by a UML interface containing the service for the use case. The stake holder functional requirements are specified in terms of output (return value), pre- and post-conditions with the pre- and post-conditions formalized as OCL constraints. The quality requirements are specified using the UML quality of service profile. The linkage between the stake holders and their specific requirements is established by inserting dependency relationships from the stake holder to the respective requirements and stereotyping them with the URDAD *requires* stereotype.

Figure 1: The URDAD process.

The first step in the design phase is to identify the functional requirements for the service, i.e. the functions or services required to address the pre- and post-conditions for the service. This introduces the services contracts for the lower level services from which the core service is to be established. For leaf services which are not assembled from lower level services the required logic is fully specified by atomic pre- and post-conditions.

Next one assembles the business process in a UML activity diagram for the higher level service which shows how the process is assembled form the lower level services identified in the previous step. Finally the collaboration context is projected out. The collaboration context shows the services provider contracts with the required services and the required message paths between the service providers.

If any of the lower level services are to be realized within the system, organization or department, then the process is repeated for that lower level service. If the service is to be sourced externally from other systems, departments or external service providers, then one stops after having specified the services contract for that service.

5.3 Structure of an URDAD PIM

[1] There has been a small change from the methodology as it has been originally published in [**?**] in that the contract specification has been shifted forward to represent the core stakeholder requirements specification whilst the functional requirements which is interpreted ad *requirements for functions/services* is done off the stakeholder contract specification.

Table 2: Quality requirements and design principles through which they are realized.

design principle	completeness	consistency	simplicity	modifiability	reusability	testability	traceability	cohesion
single responsibility principle			x	x	x			x
layers of granularity			x	x	x	x		x
decoupling via contracts			x	x	x	x		
add traceability links	x	x		x		x	x	
structure from process	x	x	x				x	
fixed PIM structure	x	x	x			x		
diagrams as views onto model		x	x	x				

An URDAD PIM encourages parsimony by using a minimal subset of UML. Not only is each model element required, by virtue of the parsimony, but each model element also has a precise semantics. An URDAD PIM has the following elements:

- Each required service is specified in a services contract (UML interface) which indicates the inputs and outputs for the service, the pre-and post conditions and the quality requirements specified for the service. The highest level services are the services offered to users.
- Each pre and post-condition is the end-point of a *requires* dependency leading from the stakeholder(s) who require it. The *requires* dependency is defined in the URDAD profile. Stakeholders are to be represented by UML interfaces.
- The functionality used to realize each pre- and and post-condition is specified via a *requires* dependency pointing to the use case representing the functional requirement.
- For each use case, there is a realization relationship from a service in a services contract to the use case.
- Concrete service providers are represented by classes which realize/implement services contracts. (Business) processes in the form of activity or sequence diagrams are required to be attached to the services of concrete service providers. These concrete service providers define technology-neutral processes realizing the services in the services contracts.
- Services either provide the required output or signal that they will not realize the requested service by throwing an exception. Each exception must be linked to a pre-condition for the service via a usage relationship from the pre-condition to the exception class.
- The data structures for all inputs and outputs must be specified.
- User roles are represented by interfaces representing user contracts that accumulate the user's obligations around the services used by that user role. The interaction (sequence of messages exchanged) in the context of a user making use of a service specified in a services contract are specified using a UML sequence diagram.
- The effect of atomic/leaf services (services which are not assembled from lower level services) on the environment is fully specified using pre- and post-conditions.

6. ASSESSING THE QUALITIES OF AN URDAD PIM

In this section we assess the extent to which an URDAD PIM satisfies the stakeholder quality requirements. We thus go through the list of stake holder quality requirements in order to qualitatively assess to what extend and through which mechanisms an URDAD generated PIM realizes them.

6.1 Completeness

Completeness is one of the core quality requirements needed for a model which is to be transformed to code and for to be used for generating tests. It is measured by the extent to which the model contains all required elements. Most authors [**?**, **?**, **?**, **?**] assess completeness by assessing whether all dependencies within a model are defined—that is by assessing whether the model is internally complete. In a similar way [**?**] define a set of requirements as complete if all dependencies within the requirements are defined.

