REuse and Migration of legacy applications to Interoperable Cloud Services
REMICS
Small or Medium-scale Focused Research Project (STREP)
Project No. 257793

Deliverable D2.2

REMICS Methodology, Interim Release

Work Package 2

Leading partner: Tecnalia

Author(s): Tecnalia, SINTEF, DOME, Fraunhofer, SOFTEAM, Netfective, DISYS

Dissemination level: Public

Delivery Date: 30 September 2011

Final Version: 1.0

Versioning and contribution history

Version	Description	Contributors
0.1	Document initialized	Gorka Benguria (Tecnalia)
0.2	Template for different phases	Gorka Benguria (Tecnalia)
0.3	Introduction of EPF approach	Gorka Benguria (Tecnalia)
0.4	Definition of Activity areas	Tecnalia, SINTEF, DOME, Fraunhofer, SOFTEAM, Netfective, DISYS
0.5	Definition of the life cycle	Tecnalia, SINTEF, DOME, Fraunhofer, SOFTEAM, Netfective, DISYS r
0.6	Final editing	Gorka Benguria (Tecnalia)
0.7	Review	Parastoo Mohagheghi (SINTEF)
0.8	Review	Yolanda Gomez (Dome)
0.9	Review	Christian Hein (Fraunhofer)
0.9.5	Review	Brice Morin (SINTEF)
1.0	Final version	Gorka Benguria (TRI)

Executive Summary

This deliverable is focused on the documentation of the REMICS methodology for the migration of legacy systems to Service Cloud platforms. The document has a parallel representation of the methodology in a software process engineering model, in this case a representation in SPEM in the EPF tool. This means that this document is accompanied by a standard definition of the methodology described.

The methodology reuses existing good practices and method fragments from previous methodologies and it explicitly defines dedicated method fragments developed in REMICS. This is the main deliverable of the Task 2.2 of REMICS.

This deliverable follows the SPEM (Software Process Engineering Metamodel) approach for formalisation of the methodology modelling defining work products, workflows, tool support and guidelines. The methodology model is released as an open source. In parallel to EPF it can also contain standard definitions of the same methodology in other languages such as the possible resulting representation from the SEMAT initiative.

Table of contents

1 Introduction

The cloud scenarios (IaaS[1], PaaS[2], and SaaS[3]) offer software developers and providers a new wide range of possibilities for solving many issues in their solutions and even for providing new functionalities. Some examples of issues that the cloud technologies may solve are: server availability, server resizing, load balancing or storage resizing usage accountancy. Some examples of new functionalities that could be integrated are backup facilities, remote management, remote monitoring or statistics.

Unfortunately, the cloud scenarios could not be used directly in all the cases. Depending on the vision of the future system on the cloud it may be necessary to reengineer partial or totally the legacy system in a new cloud compatible system.

The current document presents a methodology to support the organisation in the adaptation of their legacy systems to the cloud. More specifically, this deliverable will be focused on the documentation of methodology in a report format. On parallel a the standard definition of the REMICS methodology for the migration of legacy systems to Service Cloud platforms will be defined. This means that this document is accompanied by a standard definition of the methodology described.

The methodology reuses existing good practices and method fragments from previous methodologies and it explicitly defines dedicated method fragments developed in REMICS. This is the main deliverable of the Task 2.2 of REMICS.

This deliverable follows the SPEM approach for formalisation – methodology modelling defining work products, workflows, tool support and guidelines. The methodology model is released as an open source. But it can also contain standard definitions of the same methodology in other languages.

The document is structured in the following sections:

- Migration dimensions: it describes the big context of the migration project contextualizing the REMICS focus.
- Challenges: it identifies the challenges of the REMICS use cases
- Principles: It introduces the principles that will be followed during the development of the methodology
- Overview: It makes an overview of the proposed methodology
- Reused Fragments; it makes reference to the fragments subject to be reused in the methodology
- Main Activities: description of the main activity areas of the methodology
- Lifecycle: it present the lifecycle recommended for the project
- EPF: it describes the way in which the methodology will be implemented in EPF
- Conclusions: it summarizes the document

[1] Infrastructure as a Service

[2] Platfrom as a Service

[3] Software as a Service

2 Migration Dimensions

Migration of application to the new cloud infrastructures can be understood in two different dimensions; one more business oriented while the other more technology oriented.

The Business orientation is focussed on the migration of the system to support the cloud business models; i.e. business models that take advantage of the internet capacities to deal with most of their supporting activities. It usually involves additional functional requirements over the legacy system, and few non-functional requirements.

The new market trends include globalization and service orientation as important drivers which are changing the way business operates. Businesses are requesting flexible applications that can be acquired seamlessly and independently of the location. This situation is progressively pushing businesses from the owned system orientation to the service orientation. Accordingly, more and more traditional software vendors are noticing the need to transform their current business and technology model in order to remain in the market. Software as a Service (SaaS) has been set by these companies as a mandatory way to keep their existing customers while at the same time seizing the chance of acquiring new customers in unexplored markets.

The technology orientation is focussed in the migration of the system so it is able to run in the new cloud environment taking advantage of the new technologies without adding too much additional functionalities. It usually involves new non-functional requirements over the legacy system, and few functional requirements.

The new cloud scenarios (IaaS, PaaS, and SaaS) offer software developers and providers a new wide range of possibilities for solving many issues in their solutions and even for providing new functionalities. Some examples of issues that the cloud technologies may solve are: server availability, server resizing, load balancing or storage resizing usage accountancy. Some examples of new functionalities that could be integrated are backup facilities, remote management, remote monitoring or statistics.

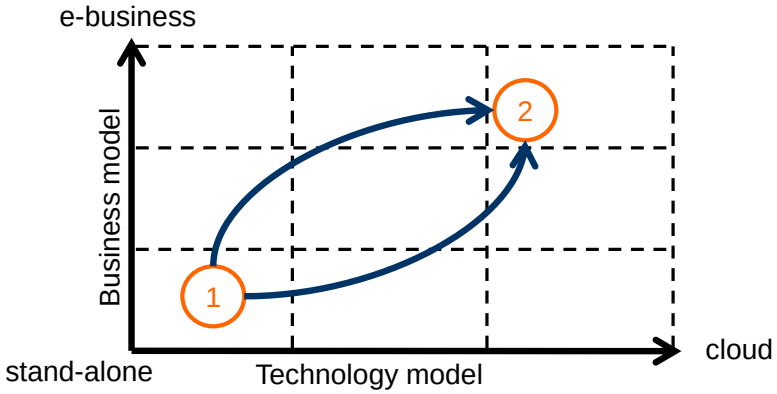

Figure 1 – Migration dimensions

The migration as shown in the previous figure could be summarised as the progress of an existing system in the business and technology model dimension.

Typical features of evolution of the business model are:

- Customer billing: the calculation of invoices, management of invoices, different ways of payments, management of accountancy errors, unpaid management, …
- Customer support: the management of incidences of customers including technical and non-technical.
- Legal: the management of the customer and company rights.
- Financing: the financing need of the customer in case it is required.

- Adds (publicity): the management of the banner based publicity.
- Marketing: Marketing of the system.
- Federation: Federation with other complementary systems.

Example features of evolution of the technology model are:

- Billing: how to control de consumption of services by the different stakeholders.
- Security: Add the necessary security to ensure that only allowed customer access the system. This security could be enhanced with other security properties such as non-repudiation.
- Maintainability: Changes to make the cloud system easier to maintain without affecting the customer work.
- Interoperability with in house and other platforms: identify interoperability scenarios and implement mechanisms to improve and ensure that interoperability.
- DB interfaces, location and replication.
- GUI (RIA, Mashups): Implementation of additional interfaces o new interfaces that allow the access to the system functionality remotely. This could be client application, RIA, Mashups, mobile applications, etc.
- User management: implementation of user management features, to register and control the users and the user they make of the system.
- Performance: Activities to control the performance of the system and be able to adapt the cloud infrastructure in line with the performance needs of the system.
- Availability: Features to control the availability of the system.
- Scalability: How the system can be resized to adapt to lower or bigger requirements of size and performance.
- Configurability: The system should be configurable to the different needs of the customers. It should also be able to record those configuration needs.
- Internationalization: Depending on the expected users, it could be necessary to implement internationalization features on the system.
- Governance: Features to manage the different services provided by the system.
- SLA QoS: Control the provided quality, the committed quality, the quality expected from other external services, etc.
- Updating mechanisms: mechanisms to include new features on the system, including the possibility of managing different branches depending on the contract with the customers.
- Monitoring and logging: recording of the performance of the system in their different components and key variables.
- Development environment: Implementation of testing (alfa, beta, …) versions of the system, that allow to develop and test new features without affecting existing ones.

