User Requirements for a Functional Design of an Enterprise Architecture Model Repository

Ruud Oude Maatman
ruud@oudemaatman.com

ABSTRACT
Many enterprises document their activities, business processes, information systems and technologies in the form of models. Together these models form Enterprise Architecture (EA). There are different levels of EA maturity according to recent surveys. In this paper it is argued that the requirements for a repository for the storage of the EA are dependent on the EA maturity. The conducted survey shows that the requirements for an Architecture Model Repository show differences between the EA maturity phases. For a combination of phases a functional design is proposed.

Keywords
Enterprise architecture, EA maturity, architecture model, EA repository, functional design.

1. INTRODUCTION
Many enterprises document their activities, business processes, information systems and technology infrastructure in the form of models [1]. All these models form the so-called enterprise architecture (EA). Over time, these models may change and new models are created. Furthermore the number of these models can be quite large, especially in large information intensive organizations (e.g., Belastingdienst or banks), in which the number of business processes are in the order of thousands. The question that emerges is how these models can be kept consistent, managed and maintained throughout the entire company? A possible solution is to use a model repository to store and manage all the architecture models. The main issue we address in this research is to specify the functional requirements for such a model repository and to propose a functional design. A literature review has been presented in [2], which also serves as the starting point of this paper. However the functional design proposed in [2] is the result of a desk research exercise. More research is needed in order to improve this design by correlating it with end-user requirements.

The structure of the paper is as follows. In section 2 we will describe what an architecture model repository (AMR) is and what functions it has. In section 3 the research is described, including the problem statement, research questions and research method. In section 4 enterprise architecture maturity is discussed, which serves together with section 2 as a starting point for the survey in section 5. The survey results will be used to create a functional design in section 6. Finally, the research will be concluded and discussed in section 7.

2. ARCHITECTURE MODEL REPOSITORY
Users may want to represent, deliver and reuse EA. This can require other modeling techniques than the techniques the users normally use. This can require methods in which they are not experienced, as they normally use project dependent modeling techniques. Typically, the required management criteria, business processes and information systems are represented by different modeling methods. This is where an Architecture Model Repository (AMR) plays an important role. Therefore, we will explain what an architecture model repository is (section 2.1) and what functions it should cover (section 2.2).

2.1 What is an architecture model repository?
A database which is used for EA needs to handle complex EA objects. According to [3], such systems often benefit from a layer which isolates the EA object from the details of the database access code. A repository (which uses the isolation layer) can be seen as the mediator between the data in the database and the objects, this is done by mapping. Often, a collection-like interface is used to access the EA objects. Client objects construct query specifications declaratively and submit them to the repository for satisfaction. Objects can be added and removed from the repository, as they can from a simple collection of objects, and the mapping code encapsulated by the repository will carry out the appropriate operations behind the scenes. Conceptually, a repository encapsulates the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer. [3]

According to [4], a repository is a general tool for knowledge management. It can include information about data, processes, physical hardware, human, and physical resources. It is also stated that a so-called Enterprise Model Repository (EMR), is one of the best tools for knowledge management. It can represent, deliver, reuse and recreate a variety of knowledge representation. Thereby creating possibilities to compare, transfer and merge knowledge represented by different models.

An architecture model repository (AMR) can thus be seen as a special type of content management system (CMS). The system can be used to save, browse, search and view different types of architecture models (e.g. business process models, UML diagrams, architect models etc.), but also other types of documents or artefacts (e.g., PowerPoint presentations).

2.2 What are the functions of an architecture model repository
An AMR can contain all types of data, but its main use is for the storage of architecture related artefacts. This means that the main purpose of the AMR is to store models and, optionally, to facilitate relating stored models to each other and to other documentation and multimedia.

The repository must be able to keep track of these artefacts and their relations. Additionally it needs to have the capability of
showing the traceability between models. The AMR has among others the create, read, update and delete (CRUD) functions on the artefacts and their relations [4].

Another important function of the AMR is configuration management [5] which deals with the management of the different versions, revisions and releases of models and groups of models. The access and use of an AMR must be controlled by the user management functionality that facilitates the definition of user roles, profiles and access rights. Furthermore, the users of an AMR must be supported in browsing and searching the content of the AMR.

An important support function of the AMR consists of the scanning, searching, reporting and transferring [4] of artefacts. The AMR’s search functionality may be based on, for instance, some domain ontology, metadata and/or semantic matching. The AMR (content) can be visualized through a web-based portal application endowed with model viewers, and additional business intelligence and/or reporting tools.

3. RESEARCH
In this section it is described how the research is conducted. The problem will first be stated (section 3.1), from which the main research question will be derived. With the problem stated we define the main research question (section 3.2) and divide it in smaller research questions. The research method is stated in section 3.3.