Lange et al. [**?**] point out that, in addition to internal model completeness, one needs to consider completeness from a client's perspective, i.e. the extent to which the client requirements are fulfilled. They note that this aspect of completeness is generally verified through client acceptance tests. They also include under completeness the notions of consistency and well-formedness. In this paper we treat consistency separately, while some aspects of well-formedness are included under the *correctness* quality.

Here we look at four aspects of completeness: completeness from a requirements perspective; internal completeness of both the analysis and the design; completeness from a model usage perspective; and completeness from a meta-model perspective. In each case, we discuss ways in which these can be verified if the PIM has been derived by an URDAD process.

6.1.1 Completeness from a requirements perspective

This is a measure of the extent to which the stakeholder requirements are addressed within a design model. Even though it is true that this aspect of completeness is generally assessed using acceptance testing [**?**], some aspects can be automated within an URDAD validation suite. In particular, URDAD requires that each pre- and post-condition is linked to at least one functional requirement and that the funtional requirement must be realized by a service defined in a services contract. This ensures that each pre- and post-condition is addressed within the URDAD PIM.

6.1.2 Completeness from a model utility perspective

This aspect of completeness refers to the extent to which all model elements required for different model usages like model transformation (including code, test and documentation generation) are provided. [**?**] identifies the PIM elements required for implementation mapping, under the understanding that the *Platform Model* (PM) needs to be provided with the PIM in order to facilitate the model transformation to the *Platform Specific Model* (PSM). It is argued that URDAD provides a sufficient, yet minimal set of model elements from a utility perspective.

In particular, it is argued that one requires:

- the full specification of the inputs and outputs of the business process;
- formal specification (using the *Object Constraint Language*, OCL) of the pre- and post-conditions and quality requirements for each service;
- the business process showing how a service is assembled across lower level services, the decision and merge points in the business process, any concurrency and synchronization points in the business process;
- and recursively, for each lower level service requested in the business process,
  - how the request objects are constructed from the information available at that point in the business process,
  - for lower level composite services the business process showing how the service is assembled across lower level services, as well as how the requests and outputs are constructucted from the available information.
  - for leaf services, how the result is to be constructed from the available information as well as any changes which must have been made to the environment within which the service is executed (in the form of formally specified post-conditions), and
  - for services which are outsourced to external systems/service providers, the complete services contract with inputs, outputs, pre- and post-conditions and quality requirements.

Of course there is a model boundary. In URDAD the model boundary is represented by those contracts for which no implementing service providers (with corresponding process design) have been specified. Such services need to be sourced from the environment as either low level system services, services sourced from existing or off-the-shelf systems which are not part of the model domain or from other external service providers.

6.1.3 Internal completeness of both analysis and design

Internal completeness refers to the extent to which all dependencies of the requirements and design are defined. The normal completeness tests as referred to in [**?**]

6.1.4 Completeness with respect to meta-model

This refers to the extent to which a model has all elements as required by the meta-model for that type of model. Even though an URDAD PIM is in the form of a UML model, it must comply to the URDAD meta-model which specifies a list of required model elements and relationships between them. This aspect of completeness can thus be readily assessed for an URDAD model.

6.2 Consistency

Consistency is an essential PIM quality, particularly needed for transformability and maintainability. Even though a UML model assists with being able to enforce consistency

across diagrams, UML neither enforces consistency nor provides a way to verify model consistency [?]. A range of UML model consistency verification techniques have been proposed [?, ?].

URDAD itself provides a design process which aims to enforce consistency, specifying the required diagram elements and model links between these. These are assessed with the URDAD correctness validation suite. In addition standard UML consistency checks can be applied to an URDAD model.

6.3 Simplicity and understandability

Simplicity is an inverse measure of complexity. It is generally accepted [?, ?] that design simplicity and understandability are improved by hierarchical decomposition of functionality, enforcing the single responsibility principle and decoupling. These three design aspects are enforced within an URDAD PIM. There is, however, a point where the overheads of further decomposition exceed the complexity reduction otherwise obtained from the decomposition[?]. URDAD does not provide any guidance on when a service should be treated as atomic and leaves this to the disgression of the designer.