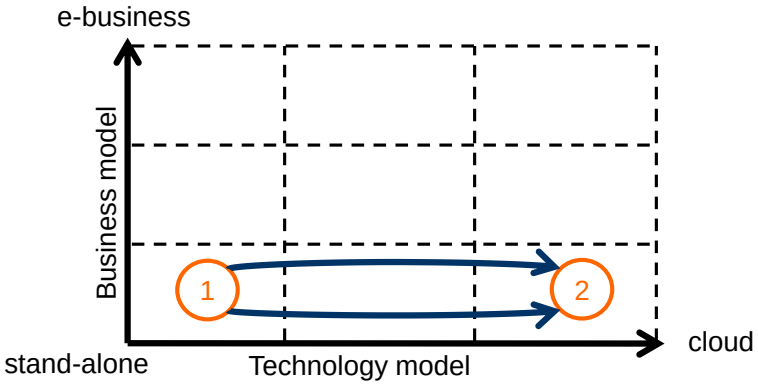

Figure 2 – REMICS Migration scope

In REMICS we focus mainly in the evolution on the technology model without adding too much extra features apart from those already available in the existing legacy application. The covering of the business model migration will require a more in depth analysis of the different e-business models, the

features they include and the good practices to implement those features. This is out from the scope of the project and is not required by the scenarios.

3 Migration and REMICS Challenges

Existing approaches and methods, presented in the deliverable D2.1, for transforming a legacy system into a cloud compatible system still have some shortcomings. These shortcomings appear both on the technical side, in the way in which they treat interoperability, reliability, QoS, SLA management, scalability, configurability or multi-tenancy, basic issues in the migration of cloud applications and even in the business side usually in the way in which they treat the service provision, some additional maintenance features and procedures may be required. The business side shortcomings may grow dramatically if the cloud migration involves some kind of transformation of the migrated software into a SaaS.

A deficient management of these shortcomings may end up with high investments from the side of the companies with little security that the product, offered in the context of the cloud, will be feasible for their current customer spectrum.

Therefore, a new complete procedure model is needed, which helps enterprises improve their technical know-how in order to migrate their software to cloud computing platforms. This solution will need to face up with the following identified challenges:

- Legacy **applications rarely can be applied directly without adjustments to the cloud scenarios**. Changes are required in the business model in order to ensure some degree of success in the cloud adaptation process.
- **Not all components are migrated in the same way**. Depending on the cloud migration scenario and final vision of the system it could be possible to find different situations: components coded from scratch, components recovered, components wrapped, components untouched, components acquired, components subcontracted, etc.
- The **requirements** of the final system come in its majority **from the legacy** system. Besides, depending on the reengineering of the system it may be possible that some of them are implemented completely in the reused components and it is not really valuable to expend effort in their standard definition.
- It is quite difficult to avoid the introduction of new features during the reengineering of the system. It is important to identify these **additional requirements** and the way in which they will be validated.
- The cloud orientation requires in most of the cases the implementation of a **service oriented architecture** in order to communicate the different parts of the system in a flexible and explicit way.
- Besides, the product of the migration to the cloud is generally provided to the final user as a service. This also supports the recommendation to evaluate the possibility of implementing service oriented architecture. This may also introduce some SaaS issues that may require the introduction of additional features to the system.
- Service based software is linked to specific architectural aspects such as multitenancy, monitoring, metering, billing, security, SLA and QoS that have to be taken into consideration.
- Software provided as a Service needs to be managed and controlled in the provision phase. Several generic functionalities such as monitoring, billing, accounting and access control have to be included for each service offered as a Service.
- The quality of services-based software is especially critical when transforming and maintaining the performance of the legacy application in the new environment.
- The **evolution and re-engineering of legacy software is unpredictable**, costly, and time-and resource-consuming, thus, is difficult to calculate the resources needed and the ROI to achieve.
- Currently, the provision of cloud services (namely IaaS and PaaS) is concentrated on a number of large companies. Placing an application on any of these providers usually means automatic **vendor-lock in** and no interoperability with other cloud providers, especially critical when wishing to port an application from one provider to another.

- The resulting system will require **new maintenance procedures**. For example, the resulting system will require procedures to access the cloud hosted parts of the system, to monitor the state of the cloud based parts and their accessibility, or to backup the information gathered in the cloud in case that exist.
- The resulting system will require **withdrawal procedures**. The usage of cloud infrastructure introduces the need to better define what should be done when stopping the service. The finalisation of a service may involve at least the backup of the stored information and the backup of the system as is.
- The resulting system must be **open to changes in the supporting cloud**. The usage of cloud infrastructure introduces a big dependency on the cloud provider, which in the end is transformed in a big risk in the business model. The application should be independent enough from the supporting cloud so it is possible to quickly reallocate some of all the parts allocated in the cloud to other places in order to reduce the impact of the changes in the supporting cloud.
- The migration supposes an opportunity to introduce **multiple interfaces** to the system: desktop clients, internet rich applications or mobile interfaces.
- The migration process is in general a process that occurs **exceptionally in a company**. It is a processes that take place each time that a new technology appears that may suppose a benefit for the system and requires deep changes on it.
- **Not well-known infrastructure and technology**. When migrating a system to a new technology usually the organisation and the people in charge have to deal with barely mature technologies that possibly nobody in the company know in detail.

All these challenges have been taken into account in the definition of the REMICS methodology affecting their activities, workproducts, and life cycle. Bellow we summarise the challenges and the way in which they are addressed by the methodology.

Table 1 – Challenge coverage in the methodology

Challenge	Coverage
Adjustment to the systems are necessary	Covered by the whole methodology
Not all components are migrated in the same way.	There is a preliminary analysis in the first stages about the expected vision of the migrated system.
The requirements come from the legacy.	There are recover activities supported by tools that gather existing requirements from legacy.
New features as additional requirements.	The methodology encourages the explicit identification of the additional features
Support for implementation of a service oriented architecture.	The methodology promotes the definition of a service oriented architecture as a part of the migrate activity
Support for SaaS at technology level.	The methodology supports the migration of a system to a SaaS at technology level.
Multitenancy, monitoring, metering, billing, security, SLA and QoS.	The methodology addresses these characteristics in different phases of the development. E.g. monitoring, metering and billing will be mainly addressed by the supervision activity area.
Software provided needs to be managed and	The methodology contains a supervision phase

controlled in the provision phase.	to take care of the provision phase.
Performance of the legacy application in the new environment.	The supervision phase take cares of the gathering of the information required for the monitoring and control of the final system.
Difficult to calculate the resources needed and the ROI to achieve.	The methodology proposes techniques in the early stage to analyse the best alternatives for the migration of the different parts of the system.
Vendor-lock in problem	The methodology promotes the implementation of SOA that identifies explicitly the boundaries of the systems and the interactions with external services. This facilitates the replacement of those services when necessary.
New maintenance procedures are needed	The supervision phase includes the development of the necessary features to support the maintenance of the final system along the time.
Withdrawal procedures are needed	The withdrawal phase defines the activities to be performed ant takes care of the development of the necessary features on the system.
Changes in the supporting cloud.	The methodology promotes the implementation of a service oriented architecture that identifies the interactions of the system with the cloud and help the developers to deal with those when modifying the cloud configuration.
Multiple interfaces may be required.	The service orientation facilitates largely the development of interfaces in different devices and technologies that access the features of the system through those interfaces.
No repeatable effort.	The methodology does not stress elements that are oriented for the support of repeatable processes such as the identification of reusable assets, or the gathering of lessons learnt to improve the process.
Not well-known infrastructure and technology.	The methodology proposes the application of a iterative and incremental lifecycle that help to better deal with problems derived from the lack of experience in a given domain.

4 Methodological Principles

Taking into account the scope of the methodology and the challenges to be addressed during the implementation of the system migration initiatives to cloud environments a set of methodological principles to guide the development of the methodology were selected.

4.1 With Respect to the methodology

- Migration Oriented

  Migration is the activity to deploy a system in a different platform. This migration usually involves changes to the existing system so it can be executed in the new deployment environment. To achieve this principle the methodology includes activities to identify the deployment environment and allocate de different components on those components.

- Interoperability Oriented

  Interoperability is a sensitive issue when addressing system integration. This even more sensitive when dealing with integration with external systems. To achieve this principle the methodology promotes the identification of interoperability issues in the new system and provides appropriate methodologies and tools to deal with them.

- Recovery Oriented

  When developing systems from legacy it exist the possibility to recover the application logic of part of the systems through the application of recovery techniques. This facilitates the actualization of the old components to more modern implementation that could highly improve the components. To achieve this principle the methodology promotes the recovery of the application logic and provides practices and tools to deal with that activity.