3.1 Problem statement
One of the most important conclusions in [2] is that it is highly probable that the needs an organization might formulate, with respect to the functionality of an EA repository, very much depends on the maturity of its enterprise architecture (EA). We argue that there must be a fit between the repository and the EA maturity phase, for the repository to serve the organizational goals optimally. At the moment, there only is theoretical information available on the subject. The problem we therefore address in this research is: EA repository design has not been aligned with EA maturity phases. Therefore the main question is: How can we align the functional design of a repository with the EA maturity phases and create a flexible modular design that can be adjusted for each phase? To be able to collect the specific user requirements a number of companies that find themselves in different EA maturity phases will be surveyed. The results of the survey, together with the theory, will be used to create a functional modular design for an AMR.

3.2 Research questions
We refine the main research question (as stated above) by dividing it in smaller research questions.

What are the most important issues for the repository with respect to model maintenance and change? (section 2)

What theoretical requirements are there for a repository, in particular for each EA maturity phase? (section 4)

What are the concerns of stakeholders with respect to the usage of a repository? (section 5)

How can functional features/capabilities be aligned with an organization’s EA maturity phase? (section 6)

3.3 Research method
In order to answer the research questions (see section 3.2), we will carry out a literature survey with respect to enterprise architecture model repositories and EA maturity models [2]. The collected information will form the base for the design of a structured survey protocol. The survey is used as an instrument for collecting user requirements and will mainly concern the issues that stakeholders have in practice when using an AMR.

Finally, the user requirements will be combined with existing theoretical results in order to create a maturity-aligned design. The results of this research will be used as input for the implementation of a proof-of-concept prototype for an AMR.

4. ENTERPRISE ARCHITECTURE MATURITY
In this section we will address the enterprise architecture maturity. Therefore we first give an introduction to Enterprise Architecture Maturity (section 4.1). After the introduction we will explain the 5 maturity phases as used in this research (sections 4.2-4.6). For each of these phases the theoretical requirements for an AMR will be stated. In section 4.7 we will shortly describe the maturity phase in which currently the most EA mature organizations can currently be placed in. Finally, we end this section with a conclusion in section 4.8.

4.1 Introduction
Gartner states in [6] and [7] that the degree of maturity of an architecture program has a profound influence on its effectiveness. Understanding and assessing the architecture maturity phase can be used to set achievable goals, to plan and control the architecture program and to progressively improve it and remove barriers to its success. So what is architecture maturity and how does it fit in the context of this document?

In addition to the EA documentation, models, definitions, principles, and other deliverables that make up the EA, it is important to assess the EA Program and the other aspects of how the EA is being utilized within strategic governance and decision making. Generally speaking, an EA-mature organization is one that uses EA in multiple ways, from IT and decision making to organizational realignments business redesign and transformation. However, before full maturity is reached, EA can be phased over time. The maturity model we use spans over five phases and is based on [8], [9], [10], [11], [12], [13], [14] and [7] that correspond to the different maturity phases of an EA and eventually addresses the effectiveness of the architecture process.

4.2 Phase 0 – No architecture
The sole existence of an enterprise places it in this phase. The organization does not have plans to develop and use a service oriented enterprise architecture (SOEA). There is no architectural framework in place at this phase of maturity. Consequently, there exist neither an architecture process nor architecture tools.

As there is no architecture in this phase, there are no requirements for an AMR.

4.3 Phase 1 – Informal architecture program
Activities related to the development and documentation of the architecture deliverables have an ad-hoc character and are performed informally and unstructured. Organizations with enterprise architecture at this phase are still dependent on the knowledge of individuals, although a part of the architecture is becoming explicit.

There are no formal architecture governing structures. However, the top management and especially the CIO show interest in setting up a formal architecture program.

EA awareness efforts are informal and inconsistent. The use of dedicated architecture tools in this phase is extremely limited.

The projects are small and localized. The existing deliverables are stored in libraries, folder structures or, at the best, in a
document management system or a data warehouse. From this follows that there are hardly any requirements for an AMR.

4.4 Phase 2 – Supervised architecture program
The benefits of architecture are recognized and the need to invest in and to commit to architecture is accepted. The architecture framework, which is documented, is now adapted to the need for a unified architecture process across technologies and lines of business.

A need for Architecture Governance has been identified. In this phase the organization is going to decide upon a framework and a methodology and begins to develop a plan for their EA Program.

Critical EA information has priority. Up to this phase, the different domains (e.g. organizational structure, business processes etc.) are not approached in an integrated way. Every domain speaks its own language, draws its own models (if any), and uses its own techniques and tools (if any).

An AMR needs to be capable of storage, dissemination and sharing of captured EA information. The AMR has to be able to work with an integrated architecture support tools that cover several EA-aspects in different domains. Functions it has to offer are: design, analysis and visualization of business processes, organizational structure and information flow. Furthermore the AMR has to be capable of working with decision making and management tools. At the end of this phase an AMR consisting of these elements will be selected.