Alsharif et al. [?] use a modified full function point analysis approach to assess complexity. Their approach finds that complexity can generally be reduced by hierarchical decomposition of functionality. In addition they assess complexity of four design architectures including a shared data architecture which uses globally accessible storage to exchange information between functional components, an abstract data type architecture which groups processes and the data they operate on within abstract data types, an implicit invocation architecture which has a global event space and units of functionality reacting to events on the shared space, and a pipes and filters based architecture which decomposes the functionality into stateless services from which a higher level process is assembled. The latter design architecture which represents the design architecture of an URDAD PIM achieves the lowest complexity value based on responsibility localization, decoupling and stateless services.

One aspect of simplicity is parsomony which is a measure of the extent to which all model elements are required [?]. Even though URDAD claims to provide a minimal, complete set of design elements, this has yet to be formally proven.

6.4 Modifiability

Modifiability has been shown to be strongly related to decoupling [?, ?], model complexity [?], and model consistency. It is also impacted by the level of abstraction of used in the model elements [?], though abstraction and particular deep inheritance hierarchies may increase dependency of model elements and impede modifiability [?].

URDAD enforces:

- decoupling of clients/users from service providers through enforced client-specified services contracts and the expectation of the use of service provider adapters; and
- decoupling of value objects from domain objects (including service request and result objects whose reuse across services is forbidden).

On the other hand, URDAD does not facilitate the use of inheritance except in the case of value objects. This prevents rigidity caused by the excessive use of inheritance hierarchies, but also prevents any modifiability benefits which may be gained through using abstraction in processes. URDAD instead uses composability of processes through service reuse.

6.5 Reusability

Reusability is a measure of the proportion of elements in a design which can be reused and the inverse of the cost associated with the reuse of design elements. [I DON'T UNDERSTAND THE "and the inverse..." PART OF THIS SENTENCE]

In general we want to reuse

- services,
- services contracts,
- value objects, and
- domain entities.

Reuse in URDAD is focused on the reuse of services and services contracts. URDAD enforces modularity and statelessness of services across levels of granularity, the decoupling and formal service specification through services contracts and recursive assembly of services from lower level services, all of which promote service reuse [?, ?].

High level request and result objects are prevented from being reused in URDAD, whilst lower level components of these as well as domain entities are commonly reused, though nothing within URDAD specifically promotes the reuse of either.

Reusability through inheritance [?] is generally not promoted within URDAD and is structurally confined to value object.

6.6 Testability

Since URDAD is based on a design by contract [?] approach, testability revolves around testing all services contracts [?, ?] across levels of granularity. The URDAD completeness tests assess whether there are any services for which there is not a services contract specified.

The enforced levels of granularity reduce model and test complexity at any particular level of granularity [?].

In contract-based design approaches, the generation of tests can be auto-generated from the services contracts [?, ?, ?]. This further enhances testability.

6.7 Traceability

Traceability is measured by the extent to which the impact of requirements changes on the design across various levels can be assessed. It is also measured by the extent to

which any activity performed at any level of granularity can be linked back to a functional requirement of some stake holder. Traceability is usually achieved either (1) within development tools which add the linkage between requirements artifacts and UML model elements [**?**], (2) by adding rationale semantics to a UML model using a separate ontology [**?**] or (3) by extending UML using a profile which adds concepts that illustrate the design decisions [**?**]. URDAD follows the latter path, adding, within the URDAD profile, a `<<requires>>` dependency which is inserted between

- stake holders and pre- and post-conditions, and
- pre-/post-conditions and the functional requirements (which are semantically interpreted as requirements for functions/services).

Additionally URDAD uses a standard UML realization relationship between a functional requirement and a service in a services contract (the services formalizes the requirements around the functional requirement in the form of service inputs and outputs, pre- and post-conditions and quality requirements) as well as an interface realization between the contract and the service providers which realize the services contract through concrete (business) processes.

These relationships facilitate traceability of pre- and post-conditions to functional requirements (specifying the services required to address the pre- and post-conditions), through to services contracts which formalize the requirements for those services through to implementation classes and processes through which the services are realized. From the processes one can navigate through to the lower level services from which the processes are assembled.

Similarly URDAD facilitates traceability in the reverse direction by tracing from any process step to the service within which it is used to the recursively higher level services which make use of those services to a functional requirement and the pre- and post-conditions for which the functionality is required to the stake-holder who requires that pre- or post-condition. The traceability process could also start from a value or domain object leading to the processes within which these are required and then up the route dependency route as discussed in the previous sentence.