- Testability

  Provide means to test that the deployed service is working. This includes the provision of a testing interface to allow integrators to test their application without consuming the service, and to provide alternative graphical interfaces to test that the service is still running in case of an abnormal behaviour of the consumer application.

- Supported

  There must be means to ask for help in the different aspect of a service usage; i.e. aspects such as, the integration of the service, the usage of the service, the billing of the services, etc.

- Standard based Definition of the Methodology

  The apart from being the right methodology a methodology has to be described in a proper way so other people may consume the provision. To achieve this principle the methodology is described in EPF: EPF is a tool compatible with the SPEM standard that allows modelling the existing methodology so it can be latterly used to deploy an intranet or to deploy the process in a process management framework. In the future depending on the evolution of the SEMAT initiative another representation of the language may appear that provide an easier improvement.

- Iterative and Incremental Approaches.

  Methodologies may have different lifecycles, each lifecycle has some characteristics that makes them more appropriate depending on the context of the project where the methodology is going to be applied. The methodology will recommend an iterative and incremental approach for software development. This kind of lifecycle is appropriated to those cases where the domain is not mature and there is not enough experience in that kind of projects.

- No Methodological Framework.

  Methodologies can be defined as a standalone methodology ready to be used or a methodological framework that must be used as a basis to get the real methodology of the

project. In the project we expect to have a directly usable methodology that does not require hi configuration systems.

- Deployment and Exploitation Oriented

  This methodology provides good practices and support material to help in the creation of user manuals, tutorials,…, to help people to learn the system operation.

- Documented Design.

  In order to ensure a good communication among the project stakeholders (programmers, clients, maintainers,…), the methodology encourages the use of documenting good practices. In this sense the methodology push the use of standard such as UML.

4.2 With Respect to the Modelling

- Model Driven Development

  The methodology aims to hide the complexity of the technical details (platform specific model) of the system usage until the last stage of the development process. To achieve this objective the methodology will make use of the model driven development approach that will allow hiding the details of the interaction during the application logic specification (platform independent models).

- UML Standardised Notation.

  It promotes the use of UML as a standard notation for the project documentation. UML is a well know notation that enables the common understanding through different development teams and even with our clients.

- Standardised Notation Extensions for Models.

  UML is a standard modelling language defined to address any kind of system/software development. Because of that generality in some cases UML lacks a good support for some development scenarios. This is for example the case for SOA and Cloud deployment environments. This impacts the understanding and the possibility of applying analysis and automatic transformation over the resulting models.

4.3 With Respect to the Architecture

- Service-Orientation

  Service orientation has proven to be a successful method for the integration of systems during the last years. It provides a clear image of the interfaces that allow the interaction of the system with the external system and even a clear image of the interfaces among the components of the system. To achieve this principle the methodology promotes the identification of services between the different components of the architecture.

- Architecture Centric.

  It is very helpful to identify components and to better split the tasks, so the management of the project is easier. Besides it enables an overall view of the system, that could be very useful form introducing new project team members and in case that the application need to be maintained later on.

- Component Based.

  It promotes the division of the system in parts, and the identification of the relationships among those parts. In most of the cases those parts will be implemented as components with concrete interfaces. This promotes the parallel work and the later reuse of the developments.

  Moreover, the identification of the different parts that will build up the final system also allows an easier management of the work to be done. It permits the project manager to evaluate different alternatives regarding to the way in which each part will be obtained. It can decide about reusing, developing, subcontracting or buying in the COTS market each part of the system.

- Integration Oriented

  Integration of system may be implemented in different ways, file based, database based, memory based, interface based and of course service based. Independent form the concrete realisation of the integration the communication between different systems always involves the analysis of integration related issues such as security, transaction, backup, monitoring,. Therefore, this methodology explicitly requires the evaluation of certain requirements directly related with the integration issues such as security, transaction, backup, monitoring, etc.

  On the other hand, the integration needs make reference not only to the communication with existing system but also with the future systems to come. Consequently, it also takes into account possible future integration needs during the development of the system.

- Standard Based Integration

  The use of standards in the integration can make reference to two areas of concern: the communication and the messages. The use of communication standards is crucial to ensure the accessibility and the successful provision of services. For that reason, the methodology underlines the relevance of their use and the convenience of the evaluation of the standards that we are going to use and the way in which we expect to use them.

  On the other hand, the need to use of message standards depends on the use that we expect for our services. If we plan to use the services internally there is no need of such standards. But if we plan to provide those services to other companies it is advisable to use the message standard that is common in that domain. This will make easier the integration of our web service.

4.4 With Respect to the deployment

- Cloud Oriented

  The methodology aims to help the system and software developers in the development of cloud compatible systems and software. To achieve this principle the methodology will include specific activities to decide the deployment of the system, languages to represent that deployment in a shared language and tools to support the modelling.

- Availability

  The system should be available all the time. This usually involves the usage of replicated systems and the development of the system in a way that the system is capable of detecting abnormal situations and recovers from them.

- Predictability

  The system should be designed in a way that service execution time is known and repeatable, or at least that it can be calculated based on the input parameters.

- Security

  Based on the value of the information received and sent by the service it is necessary to establish the necessary security mechanisms. These mechanisms should cover not only the software system and the way in which it receives and sends the information but also the organisational environment that stores the system and the information.

- Observable

  In order to ensure the visibility of the state of the service, the system should provide means to ensure the safety and the security.

- Longevity

  The provision of the service should be ensured forever and in case that it is not possible at least for a predefined time. To ensure the provision of the service it is crucial to develop it taking into account the nature, vision, mission, and objectives of the involved companies.

4.5 Other secondary principles

- Traceability.

  The methodology pays a special attention to the explicit traceability between the work products resulting from the different tasks performed along the project. The traceability is crucial to manage in an effective way the change requests issued from the client during the product development as well as the maintenance of the products once the project has been finished.

- Concurrent Engineering.

  This methodology supports the Concurrent Engineering (or parallel development) through the split of the system in parts that could be implemented independently. Moreover, once the parts are identified and their functionalities are defined, the different parts can be subcontracted, bought, reused from previous developments, adapted…

- Systematic Reuse.

  During the project development it is necessary to evaluate other requirements apart from those of the client. Requirements related with the future use of the assets that build up the client product. Are we expecting to use those assets in the future? Do we expect to use them as they are or with some kind of adaptation?

- Interface Based Programming.

  It promotes the delegating of work based on a set of well defined interfaces. A well defined interface includes both functional (that could include pre-conditions and post-conditions) and non-functional requirements. The interface will be not only the basis for the development of the component but also the foundation for the testing of that component once it has been created or modified.

- Adaptable Methodology

  The methodology contains variation points that helps to customise it to the particular context in which each project is performed. Besides, there is also a strategy to process these variation points in order to obtain the right methodology for our project.

- Supports CMMI Certification.

  This methodology supports and is compatible with the CMMI certification achievement.

- Aspect Oriented

  The methodology aims to speed up the development while keeping the right level of quality. Aspect oriented programming helps to achieve this objective as it allows performing a better separation of concerns during the system development. It promotes the identification of functional components and a set of aspects that affect functional components.

- Market Orientation.

  e-Services are developed to make business through the selling of functionalities that the enterprise is able to provide through web services. In order to support that scenario the methodology should include the development of the business model of the service, the advertisement of the new web service(s), as well as the definition of the customer support and SLA management.

- Web Service Based.

  Web services are the next step after components in the implementation details isolation. They provide the same level of isolation from the implementation details while the also provide isolation from the installation details. Therefore, the use of web services instead of components discharge the developers (and later the people in charge of the deployment) from the problems that usually appear during the installation of several components. Therefore, the methodology also considers the use of a web service as an alternative to implement the functionality required from one of the part that are needed to build the system.

Web services, in the same way as components, are characterised by an interface and interface realisation. The interface is the most important information that a developer needs to be able to use a web service. Therefore the methodology makes emphasis in the development of the interfaces (WSDL) required for describing the web services developed.

- Business Logic First

  Specialising the general Model Driven Development that aims to hide the complexity of the technical details, the methodology will make use of web service profile that will allow hiding the details of the interaction during the business logic specification.

5 Methodology overview

Based on the REMICS scope, the challenges to be addressed and the methodological principles a methodology has been developed to support the organizations in the migration of their systems to cloud based scenarios. The overview of the methodology has two main parts: the in the activities to be performed and the lifecycle to be applied.