4.5 Phase 3 – Governed architecture program
The enterprise architecture framework is well defined. The usage of approved standard and/or templates is common practice. Most business processes are documented. The progress of the architecture process is evaluated. The architecture framework have been defined and documented. The scope of the architecture has been broadened to encompass the entire enterprise in its current state.

The organization is tracking and measuring its progress against plans, identifying and addressing variances and then reporting on its progress. The focus is on developing architecture deliverables according to the selected framework and methodology, tools, and established management plans.

An AMR needs to support the common practice in this phase, therefore the following requirements are necessary:

- Visual, Concurrent, Distributed, Collaborative or off-line Modeling
- Visualization of EA deliverables
- Reporting, publication (on the intranet), reuse (including import and export) of models
- Analysis, simulation, transformation and execution of models
- Storage and version management EA deliverables
- Documenting EA deliverables
- EA planning: improvement of the adopted architecture framework and method, EA policies and management.

4.6 Phase 4 – Iterative and incremental architecture program
The organization has completed the description of the current state of the EA. The completed deliverables collectively describe the enterprise in terms of business, performance, information/data, service/application, and technology architecture.

The usage of enterprise architecture tools and repository has become common practice. The focus for the usage is on the continuous maintenance, update, change and, most important, usage of these deliverables. From now on any design activity will focus on the future (“to-be”) situation and the migration from the current to the desired situation (the migration plan).

Regarding the theoretical requirements in this maturity phase the focus is on the continuous maintenance, update, change and usage of EA deliverables. Therefore no additional functions are needed, besides those stated in phase 5 (section 4.5).

4.7 Current positioning of enterprises
According to recent surveys ([15], [1]) the most advanced enterprises can be placed somewhere in what in our maturity model is described as phase 3.

4.8 Conclusion
For maturity phases 0 and 1 no repository is needed, as the EA knowledge is mainly in the minds of the employees. A normal database would be sufficient for those phases. In maturity phase 2 the need for an AMR becomes distinct. EA deliverables need to be stored in a unified manner and need to be analyzed which must be supported by the repository. Maturity phases 3 and 4 practically have the same requirements for an AMR as the main difference between the phases is the time focus: the current (phase 3) versus the to-be (phase 4). Therefore for a modular AMR design these phases can be combined.

5. REQUIREMENTS FOR REPOSITORY SUPPORT
In this section we will address the requirements for repository support. To collect user requirements with respect to the usage of an AMR we conduct a survey (Appendix A) with architects from different enterprises. The participants will be presented in section 5.1, and will be classified according to their architecture maturity. In section 5.2 we will describe the survey protocol. The survey data will be analyzed in section 5.3. Lastly, we will show the research results (section 5.4) in which we will show the requirements for a maturity phase dependent AMR.

5.1 Respondents
The survey has been sent to several architecture stakeholders from the following organizations:

- ING Investment Management (Phase 2)
- Agis (Phase 2/3)
- Reaal (Phase 2/3)
- Atos Origin (Phase 3/4)
- Logica CMG (Phase 3/4)
- IFEAD (Phase 4)

The maturity classification of these organizations is based on experiences and information from previous projects with these organizations. Agis and Reaal are positioned between phase 2 and 3 because the answers provided by these companies were not compliant (the requirements were more advanced) with the expected answers and the answers provided by ING Investment management.

5.2 Survey Protocol
In this section we will give a summary of the survey, the survey itself can be found in Appendix B. As stated in section 2.1, a repository can be seen as a special type of content management system (CMS). Therefore the subjects of the survey are derived from the core component which a CMS encompasses. According to [16] some core components of enterprise content management (ECM) (which includes CMS) are:
• Document management for check-in/check-out, version control, security and library services for business documents.
• Document capture and document imaging for capturing and managing paper documents.
• Document-centric collaboration for document sharing and supporting project teams.
• Workflow for supporting business processes and routing content, assigning work tasks and states, and creating audit trails.
• Ease of integration with a variety of client applications.

5.2.1 Models
Regard the models we would like to investigate whether the repository needs to allow different modelling languages and storage of the metamodels of these languages and the storage of other artefacts. Furthermore we would like to research if the repository must allow the editing of models, the definition of relations between models and the definition of relations between models and other artefacts.

5.2.2 Locking and configuration management
With regard to locking we would like to know whether the AMR needs to have an explicit locking mechanism, a model check-in/check-out. With regard to configuration management we are interested in finding out whether the AMR needs to have version management functionality and needs to support control of changes and support for revisions/releases.