6.8 Cohesion

Cohesiveness [**?**] is addressed in URDAD by guiding designers to apply the single responsibility principle. This encourages the outward projection of lower level responsibilities onto lower level services defined in different services contracts. Both of these are manual processes which are neither enforced by the process nor validated through the URDAD validation suite.

7. CONCLUSIONS AND FUTURE WORK

The URDAD methodology enforces some of the accepted design principles which promote model quality. The process provides a simple design algorithm with specified inputs and outputs for each design step. The resultant PIM which has a specified model structure addresses many of the stake holder quality requirements for the PIM. The PIM confines complexity, provides bi-directional traceability and enforces contracts based decoupling, testability and reusability. The defined model structure facilitates testing for completeness and consistency and simplifies model utility like model transformation in order to generate implementation, tests or documention.

Future work includes the completion of the model validation suites, upgrading of documentation generation tools, the development of URDAD centric tools which can improve flexibility and can enforce the URDAD model structure and various transformation tools which can be used to generate platform specific models and code.

7.1 Measurement results and summary

8. CONCLUSION AND OUTLOOK

9. REFERENCES

[1] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: a comparative analysis. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 244–254, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Mohsen AlSharif, Walter P. Bond, and Turky Al-Otaiby. Assessing the complexity of software architecture. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 98–103, New York, NY, USA, 2004. ACM.

[3] D. E. Avison and G. Fitzgerald. *Information System Development: Methodologies, Techniques and Tools.* Blackwell Scientific Publications, Oxford, UK, 1988.

[4] Hakim Belhaouari and Frederic Peschanski. Automated generation of test cases from contract-oriented specifications: A csp-based approach. In *HASE '08: Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium*, pages 219–228, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Grady Booch. Measuring architectural complexity. *IEEE Softw.*, 25(4):14–15, 2008.

[6] Rolv Braek and Geir Melby. Model-driven service engineering. In *Model-Driven Software Development*, pages 385–401. Springer Verlag, 2005.

[7] F. Brooks. No silver bullet: Essence and accidents in software engineering. *IEEE Computer*, 20(4):10–19, 1987.

[8] I. Cardei, M. Fonoage, and R. Shankar. Model based requirements specification and validation for component architectures. In *Systems Conference, 2008 2nd Annual IEEE*, pages 1–8, April 2008.

[9] Si Won Choi and Soo Dong Kim. A quality model for evaluating reusability of services in soa. In *CECANDEEE '08: Proceedings of the 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services*, pages 293–298, Washington, DC, USA, 2008. IEEE Computer Society.

[10] Wang Chu and Depei Qian. A component-oriented development approach to e-business applications. In *ICEBE '08: Proceedings of the 2008 IEEE International Conference on e-Business Engineering*, pages 45–52, Washington, DC, USA, 2008. IEEE Computer Society.

[11] Michael Coram and Shawn Bohner. The impact of agile methods on software project management. In *ECBS '05: Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 363–370, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Steve Counsell, Stephen Swift, and Jason Crampton. The interpretation and utility of three cohesion metrics for object-oriented design. *ACM Trans. Softw. Eng. Methodol.*, 15(2):123–149, 2006.

[13] Jeremy Dick. Design traceability. *Software, IEEE*, 22(6):14–16, Nov.-Dec. 2005.

[14] George Feuerlicht and Amalka Wijayaweera. Determinants of service reusability. In *Proceeding of the 2007 conference on New Trends in Software Methodologies, Tools and Techniques*, pages 467–474, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[15] David S. Frankel. *Model Driven Architecture: Applying MDA to enterprise computing.* John Wiley & Sons, New York, 2003.

[16] Marcela Genero, Mario Piatini, and Esperanza Manso. Finding "early" indicators of uml class diagrams understandability and modifiability. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 207–216, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Tauqeer Hussain, Mian M. Awais, and Shafay Shamail. Applying fuzzy logic to measure completeness of a conceptual model. *Applied Mathematics and Computation*, 185(2):1078–1086, 2007. Special Issue on Intelligent Computing Theory and Methodology.

[18] C.F.J. Lange and M.R.V. Chaudron. Managing model quality in uml-based software development. In *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*, pages 7–16, 0-0 2005.

[19] Christian F.J. Lange. Improving the quality of uml models in practice. *ICSE'06*, ??(??):993–996, 2006.