5.1 Methodology Activity Areas

The REMICS Migration Methodology currently implements the following activity areas:

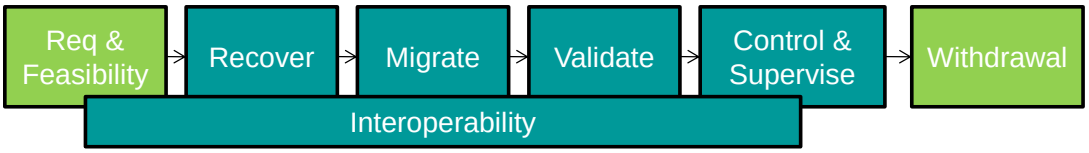

Figure 3 – REMICS Methodology Activity Areas

- **Requirements and Feasibility:** The purpose of the requirements activity area is to gather the migration requirements for the system, and to identify the main components of the solution and their implementation strategy. The purpose is not an exhaustive description of all requirements of the objective system, but the description of the requirements that will require development effort and will be used as a basis for the validation of the system. In this initial requirement elicitation process it is also not necessary to focus on those requirements that will come up from the systematic analysis of the legacy. This affects mainly the requirements of components that are going to be reengineered. Requirements that will appear during the recover activities through the application of migration tools.
- **Recover:** The purpose of this activity area is recover the knowledge from those legacy components that in the feasibility analysis has been pointed as candidates to be reengineered. The application of recover methods and tools will provide the application model of the legacy application. Moreover, the application of recover methods and tools may provide information on the requirements and even in the testing procedures for the migrated code.
- **Migrate:** The purpose of this macro-activity is to define and implement the new system based on the elements identified during the requirement and recover phases. This will include also the definition of the necessary new components to fulfill the past features and the additional requirements and developing a service oriented architecture

  As stated above, one of the basic requirements of a well designed SaaS application is the existence of monitoring, security and billing components (in case of new business model). These components need to be fully integrated in the resulting application and the methodology must give companies indications on when and why these components must be used.

  These components are generic and independent from the application provided but at the same time they are tightly linked to the software migrated. The separation of the supporting functionalities into different elements will provide a set of re-usable components for each application to be migrated, avoiding the necessity of having to develop these components from scratch for each new migrated product. Following, an overview of the components and the functionalities required for each one is provided:

  - Billing Component: Support for variable prizing plans and for automatic billing, purchase/clients order management and support for credit card payments.
  - Monitoring Component: Management of different monitoring parameters, SLA shaping and monitoring, and alert generation.
  - Security Component: Security for Multi-tenant environments, Information security management, and support of different security levels (technical, legal and business levels).

  - Intercloud API: Transparent support for different clouds providers.

- **Validation:** The purpose of this activity area is to define testing strategy to verify that the migrated system implements the requirements identified and verify that the components (including those not reengineered) and services work properly.

  This validation phase includes not only functional validation but what it is more important, non-functional validation, especially performance, reliability and security. In the case of cloud computing applications these three aspects must be stressed on.

- **Supervise:** The purpose of this macro-activity is to provide elements to control the performance of the system and to modify that performance.

  The last step, control and supervision, allows a company to monitor at all times, the performance of the application once this has been released and provisioned as a service, so it can be improved in performance, reliability, resources used and beware of possible degradation

- **Interoperability:** The purpose of this macro-activity is to provide tools that solve interoperability problems with 3rd part providers or any external components and services. This may include the development of new components.

  Interoperability is a crosscutting activity to the general methodology that deals with the interoperability issues that affect SaaS along the other activity areas (requirements, recover, migrate, validate, supervise, and withdrawal).

- **Withdrawal:** The purpose of this macro-activity is to provide elements to stop the service, with the purpose of finalizing it or with the purpose of moving to another cloud infrastructure.

5.2 REMICS Project Scope

The REMICS project is specially focussed in the recover, migrate, validate and supervise activity areas, and for those areas it will provide specific tools and techniques. The other two activity areas requirements and withdrawal are included in the methodology in other to fully cover the lifecycle of the cloud based applications. These two activity in principle will not receive special support.

After the extension of the REMICS project in 2011, with the incorporation of new partners with new capabilities the scope was extended to cover also the requirements activity area. Therefore, since 2011 the requirement activity area will also be provided with tools and techniques from the REMICS project.

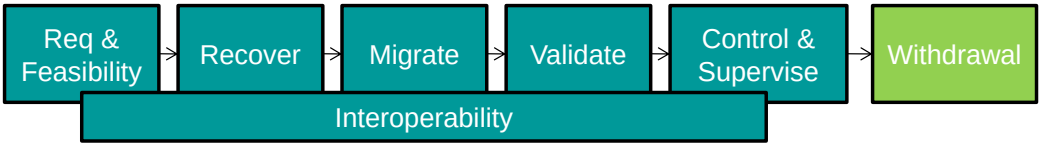

Figure 4 – REMICS Methodology Scope since 2011

6 Reused Fragments

Since the start of the software industry several methodologies have appear and have been applied successfully for different purposes in different technological contexts. It is clear that those methodologies cannot be directly applied to the REMICS context, as the purpose and the technological context is totally different. Anyway, this not prevent of reusing some parts of those methodologies that may be addressing common problems to the problems addressed in the migration projects aimed by the REMICS methodology.

For the implementation of the activities contained in the activity areas of the methodology the following potential source of reusable methodology fragments have been identified in the context of the D2.1:

- SHAPE for service orientation, and scoping of the methodology
- MOMOCS for model based migration strategies
- WEBMON for service composition testing
- COMBINE for different models to development of component based solutions
- SMART for feasibility analysis of the migration options of the different parts of the components
- SAE scoping of the methodology
- SOAD and SODA for development of new services
- SCRUM and XP for the methodology lifecycle management and agile practices

All these works are analysed in detail in the D2.1.

7 Main activities

7.1 Requirements and Feasibility

7.1.1 Purpose

The purpose of the requirements activity area is to gather the additional requirements for the migrated system, and to identify the main components of the solution and their implementation strategy. The purpose is not an exhaustive description of all requirements of the objective system, but the description of the requirements that will require development effort and will be used as a basis for the validation of the system. In this initial requirement elicitation process it is also not necessary to focus on those requirements that will come up from the systematic analysis of the legacy. This affects mainly the requirements of components that are going to be reengineered, requirements that will appear during the recover activities through the application of migration tools.

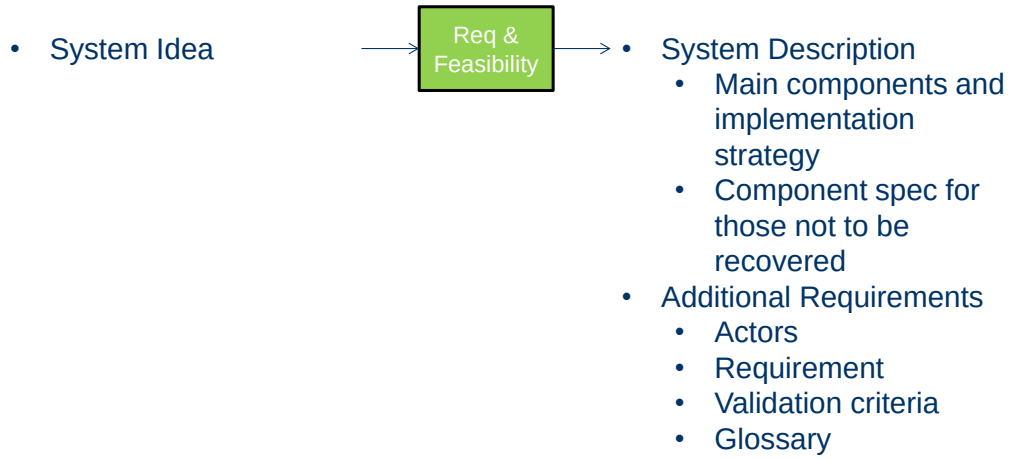

Figure 5 – REMICS Requirements Activity Area Overview

7.1.2 Activities

Describe the system

A preliminary description is necessary, so with the System Idea, and some additional information, a description of the system will be obtained. The System Idea could come from inside the company or outside (a customer). Regarding to the additional information, if any is needed; it could be obtained through several means such as surveys, meetings and market reviews.

One of the objectives of this preliminary description is to serve as a basis for the evaluation of the feasibility of the system. Therefore, during this activity it is possible that more than one description is developed in order to allow the evaluation of different alternatives. Another objective of the System Description is to serve as a brief summary of the nature of the service. It will be useful to acquire a quick idea about the system that it is going to be developed without having to read the entire System Requirements document.

In the end, the output of this work will be the System Description, and it will contain relevant information about the system to develop.

Users are who have to describe their needs and expectations, while the analysts have to help them in the identification of all the relevant aspects of the system. Finally the users, the analysts and the project manager should agree with the description.