5.2.3 Traceability
With regard to traceability we would like to know whether when an object is reused or referred to, the object may not be deleted as is. The co-occurrences and references to it should be deleted too.

5.2.4 Navigation, Search and Filtering
With regard to navigation, search and filtering we would like to know whether the AMR needs to support this by name, object type, associations/relations and attributes. Furthermore we would like to know whether the AMR needs to have configurable navigation toolbars and graphical model browsing.

5.2.5 Model visualization
With regard to model visualization we would like to know whether the AMR must allow configurable/customizable, zooming in/out models, viewpoint-based, role-based, rights-based visualization. Moreover we would like to know whether the AMR must allow visualization of models based on custom defined filters.

5.2.6 Monitoring, analysis, reporting and publication
With regard to monitoring, analysis and reporting we would like to know whether the AMR needs to include monitoring and reporting functionality and analytical tools. With regard to publication we would like to know whether the publication/export of models must is a requirement.

5.2.7 User and access rights management
With regard to user management we would like to know whether the AMR must support role-based user management. With regard to access rights management we would like to know if the AMR must support role-based and user-based access rights and the definition of access rights at model level and model element level. Additionally we would like to know whether the AMR must support provision of feedback.

5.2.8 Integration
With regard to integration we would like to know whether the AMR must allow integration with other applications and tools, single sign-on and the definition of access rights integration with LDAP.

5.3 Data analysis
5.3.1 Models
For enterprises in maturity phases up to phase 2/3 the storage of models with different modeling languages is a function they would like to see on the long term. While for maturity phases 3 and 4 the storage of models with different modeling languages the organizations think this is nice to have on the short term.

The functionality of the metamodels of the required languages is nice to have on the long term for maturity phases up to 2 and nice to have on the short term for maturity phases 2/3 and above.

The function of editing models is considered to be important for maturity phases up to 3. Organizations in maturity phase 4 deem this to be critical for an AMR.

The possibility to store other artefacts than models is considered to be important (now) for organizations up to maturity phase 2/3. Maturity phases 3 and 4 organizations consider this option to be critical for an AMR, they would even like full functionality with these artefacts (e.g. creating new, defining relations, change attributes).

The most required artefacts are:
• Text (e.g. Word, Pfd)
• Presentations (e.g. Powerpoint)
• Diagrams/models (e.g. Visio)

The definition of relations between models is equally important for organizations in all maturity phases, they unanimously think this is important (now) for an AMR. The most required type of relation is a cross-model reference, although model hierarchy and nesting are also considered to be important. The definition of relations between models and other artefacts seems to be of far more importance for maturity phase 3 and above. Up to maturity phase 2 organizations think this nice to have on the long term, while organizations in maturity phases 2/3 and above think this important (now) for an AMR. These organizations would like to see the possibility to link all common used file-types, whereas the lower maturity phases (up to 2) would like to see linkage to documents (text).

5.3.2 Locking management
An explicit locking mechanism for objects while having concurrent access to models, is equally important for organizations on all maturity phases, which is important. Most of the organizations would like to see an object-based locking (phase 4 organizations would additionally like to see notifications indicating that the model is locked to the users that access an artefact which uses the model in progress). Organizations on all maturity phases think it is important (now) to do this with a model check in/out, which is used to explicitly lock (specific parts of) the model which is worked on.

5.3.3 Configuration Management
The AMR function for version management is considered to be important for maturity phases 2/3 and above, although Logica CMG thinks this will be important in the future. The version management function should work at model level. The control and recording of changes function is nice to have on the short term for organizations on all maturity phases. Differences between maturity phases are hard to define. The support for revisions/releases of collections of models is considered equally important for all maturity phases, the organizations think this is nice to have on the short term.
5.3.4 Traceability
When reused or referred to, an object may not be deleted as is: co-occurrences and references to it should be deleted too. This means the repository should enforce referential integrity between artefacts. As a result, the repository should be aware of relations between models and model artefacts. This traceability function is important (now) for organizations up to maturity phase 2 and critical maturity phases 2/3 and above.

5.3.5 Navigation, Search and Filtering
Navigation, search and filtering by object type, name or attributes is equally important for organizations on all maturity phases. These functions are considered to be important (now), where as the name search is the most critical. The navigation, search and filtering on associations/relations is important to have now for organizations in maturity phases 2/3 and above. For organizations in maturity phases up to 2 this is nice to have on the short term.

Configurable toolbars are nice to have on the short term for organizations in maturity phases up to 2/3. For organizations in maturity phases 3 and above it is important to have this functionality now. Graphical model browsing is considered nice to have on the short term for organizations in maturity phases up to 2, while maturity phases 2/3 and above consider this to be important (now).