[20] Christian F.J. Lange and Michel R.V. Chaudron. An empirical assessment of completeness in uml design. In *Proceedings of EASE – International Conference on Empirical Assessment in Software Engineering*, pages 111–121. nstitution of Electrical Engineers (IEE) Press, 2004.

[21] Radu Marinescu. A multi-layered system of metrics for the measurement of reuse by inheritance. In *TOOLS '99: Proceedings of the 31st International Conference on Technology of Object-Oriented Language and Systems*, page 146, Washington, DC, USA, 1999. IEEE Computer Society.

[22] Bertrand Meyer. Applying design by contract. *Computer (IEEE)*, 25(10):40–51, 1992.

[23] Parastoo Mohagheghi and Jan Aagedal. Evaluating quality in model-driven engineering. In *MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering*, page 6, Washington, DC, USA, 2007. IEEE Computer Society.

[24] Parastoo Mohagheghi and Vegard Dehlen. An overview of quality frameworks in model-driven engineering and observations on transformation quality. 2008.

[25] Clementine Nebut and Franck Fleurey. Automatic test generation: A use case driven approach. *IEEE Trans. Softw. Eng.*, 32(3):140–155, 2006. Member-Le Traon, Yves and Member-Jezequel, Jean-Marc.

[26] Clémentine Nebut, Franck Fleurey, Yves Le Traon, and Jean-Marc Jézéquel. Requirements by contracts allow automated system testing. In *ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering*, page 85, Washington, DC, USA, 2003. IEEE Computer Society.

[27] Rodrigo Perozzo Noll and Marcelo Blois Ribeiro. Enhancing traceability using ontologies. In *SAC '07:*

Proceedings of the 2007 ACM symposium on Applied computing, pages 1496–1497, New York, NY, USA, 2007. ACM.

[28] Geert Poels and Guido Dedene. Evaluating the effect of inheritance on the modifiability of object-oriented business domain models. In *CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, page 20, Washington, DC, USA, 2001. IEEE Computer Society.

[29] L. Reynoso, M. Genero, M. Piattini, and E. Manso. Assessing the impact of coupling on the understandability and modifiability of ocl expressions within uml/ocl combined models. In *Software Metrics, 2005. 11th IEEE International Symposium*, pages 10 pp.–14, Sept. 2005.

[30] L. Reynoso, M. Genero, M. Piattini, and E. Manso. The effect of coupling on understanding and modifying ocl expressions: An experimental analysis. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 4(2):130–135, April 2006.

[31] Douglas C. Schmidt. Model driven engineering. *IEEE Computer*, 39(2):25–31, February 2006.

[32] Y. Shinkawa. Inter-model consistency in uml based on cpn formalism. In *Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific*, pages 411–418, Dec. 2006.

[33] Jon Siegel. Developing in OMG's Model-Driven Architecture. White paper, Object Management Group, November 2001.

[34] Gregor Snelting. Paul feyerabend und die softwaretechnologie (zur diskussion gestellt). *Informatik Spektrum*, 21(5):273–276, 1998.

[35] Fritz Solms. Technology neutral business process design using urdad. In *Proceeding of the 2007 conference on New Trends in Software Methodologies, Tools and Techniques*, pages 52–70, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[36] Fritz Solms and Dawid Loubser. Generating mda's platform independent model using urdad. *Knowledge-Based Systems*, 22:174–185, 2009.

[37] M. Usman, A. Nadeem, Tai hoon Kim, and Eun suk Cho. A survey of consistency checking techniques for uml models. In *Advanced Software Engineering and Its Applications, 2008. ASEA 2008*, pages 57–62, Dec. 2008.

[38] Jan Verelst. The influence of the level of abstraction on the evolvability of conceptual models of information systems. *Empirical Softw. Engg.*, 10(4):467–494, 2005.

[39] Weiqun Zheng and G. Bundell. Test by contract for uml-based software component testing. In *Computer Science and its Applications, 2008. CSA '08. International Symposium on*, pages 377–382, Oct. 2008.

[40] Liming Zhu and Ian Gorton. Uml profiles for design decisions and non-functional requirements. In *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, page 41, Washington, DC, USA, 2007. IEEE Computer Society.

[41] Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993–1009, 2003. Eighth International Workshop on Requirements Engineering: Foundation for Software Quality.