This activity has no automation possibilities.

There are tools for requirement gathering that can be used in this activity. From REMICS perspective, this activity is not expected to be supported by specific tools developed in the REMICS.

Apply techniques to evaluate feasibility.

In a migrated system not all the parts are equally reusable; in fact there are many different ways to reuse components. In some cases the best way to reuse a component may be to wrap it, in other cases to reengineer, in other cases to replace with an external one, and in other cases to implement from scratch, etc. The SMART method provides a methodology to evaluate the feasibility of the different approaches, providing valuable information for this decision making process.

In the end, the output of this work will be the System Description, and it will contain relevant information that justifies the selection of the implementation strategies for the different parts of the system.

Analyst, project manager, user and customer should participate in this analysis and in the final decision about the best strategy for the division of the system.

This activity has no automation possibilities.

There are tools for feasibility analysis that can be used in this activity. From REMICS perspective, this activity is not expected to be supported by specific tools developed in the REMICS.

Identify actors

With the gathered information in the System Description and the System Requirements it should agree with stakeholders (especially with the one that will pay the system) which actors are required to take part in the final solution. An actor is any person, machine, entity and so on that interacts with the system.

The outputs from this work will be the description of the involved actors that will take part in the delivery of the system functionality described in the System Requirements Document.

Users provide all the external actors that they require in the final solution. The analysts will be the responsible of the gathering of this information from the users.

Identify additional requirements (it may be split in functional and non functional)

Usually migration implies also the introduction of new features or the modification of some of the existing ones. Based on the System idea and the preliminary system description identify the new and the changing features. These changing features may include both functional and non-functional changes.

These functionalities will be represented as UML use cases, and they will include a description that optionally could incorporate an activity or a sequence diagram. The result from this work will be the System Requirements requested by the entity that will pay for the system.

All the features are provided mainly by the users (or a people that represent his interests). These features will be evaluated and estimated by the analysts and finally selected and prioritised by the one that will pay for the development.

This activity has no automation possibilities.

There are tools for requirement gathering that can be used in this activity. From REMICS perspective, this activity will be supported by a customised version of trac.

Establish validation criteria

Specify the indicators which will demonstrate that the system operates perfectly and that it meets the requirements. This will be obtained agreeing with the user the set of Validation Criteria that will be used in the acceptance test of the system. It is difficult to obtain an agreement of the validation criteria but it is crucial to establish an end to the project.

The output is a Validation Criteria document which contains a set of valuable indicators.

Analysts and the project manager together with the users prepare the set of validation criteria and later the users will decide to accept or reject. All can ask for changes until an agreement is obtained.

This activity has no automation possibilities.

There are tools for requirement validation that can be used in this activity. From REMICS perspective, this activity will be supported by a customised version of Trac.

Elaborate glossary

Gather the system terminology in a Glossary which supports the non-ambiguous use of the terms related with the system.

The output is a Glossary document that has to be updated during the course of the system development.

Analysts, users and the project manager have to gather together the terms and agree with a common definition, so the understanding between all the roles in the project is better.

This activity has no automation possibilities.

There are tools for glossary definition that can be used in this activity. From REMICS perspective, this activity is not expected to be supported by specific tools developed in the REMICS.

7.1.3 Tools

- Trac - for gathering the requirements

7.1.4 Inputs

⇒ System Idea and migration goals

7.1.5 Outputs

⇐ System Description

- Main components and implementation strategy
- Component spec for those not to be recovered

⇐ System Requirements

- Actors
- Requirement
- Validation criteria
- Glossary

7.1.6 Support Material

- System Description Template
- System Requirements Template

7.1.7 Workers

- Analyst

- User
- Customer
- Project Manager

7.1.8 Tips

- ☑ While documenting the System Requirements, use question marks in those points of the specification where doubts exist.
- ☑ Evaluate the need of using standards at several levels. At communication level agree on the standards that will be used and the way in which they will be used. At message level evaluate and agree on the standards to be used to improve the future integration of the system. The use of standard message types can improve the interoperability between the different components of the system and future components that could be acquired.
- ☑ Promote the interaction with the customer in order to speed up the development, establishing continuous and flexible communication mechanisms with stakeholders (customers and users, third parties involved in service delivery, other functional areas within the company, terminals manufacturers, etc.). Increasing communication with stakeholders, with minimum bureaucracy, and favouring their involvement drives to a prompt identification of inputs and needs that will prevent problems later in the life cycle, when solving them is much more costly.
- ☑ Ask the customers to prioritize the user requirements. It could be useful when doing several iterations whose result is a improved new version of the system.
- ☑ Optionally, it could be interesting to list the system requirements (from users and other stakeholders) in a System Requirements Table to improve the traceability of the requirements along the different stages and work products.
- ☑ Identify those aspects that crosscut the functional components such as log systems, performance meters, security monitors, etc. The objective is to define a global solution that could be reused in the affected functional components instead of building those aspects inside each of the affected functional components.
- ☑ In case that the requirements are not clear, use prototyping in order to clarify them. Remember to evaluate the cost of the prototyping.
- ☑ Encourage glossary usage to ensure that everyone in the team uses the same term to appoint the same issue. Usage includes consulting and upgrading.
- ☑ Detail the different communication channels or terminals (phone, WAP, email, SMS, HTTP, etc.) to be used by each group of users.

7.2 Recover

7.2.1 Purpose

The purpose of this activity area is recover the knowledge from those legacy components that in the feasibility analysis has been pointed as candidates to be reengineered. The application of recover methods and tools will provide the application model of the legacy application. Moreover, the application of recover methods and tools may provide information on the requirements and even in the testing procedures for the migrated code.

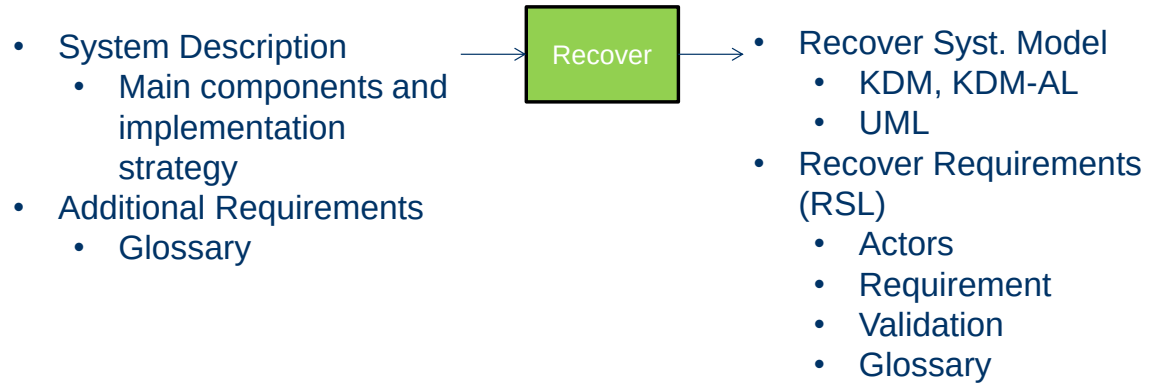

Figure 6 – REMICS Recover Activity Area Overview

7.2.2 Activities

Collect the code

The first activity in the recover activity area is to gather the code of those components designated to be reengineered through the recovery process.

In the end, the internal output of this work will be the code of the component that embeds the application logic of that component.

Analyst and the project leader are those in charge of the gathering of the legacy code of the components to be reengineered.

This activity has no automation possibilities.

There are no tools for collecting the code that can be used in this activity. From REMICS perspective, this activity is not expected to be supported by specific tools developed in the REMICS.

Recover system knowledge

Once the code has been collected, the next activity is to process the code and recover the application logic, business logic, business rules, architecture, etc. out from it. This is achieved by the searching of structural statements in the code and patterns of code. These elements are then used to construct the KDM of the system. The KDM representation is a model that represents the knowledge of the system.

In the end, the output of this work will be the Recover System Model in KDM format, which contains the system knowledge.

Analyst and the project leader are those in charge of the processing the legacy code of the components to be reengineered in order to extract the system knowledge..

This activity is highly automated by the usage of code analysis tools. Depending on the language and the support that the tool has for that language the automation may be higher or lower. For new

language it is necessary to define the elements and the patterns to be found in the code in order to extract the system knowledge.

From REMICS perspective, this activity will be supported by specific tools developed in the REMICS for system knowledge recovery from code.

Refine system knowledge

Once we have a preliminary version of the system knowledge, this representation is analysed to find additional patterns proper of the application that can facilitate and simplify the understanding system knowledge extracted from the code.