5.3.6 Model visualization
Configurable/ customizable visualization is nice to have on the long term for organizations in maturity phases up to 2, nice to have on the short term for maturity phase 3 and important (now) for maturity phase 3. The zooming in and out of models is nice to have on the long term for maturity phases up to 2, and important for maturity phases 3 and 4.

A viewpoint based visualization of the models is nice to have on the long term for organizations in maturity phases up to 2, important (now) for maturity phases 2/3 and 3 and critical for maturity phase 4. A role based or a rights-based visualization of the models is nice to have on the long term for organizations in maturity phases up to 2, and nice to have on the short term for maturity phases 2/3 and above.

A custom defined filter for visualization of models is nice to have on the long term for organizations in maturity phases up to 2, and nice to have on the short term for maturity phases 2/3 and above.

5.3.7 Monitoring, analysis, reporting and publication
The monitoring and reporting functionality of the AMR is nice to have on the short term for organizations in maturity phases up to 2, important (now) for maturity phases 2/3, and critical for maturity phase 3 and 4. The analytical function of the repository (e.g., consistency checks, referential integrity mechanisms, impact of change analysis, model comparison and gap analysis) is nice to have on the short term for organizations in maturity phases up to 2 and important (now) and sometimes even critical for maturity phases 2/3 and above.

A publication or export function is nice to have on the long term for maturity phases up to 1/2, but critical for maturity phases 2/3 and above.

5.3.8 User & access rights management
Role based user management is equally important for organizations on all maturity phases, the organizations think this is important to have now. Role-based access rights management is considered to be nice to have on the short term for organizations in maturity phases up to 2 and important (now) for maturity phases 2/3 and above. User based access rights management is considered to be less important, most organizations would like to see that on the short term and no difference in maturity phases can be determined.

The definition of access rights at model level is nice to have on the short term up to organizations in maturity phase 2 and important (now) for maturity phases 2/3 and above, while the definition of access rights at model element level is considered not important up to maturity phase 2 and nice to have on the short term for phases 2/3 and above.

A feedback providing function is considered equally important for all maturity phases, the majority of organizations think this nice to have on the short term.

5.3.9 Integration
The integration of other applications and tools is considered equally important for organizations on all maturity phases, most organizations think this in nice to have on the short term. Single sign-on is nice to have on the long term for organizations in maturity phases up to 2, and nice to have on the short term for maturity phases 2/3 and above. The definition of access rights integrated with LDAP is considered to be nice on the long term for all maturity phases.

5.4 Survey results
The results of the survey have been summarized in Appendix A. If a function has got a “nice to have on the long term”, or lower classification for the respective phase it is not included in the design of the repository.

The results show that EA maturity can be aligned with AMR requirements. The classification of the AMR requirements is mainly based on the enterprise classification in section 5.1. However, some adjustments have been made. As can be seen in section 4, up to EA maturity phase 2 hardly any requirements for an AMR are needed. Therefore we combined these phases in one modular part for an AMR. An additional modular part has been made between maturity phase 2 and 3 to cover the major difference in requirements between maturity phases up to 2 and phases 3 and up. Maturity phase 3 and 4 have been combined in one modular design part, as there are only minor differences in requirements between these phases. This result confirms the conclusion formed in section 4.8 which states that phase 3 and 4 can be combined with regard to AMR requirements. Although we defined 5 maturity phases in section 4, only 3 phases are used in the modular design.

One additional detail can be seen in the results of the survey. The main difference between the EA maturity phases are formed in the use of the repository (visualization, navigation, analysis). The data layer of the repository only shows minor differences in requirements.

6. REPOSITORY DESIGN
In this section we describe the design of a modular AMR which is aligned with EA Maturity. In order to allow other applications to work with the models stored in the repository we have included a Repository Interface in this architecture. Besides being able to access data (i.e., models) in the repository, the Portal will also use and store its own portal specific data in the repository (e.g., webpage style sheets). Figure 1 depicts an informal way the layered architecture of the repository and the surrounding applications. Three layers can be distinguished:

- a data layer consisting of the repository itself and the login information storage
- an application layer (e.g., portal, (meta)modelling tools, analysis/reporting tools, transformation engines, workflow execution engines etc.) consisting of
applications that can access, extract and change data from/in the repository

- an interface layer between the two above mentioned layers.

In this section we will mainly focus on the data layer (section 6.1) of the repository, as this is of more importance for the design then the application and interface layer. The survey showed that the portal differs in the different phases. Therefore we also describe the functionality of the portal (section 6.2).

6.1 Data layer

In this section we will discuss the storage of data (see the bottom layer in Figure 1). This architecture follows the principles stated in [17]. Login data is used for the authentication of portal users. It includes the user names and passwords. We have decided to keep this data apart from the actual model repository, since it is not concretely specified in advance where this data will come from.

The most significant portion of the data stored in the repository consists of models. These models will be each stored in the data base as XML. These XML data can be read and handled by the repository interface.

6.1.1 Artefacts tree containers and types

Generally speaking, the smallest storage unit in the repository is the artefact. We distinguish between two types of artefacts: models and documents. A model is the product of some modeling tool. A document is any other piece of information which is not a model. Any operation in the repository will be defined on an artefact. It is also possible to define relations between artefacts (see Section 6.1.4). The definition of a modeling language is given in a so called metamodel which defines all the modeling concepts of that language and all the possible relationships between the concepts.

Models are made of instances of modeling concepts and modeling relations. These are represented in our repository model by the concept of ModelElement (see Figure 2). Each model element has (i.e., is an instance of) exactly one ModelElementType (Figure 2).

The models need to have an explicit locking mechanism, which can partially guarantee the integrity of the information.

With regard to maturity alignment we can say that only organizations in maturity phase 2/3 and above need the storage of fully functional metamodels in the repository. As the storage of metamodels itself in the repository is of great importance for the functioning of the repository [18], [19], [20] the normal metamodels will always be stored in the repository. The fully functional metamodels however will be stored in the repository for maturity phases 2/3 and above.

![Figure 1 Repository Architecture](image1.png)

**Figure 1 Repository Architecture**

6.1.2 Release and configuration management

An important role in the design of the repository will be played by configuration management (see [5]). In our repository model we distinguish between three main configuration management concepts: ArtefactVersions, Revisions and Releases (Figure 3). From each artefact several different versions may be stored and co-exist simultaneously in the repository. Revisions and releases can be regarded as configurations (Figure 3). A new revision is created when the current revision is changed (i.e., new versions of one or more artefacts have been created). A release is a revision that is made public (i.e., access to this release is granted to all entitled user groups, i.e., users that had access to the previous release) and which comes possibly after several consecutive revisions.

![Figure 2 Artefacts and containers](image2.png)

**Figure 2 Artefacts and containers**

![Figure 3 Versions, releases and revisions](image3.png)

**Figure 3 Versions, releases and revisions**

Releases have a ReleaseState. Examples of such states are: PAST, CURRENT, IN DEVELOPMENT and TARGET. The states are stored in the repository and each release must have such a state associated. Moreover, two consecutive releases must have different states (unless they are both past releases). For each release state several ArtefactVersionStates are defined. For example, for the release state “in development”, artefact states such as ‘Concept’, ‘Draft’, ‘Final’, ‘Production’, etc. may be defined. The artefacts that belong to a certain release may only have artefact states that are allowed for the ReleaseState of that particular release.
With regard to maturity alignment no distinction is made in the functional design.

6.1.3 Users
This section describes the concepts and functionality concerning the handling of users. The login information, such as passwords, is not stored in the repository, as we assume the authentication process takes place outside the repository. Each user has a UserProfile, which consists of one or more roles. After logging in, a user must specify in which of the roles included in his profile he will use the repository. A role may contain the attributes Name and Access rights specification (determines the Create, Read, Update, Delete and Publish operations).

With regard to maturity alignment, maturity phases 2/3 and above will need single sign on to be used for the complete repository and the desired applications. Furthermore, the definition of access rights will be stored in the artefact for maturity phases 2/3 and above.

6.1.4 Relations
This section describes the relations that can be defined between the Artefacts (or between ArtefactVersions) that are stored in the repository. The Artefacts (or ArtefactVersions), together with their relations, form a graph. The Artefacts (ArtefactVersions) are in this case the nodes of this graph while the relationships constitute the edges.

Examples of relations in the AMR are (for maturity phases up to 2):
- Between two models A and B: A ‘is a variant of’ B.
- Between a model A and a ModelElement a from another model: A ‘is a refinement of’ a.
- Between a ModelElement a from one model A and a ModelElement b from another model B: a ‘is mapped on’ b (mapping is an example of 1-to-1 model transformation).

For maturity phases 2/3 and above
- Between a model A and a document B: A ‘is documented in’ B.
- Between a ModelElement a from a model A and a document B: a ‘is explained in’ B.
- Between two documents A and B: A ‘is an appendix of’ B.

The elements of a relation can be seen in Figure 5. It is of great importance that the repository maintains the referential integrity.

With regard to maturity alignment we see a difference in the relations between ArtefactTypes. For maturity phases up to 2 only relations between models are required. For the remaining phases relations between all ArtefactTypes are necessary. Furthermore from maturity phases 2/3 and up analytical tools are required. Examples hereof are: impact of change analysis, model comparison and gap analysis.

6.2 Portal functionality
This section describes the functionality that should be made available via a portal built on top of and interacting with the repository. The portal is eventually a web interface for the repository that facilitates the administration of and the access to artefacts stored in the repository. According to the survey (section 5.3) this is the part of the AMR which has the largest maturity impact.