In the end, the output of this work will be the refined Recover System Model in KDM format, which contains the system knowledge.

Analyst and the project leader are those in charge of understanding the extracted system knowledge model and identify further patterns and knowledge on it.

This activity is partially automated by the usage of code analysis tools. Tools may provide recommendations about the possibility of additional patterns in the system knowledge.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for system knowledge analysis.

Generate system model

With the refined system knowledge representation (KDM) the next step will be the generation of the UML representation of the system knowledge. UML in contrast to KDM, is a modelling language familiar to analyst and programmers. KDM is an abstract representation of the system knowledge that can be placed in the middle way from the code to the UML models.

In the end, the output of this work will be the refined Recover System Model in UML format, which contains the system knowledge.

Analyst and the project leader are those in charge of requesting the UML generation out from the KDM representation of the system knowledge.

This activity is highly automated by the usage of model transformation tools, transformation from KDM representation to UML models.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for system knowledge transformation.

Generate system requirement

With the refined system knowledge representation (KDM) the next step will be the generation of the requirements out from the system knowledge. These requirements will be represented using the RSL (Requirement Specification Language) approach. RSL propose to describe the requirements of the system with a combination of use case models, scenario description language and domain models.

In the end, the output of this work will be the Recover Requirements preferably in RSL format.

Analyst and the project leader are those in charge of requesting the Requirements generation out from the KDM representation of the system knowledge.

This activity is highly automated by the usage of model transformation tools, transformation from KDM representation to Requirements (RSL) models.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for system knowledge transformation.

Recover application testing

With the refined system knowledge representation (KDM) the next step will be the generation of the testing out from the system knowledge. These testing procedures will be focused in the validation of the requirement extracted in other activity of this activity area.

In the end, the output of this work will be the Recover Requirements Validation procedures.

Analyst and the project leader are those in charge of requesting the Requirements Validation procedures generation out from the KDM or UML representation of the system knowledge.

This activity is highly automated by the usage of model transformation tools, transformation from KDM or UML to the testing models.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for system knowledge transformation.

7.2.3 Tools

- Code analysis tools
- System knowledge generation tools
- System knowledge analysis tools
- System knowledge transformation tools

7.2.4 Inputs

⇒ System Description

- Main components and implementation strategy

⇒ Code

⇒ Additional Requirements

- Glossary

7.2.5 Outputs

⇐ Recover Syst. Model

- KDM, KDM-AL, , EKDM
- UML

⇐ Recover Requirements (RSL)

- Actors
- Requirement
- Validation
- Glossary

7.2.6 Support Material

- System knowledge model template
- UML system model template
- System Requirements Template

7.2.7 Workers

- Analyst
- User
- Customer
- Project Manager

7.2.8 Tips

- ☑ Identify those aspects that crosscut the functional components such as log systems, performance meters, security monitors, etc. The objective is to define a global solution that could be reused in the affected functional components instead of building those aspects inside each of the affected functional components.
- ☑ Encourage glossary usage to ensure that everyone in the team uses the same term to appoint the same issue. Usage includes consulting and upgrading.
- ☑ Ensure that the naming conventions of the transformation match the naming conventions on the company. If not, consider adapting the generated model to comply with the naming conventions.

7.3 Migrate

7.3.1 Purpose

Migrate: The purpose of this macro-activity is to define and implement the new system based on the elements identified during the requirement and recover phases. This will include also the modernization of architecture and the definition of the necessary new components to fulfil the past features and the additional requirements.

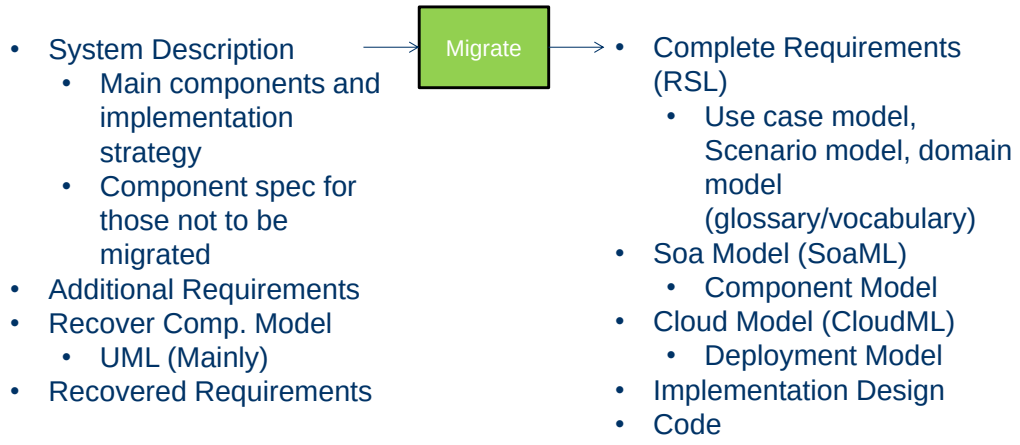

Figure 7 – REMICS Migrate Activity Area Overview

7.3.2 Activities

Complete definition of system requirement

With the system description, the additional requirement and the recover requirements integrate the final system requirement definition. This will be basically the aggregation of the additional and recovered requirements, with some requirements about the functionality at high level. The objective is to create a unique based on requirements with the purpose of validating the final system.

In the end, the output of this work will be the Complete Requirements including use case, scenarios description, validation procedures and domain models.

Analysts are those in charge of the development of the complete vision of the requirements for the migrated system.

This activity has no automation possibilities.

There are tools for requirement gathering that can be used in this activity. From REMICS perspective, this activity is supported by a customised version of trac.

Definition of service architecture

Starting with the system description and the recovered component models, define the architecture of the system to implement the requirements identified. The architecture will contain the identification of the services to be exchanged among the different components distributed in the cloud.

In the end, the output of this work will be the SOA Model of the application including the services and the components that implement those services.

Analysts are those in charge of the development of the SOA Model for the migrated system.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for SOA modeling.

Definition of cloud architecture

Starting with the system description and the SOA Model, define the deployment model of the system to implement the requirements identified. The deployment model will contain the identification of the location of the different components distributed in the cloud. Besides, it may be also necessary to take into account possible constrains (organisational, legal, etc) when defining this deployment model.

This model may raise some performance issues that may require changes in the SOA architecture or in the Cloud model.

In the end, the output of this work will be the Cloud Model of the application including cloud resources that will be used and the components that are located in those resources.

Analysts are those in charge of the development of the Cloud Model for the migrated system.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for Cloud modelling.

Implementation design

With the requirements, the SOA model of the architecture and the Cloud model of the deployment start implementing the design of the different components that require a model in order to be implemented.

In the end, the output of this work will be the implementation design of the application including UML models that describe how the components will work internally to implement their services.

Analysts are those in charge of the development of the implementation design for the migrated system.

This activity has no automation possibilities.

There are many tools on the market for component modelling that can be used in this activity. From REMICS perspective, this activity is not expected to be supported by specific tools developed in the REMICS.

PSM Transformation

With the complete model of the system (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) generate the code stubs necessary to run that system in the different cloud components.

In the end, the internal output of this work will be the code stubs of the system for the different parts of the system.

Analysts or programmers are those in charge of the generation of the code stubs out from the system models.

This activity is highly automated by the usage of model transformation tools.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model transformation.

Complete the code

With the code stubs and the system model of the system (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) complete the code to make the code stubs working.

In the end, the output of this work will be the code of the system for the different parts of the system.

Programmers are those in charge of the completion of the code stubs to obtain complete and runnable components.

This activity has no automation possibilities.

There are many tools on the market for coding that can be used in this activity. From REMICS perspective, this activity is not expected to be supported by specific tools developed in the REMICS.

7.3.3 Tools

- 💻 UML modeling tools
- 💻 SoaML modeling tools
- 💻 PIM4Cloud modeling tools
- 💻 Code generation tools

7.3.4 Inputs

⇒ System Description

- Main components and implementation strategy
- Component spec for those not to be migrated

⇒ Additional Requirements

⇒ Recover Comp. Model

- UML (Mainly)

⇒ Recovered Requirements

7.3.5 Outputs

⇐ Complete Requirements (RSL)

- Use case model, Scenario model, domain model (glossary/vocabulary)

⇐ Soa Model (SoaML)

- Component Model

⇐ Cloud Model (PIM4Cloud)

- Deployment Model

⇐ Implementation Design

⇐ Code

7.3.6 Support Material

🕮 UML system model template

- System Requirements Template
- Coding guidelines

7.3.7 Workers

- Analyst
- Programmer
- Project Manager

7.3.8 Tips

- ☑ Evaluate the possibility of using specific metamodels or UML profiles to avoid the technical aspects of the development while centring in the definition of the solution for the customer requirements. These annotations will be latterly used to generate the assets that will implement the technical concepts behind the business solution. This generation could be performed automatically if we have the appropriate transformation tool or by a technician.
- ☑ Analyse how the different aspects that crosscut the functional components could be approached to improve their reuse and their maintenance.