6.2.1 Login and access
A user must choose between the roles that are assigned to him, in order to be able to use the portal and, eventually, access the repository, which will be done by single sign on for maturity phases 2/3 and above. The role will (section 6.1.3) impose limitations regarding the usage of and access to the repository. After logging in, a user can use the functions of the AMR, when
one those functions accesses a model the portal will check-in to the model to make sure no integrity errors occur.

6.2.2 Navigation, search and filtering
The navigation, search and filtering is of great importance for all maturity phases. A greater usability results in more customer satisfaction. Therefore the navigation, search and filtering is required to be capable by object type, name, attributes and associations/relations. For maturity phases 2/3 and above the browsing of the models is graphical. And for maturity phase 4 a customizable navigation toolbar is added.

6.2.3 Visualization
For maturity phases 2/3 and above the chosen role determines the manner in which models from the repository are presented. For maturity phases up to 2 no difference in the visualization of models is required. This means that for maturity phases up to 2 the models will always be presented in the same way. For maturity phases 2/3 and above the presentation of the models is dependent of the role, e.g. an architect will see more details of the model standard, while a manager will see more of an overview. Users will therefore only see the model in a way that is most suited for them.

For maturity phases 2/3 and above a viewpoint based visualization is added. A viewpoint describes how a view should be made in terms of content (e.g., concepts and relation types), models, analysis and visualization techniques, relations with stakeholders, etc. For maturity phase 4 a zoom in/out function of models and configurable/ customizable visualization of model artefacts are added.

6.2.4 Monitoring and reporting
For organizations in maturity levels 2/3 and above reporting is included. This includes the procedures concerning the extraction, interpretation and analysis of data and meta-data about the stored model artefacts and may occur regularly (as result of monitoring) or on demand.

Furthermore the publication of models and reports in or to other formats is added to the repository for maturity phases 2/3 and above. This also holds for the export of these models and reports.

7. CONCLUSION AND DISCUSSION
7.1 Conclusion
The problem we addressed in this research was EA repository design has not been aligned with EA maturity phases. The theoretical research of EA maturity showed that different functions were needed for the different maturity phases. The results of the survey confirmed the fact that different functions were needed for different EA maturity phases. Therefore a classification according to the results can be made. The classification led to a combination of maturity phases 0, 1 and 2, maturity phase 2/3 and a combination of maturity phase 3 and 4. Additional functions are required for each increase of EA maturity. The fact that it is possible to create this classification according to EA maturity leads to the main conclusion of this research: EA design can and must be aligned with EA maturity.

The functional modular design of the AMR shows us that most differences between the maturity phases lie in the usability of the AMR and its content. Therefore the most maturity dependent modular part of the design is the Portal. There are however differences in the data layer of the repository design. These differences mainly focus on the storage of, the access to and the relations between artefacts.

7.2 Discussion
The survey which is conducted is in compliance with the research method. However, to make the outcome of the research more reliable it an ad an additional survey (with interviews) could be carried out.

Another point of discussion is the classification of the modular design. It is possible to create only 2 designs in the modular design. The first design would suit the combination of EA maturity phases 0, 1 and 2. This would resemble the combination of phase 0 ,1 and 2 with phase 2/3. Due to the major difference in the survey outcome between phase 2 and phase 2/3 this is considered to be inferior.

ACKNOWLEDGEMENTS
I would like to thank everyone who reviewed and commented on this paper. I would also like to thank the people of BiZZdesign for the help with the survey respondents. In special I would like to thank my supervisor Dr. Maria-Eugenia Iacob who provided me with case material and guided me during the research.

The work presented in this paper is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under con-tract BSIK 03025.

REFERENCES
APPENDIX A: MATURITY DEPENDANT FUNCTIONS

Numbers previous to the functions are question numbers in the survey

(1)= critical
(2)= important
(3)= nice to have on the short term

Figure 6 Functions vs Maturity
APPENDIX B: INTERVIEW PROTOCOL
For each of the questions the following answers could be given:

<table>
<thead>
<tr>
<th>Not important</th>
<th>Nice to have on long term</th>
<th>Nice to have on short term</th>
<th>Important (now)</th>
<th>Critical (Must have)</th>
</tr>
</thead>
</table>

Models
1. The repository must allow the storage of models designed using different modelling languages/notations. Please indicate which languages are relevant for you.
2. The metamodels of the supported languages must also be stored as models in the repository.
3. The repository must allow the editing of models (e.g., creation of new objects, creation of new associations/relations between models and between models and other items, modification of the properties of objects and associations, annotation of models etc.).
4. Besides ‘Models’, are there any other artefacts that should also be stored in the repository (e.g. documents, presentations, etc.)?
   If yes, specify which other types of artefacts you have in mind and what supporting functionality you expect (e.g., creation of new items, creation of new associations/relations between models and items, modification of the properties of items and, annotation of items, etc.).
5. The repository must facilitate the definition of relations between models. Please indicate which types of relations would be in your opinion important. Examples hereof are:
   - Model nesting
   - Model hierarchies
   - Cross-model references: e.g., links between (groups of) objects from different models, in-place model transformations (e.g., refinement, normalization) and out-place model transformations (e.g., translations between languages, code generation, etc.).
6. The repository must allow the definition of relations between models and other items (annotations on models, images, video, presentations, etc.) or types of documents (pdf, MS word, xls, etc.). Please indicate the types of items that should be related to models.

Locking and configuration management
7. The repository must accommodate concurrent access to models and objects and must have an explicit locking mechanism. Please specify the lock granularity you consider satisfactory (e.g., model level, object level).
8. The repository must have a model check-in/check-out function, which would allow to set the granularity of the lock and to explicitly lock-out a (portion of a) model when someone is working on it.
9. The repository must have version management functionality. Please specify the version management granularity you consider satisfactory (e.g., collection of models level, model level, model element level).
10. The repository must allow the control of changes, including the recording thereof (e.g., via labeling of models or model annotations with time/date labels, plateaus, phases, etc.), that are made to the models, and their documentation throughout the architecture development and lifecycle.
11. The repository must support revisions/releases of collections of models.

Traceability
12. When reused or referred to, an object may not be deleted as is: co-occurrences and references to it should be deleted too. This means the repository should enforce referential integrity between artefacts. As a result, the repository should be aware of relations between models and model artefacts.

Navigation/Search/Filtering
13. The repository must allow the navigation/search/filtering by object types (e.g., navigation tree based on diagram types - expanding the tree node for example for a process diagram lists all the activities and sub-processes within that process.)
14. The repository must allow the navigation/search/filtering by names (e.g., navigation tree based on model names - double-clicking on a class diagram name displays the diagram in a diagram pane).
15. The repository must allow the navigation/search/filtering based on associations/relations between models (e.g., in the case of nesting double-clicking on a sub-process in a process model causes a jump to the detailed model of that sub-process).
16. The repository must allow the navigation/search/filtering by attributes (e.g., expanding the node for a class in a navigation tree displays the list of attributes and operations for that class).
17. The repository must have configurable navigation toolbars. For example, the configuration of the navigation toolbars may allow the user to
   - select the page to be displayed when the repository (model) browser is launched. Thus, the model that is designated as the “home page” will be displayed.
   - set the number of days to keep the history list for the “visited” models.
   - define/refine a search and to choose between different available search engines.
18. The repository must allow graphical model browsing (e.g., using fish-eye techniques).

Model visualization
19. The repository must allow the configurable/customizable visualization of model artefacts: the user can select from different predefined graphical symbol sets (or define a new one) the ones that will be used to visualize model artefacts.

1 Out-place transformations can be useful, for example, in order to make model artefacts originating from one modelling tool easily accessible to other modelling tools (e.g. export to other formats).
20. The repository must allow zooming in/out models.
21. The repository must allow a viewpoint-based visualization of models.
22. The repository must allow a role-based visualization of models.
23. The repository must allow a rights-based visualization of models.
24. The repository must allow the visualizations of models based on custom defined filters (e.g., visualization design adaptable to the user organisation or to a certain user group).

Monitoring, analysis, reporting and publication
25. The repository must include monitoring and reporting functionality. Reporting includes the procedures concerning the extraction, interpretation and publication of analytical information about the stored model artefacts and may occur regularly (as result of monitoring) or on demand.
26. The repository must include analytical tools such as consistency checks, referential integrity mechanisms, impact of change analysis, model comparison and gap analysis, etc.
27. Publication/export of models and reports in/to other formats (e.g., in HTML, pdf, word, etc.) must be possible.

User & access rights management
28. The repository must support role-based user management (i.e., coupling persons to roles).
29. The repository must support role-based access rights management.
30. The repository must support user-based access rights management (i.e., based on the user’s id/profile).
31. The repository must allow the definition of access rights at model level.
32. The repository must allow the definition of access rights at model element level.
33. The repository must support the provision of feedback (i.e., a sort of “bug reporting/help desk” system) – at model element level, model level, revision level and/or release level.

Integration
34. The repository must allow the integration with other applications and tools (e.g., modeling tools, workflow management/execution engines, change/project management tools, office tools, intranet portals, other repositories/data bases, search engines etc.). Please indicate with which types of applications the integration would be in your opinion desirable and to what extent (i.e., at the data level – e.g., via import/export functionality, at the application level – e.g., via APIs, pluggins, adaptors, etc.).
35. The repository must allow single sign-on.
36. The repository must allow the definition of access rights integration with LDAP.