7.4 Validate

7.4.1 Purpose

The purpose of this activity area is to define testing strategy to verify that the generated system has the functionality and performance of the legacy system when required; implements the additional requirements identified and verify that the components (including those not reengineered) work properly.

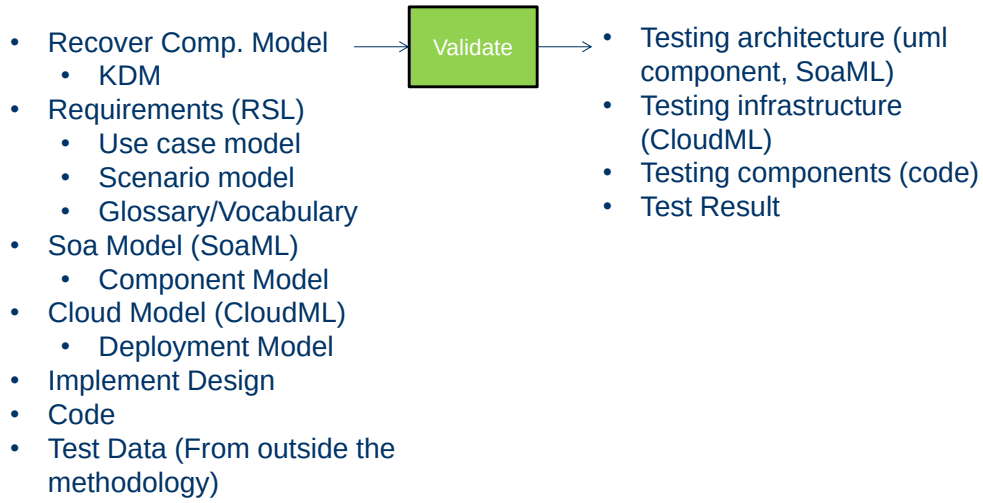

Figure 8 – REMICS Validate Activity Area Overview

7.4.2 Activities

Define testing infrastructure:

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) define the characteristics of the testing infrastructure to allow a continuous testing of the implemented features in a way that they can guarantee to certain degree the quality of the final system.

In the end, the output of this work will be testing architecture and the testing infrastructure for the system.

Analysts, programmers or testers are those in charge of the testing architecture and the testing infrastructure out from the system description and the models of the systems.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for modelling.

Identify and refine requirements to be tested

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) refine the system requirements to create test requirements.

In the end, the output of this work will be requirement with a refined set of validation criteria for the system.

Analysts, programmers or testers are those in charge of the refining of the system requirements in order to define testable validation criteria.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for requirement modelling.

Generate acceptance testing

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) and the testing environment generate test scripts to test the requirement scenarios.

In the end, the output of this work will be acceptance testing code for the system. This testing code will be focused in the scenarios described for the requirements.

Analysts, programmers or testers are those in charge generation of the acceptance testing.

This activity is highly automated by the usage of model transformation tools.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model based testing.

Import models elements to be tested

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) and the testing environment import those models that will be tested from a model perspective.

In the end, the internal output of this work will be the identification and import of those models to be tested using the model testing tools.

Analysts, programmers or testers are those in charge of the identification and import of those models.

This activity is highly automated by the usage of model import tools.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model transformation.

Define testing procedures

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) define the testing procedures to be applied in the different stages of the service lifecycle.

In the end, the output of this work will be testing procedures for the system.

Analysts, programmers or testers are those in charge of the definition of the testing procedures to validate the system quality.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model definition.

Implement testing strategy

With the description, the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design), the testing environment and the testing procedure implement the testing strategy.

In the end, the output of this work will be running testing infrastructure and the test results for the system.

Analysts, programmers or testers are those in charge of the establishment of the running testing infrastructure and the test results.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model transformation.

7.4.3 Tools

- UML modeling tools
- SoaML modeling tools
- PIM4Cloud modeling tools
- Code generation tools
- Model based testing tools
- Code testing tools

7.4.4 Inputs

⇒ Recover Comp. Model

- KDM

⇒ Requirements (RSL)

- Use case model
- Scenario model
- Glossary/Vocabulary

⇒ Soa Model (SoaML)

- Component Model

⇒ Cloud Model (PIM4Cloud)

- Deployment Model

⇒ Implement Design

⇒ Code

⇒ Test Data (From outside the methodology)

7.4.5 Outputs

⇐ Testing architecture (uml component, SoaML)

⇐ Testing infrastructure (PIM4Cloud)

⇐ Testing components

⇒ Requirements

- Refined validation procedures

⇐ Test Result

7.4.6 Support Material

🕮 UML system model template

🕮 Test result template

7.4.7 Workers

- Analyst
- User
- Programmer
- Tester
- Project Manager

7.4.8 Tips

☑ It's a good idea to have a testing plan to conduct the unitary test of the components. This is important to improve the quality of the developed products and, getting the optimal order and organization testing the components, moreover, the time of testing can be reduced.

☑ Evaluate the benefit of automating the tests for the components. This will speed up the implementation of changes from the user.

☑ It's very helpful to have an integration strategy, so fewer errors are made and the integration is easier and quicker.

☑ As in the previous phase, having a testing plan is a good practice which help us to develop high quality products and reduce testing time. Automating testing is other practice to have in mind to perform the testing of the integration.

☑ It's a very good idea to have a testing plan when doing the validation of the system. The testing plan should be based on the validation criteria.

7.5 Control & Supervise

7.5.1 Purpose

Supervise: The purpose of this macro-activity is to provide elements to control the performance of the system and to modify that performance.

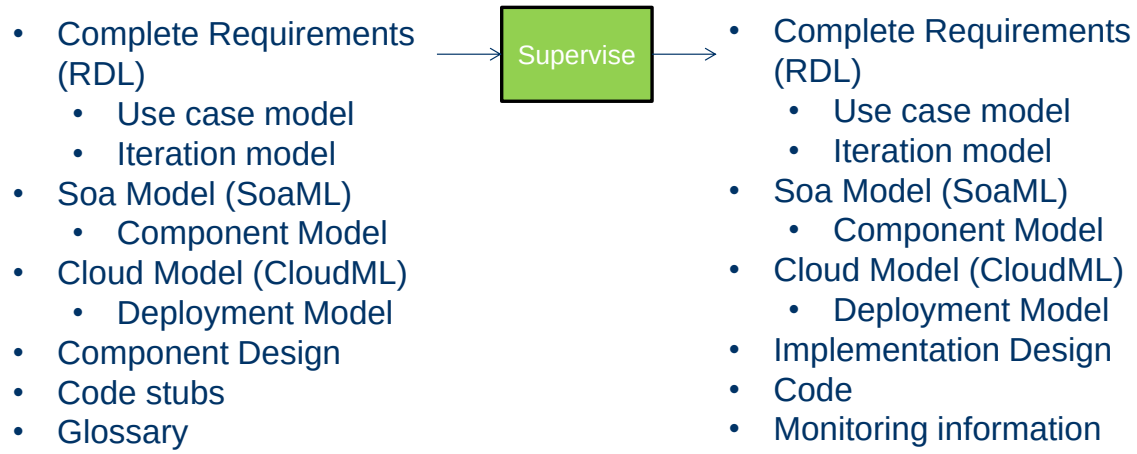

Figure 9 – REMICS Supervise Activity Area Overview

7.5.2 Activities

Identify monitoring procedures

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) define the monitoring procedures to be applied in the different stages of the service lifecycle.

In the end, the output of this work will be monitoring procedures for the system.

Analysts or programmers are those in charge of the definition of the monitoring procedures to monitor the system quality.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model definition.

Implement monitoring procedures

With the description, the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design), the monitoring procedures implement the monitoring procedures.

In the end, the output of this work will be running monitoring procedures and the monitoring information for the system.

Analysts or programmers are those in charge of the establishment of the running procedures infrastructure and the monitoring information.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model transformation.

Monitor the performance of the system in the cloud

During the stage of time in the development process that the system is running, apply the monitoring procedures to gather the monitoring data.

In the end, the output of this work will be the monitoring information.

Analysts or programmers are those in charge of the establishment of the running procedures infrastructure and the gathering monitoring information.

This activity is highly automated by the usage of the components developed as part of the project.

No tools are supported; components are developed as a part of the system.

Detect deviations

Starting from the monitoring data, develop analysis in order to detect deviations from the planned values.

In the end, the output of this work will be the monitoring information.

Analysts or programmers are those in charge of the establishment of the running procedures infrastructure and the gathering monitoring information.

This activity is highly automated by the usage of the components developed as part of the project.

No tools are supported; components are developed as a part of the system.

Perform corrective actions

Once deviations are detected, define possible corrective actions, that may consist on the tune up of certain components or in the introduction of new features in the system.

In the end, the output of this work will be the monitoring information.

Analysts or programmers are those in charge of the establishment of the running procedures infrastructure and the gathering monitoring information.

This activity is highly automated by the usage of the components developed as part of the project.

No tools are supported; components are developed as a part of the system.

Monitor corrective actions

Once corrective actions are putted in place, monitor the result of the corrective actions implemented to see the validity of the corrective action.

In the end, the output of this work will be the monitoring information.

Analysts or programmers are those in charge of the establishment of the running procedures infrastructure and the gathering monitoring information.

This activity is highly automated by the usage of the components developed as part of the project.

No tools are supported; components are developed as a part of the system.

7.5.3 Tools

- UML modelling tools
- SoaML modelling tools
- PIM4Cloud modelling tools
- Code generation tools

7.5.4 Inputs

⇒ Complete Requirements (RDL)

- Use case model
- Iteration model

⇒ Soa Model (SoaML)

- Component Model

⇒ Cloud Model (PIM4Cloud)

- Deployment Model

⇒ Component Design

⇒ Code stubs

⇒ Glossary

7.5.5 Outputs

⇐ Complete Requirements (RDL)

- Use case model
- Iteration model

⇐ Soa Model (SoaML)

- Component Model

⇐ Cloud Model (PIM4Cloud)

- Deployment Model

⇐ Implementation Design

⇐ Code

⇐ Monitoring information

7.5.6 Support Material

🕮 Requirement Template

🕮 UML system model template

7.5.7 Workers

Analyst

Programmer

Support

Project Manager

7.5.8 Tips

- ☑ Evaluate the need to define extra components with the purpose of testing or monitoring the performance of key components as a way to support the later integration of the system.
- ☑ When doing the manuals remember that the users don't know the system as well as the developers, so it's important to make an effort to explain all as clear as possible.

7.6 Interoperability

7.6.1 Purpose

Interoperability: The purpose of this macro-activity is to provide tools that facilitate the development and monitoring of the features on interoperability areas. This may include the development of new components.

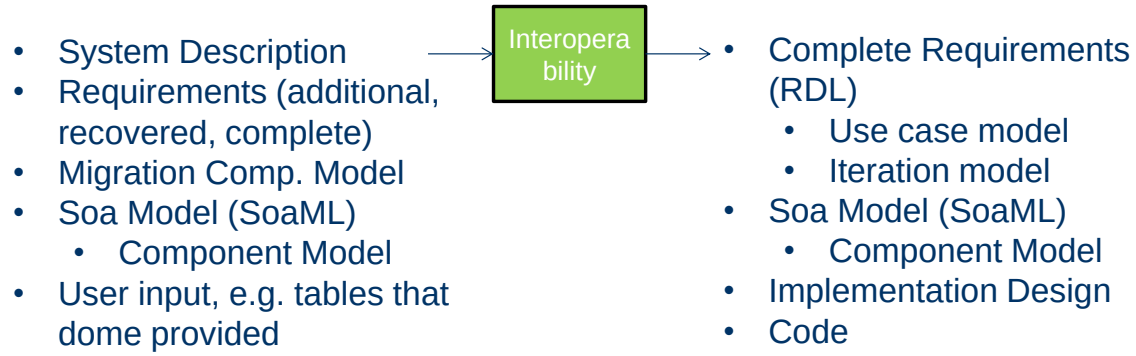

Figure 10 – REMICS Supervise Activity Area Overview

7.6.2 Activities

Identify interoperability problems/scenarios

Starting with the system description and later with the system model identify potential interoperability problems or scenarios. Typical scenarios include the replacement/extension of a modernized service by/with external services already deployed in the cloud, justified by cost, QoS issues.

In the end, the intermediate output of this work will be the interoperability problems or scenarios detected for the future migrated system.

Analysts are those in charge of the establishment of the interoperability problems or scenarios.

This activity has no automation possibilities.

No tools are supported; components are developed as a part of the system.

Define interoperability related requirements

Translate the interoperability problems and scenarios into new requirements for the final system. If no requirement can be defined for a given issue, that's possibly means that the problem is not a problem taking into account the scope of the system.

In the end, the output of this work will be the requirement specification, more specifically, the requirement specification for the interoperability management.

Analysts or programmers are those in charge of the definition of the interoperability related monitoring.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model definition.

Perform interoperability analysis

Analyse the interoperability problems and requirements to have a basis for the definition of strategies to deal with them.

In the end, the output of this work will be the components to be implemented in order to deal with the interoperability requirements.

Analysts or programmers are those in charge of the analysis of the interoperability issues and the definition of components to address those problems. .

This activity is partially automated by REMICS tools

From REMICS perspective, this activity is supported by tools developed in REMICS to assist and guide designers when defining mapping between data structures and mediators to align services at the behavioural level.

Implement interoperability components

Implement interoperability components when necessary to deal with the interoperability problems and requirements.

In the end, the output of this work will be the components to be implemented in order to deal with the interoperability requirements.

Analysts or programmers are those in charge of the implementation of interoperability related components.

This activity can be partially automated.

From REMICS perspective, this activity will be supported by tool developed in REMICS to generate code from the specification of data mappings and mediators.

Interoperability monitoring

Once interoperability components are deployed, monitor the result of the interoperability components implemented to see the validity of these components.

In the end, the output of this work will be the interoperability information.

Analysts or programmers are those in charge of the interoperability monitoring.

This activity can be partially automated.

From REMICS perspective, this activity will be supported by the tools developed in the control and supervise activity.

7.6.3 Tools

- UML modelling tools
- SoaML modelling tools
- PIM4Cloud modelling tools
- Code generation tools

7.6.4 Inputs

⇒ System Description

⇒ Requirements (additional, recovered, complete)

⇒ Migration Comp. Model

⇒ Soa Model (SoaML)

- Component Model

⇒ User input, e.g. tables that dome provided

7.6.5 Outputs

⇐ Complete Requirements (RDL)

- Use case model
- Iteration model

⇐ Soa Model (SoaML)

- Component Model including interoperability components.

⇐ Implementation Design

- Mapping between data models
- Service Mediators

⇐ Code generated from design

7.6.6 Support Material

🕮 Requirement Template

🕮 UML system model template

7.6.7 Workers

- Analyst
- User
- Programmer
- Project Manager

7.6.8 Tips

☑ No tips in this version.

7.7 Withdrawal

7.7.1 Purpose

Withdrawal: The purpose of this macro-activity is to provide elements to stop the service, with the purpose of finalizing it or with the purpose of moving to another cloud infrastructure.

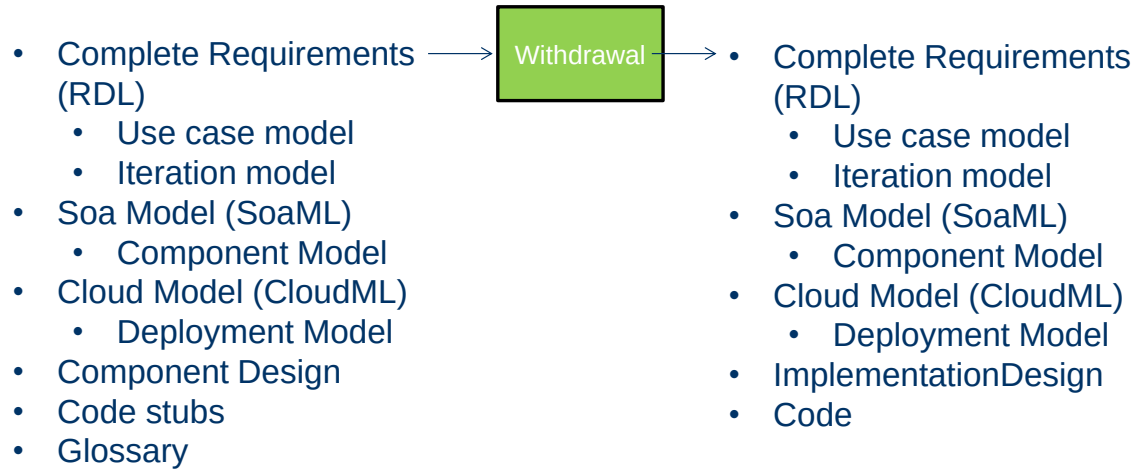

Figure 11 – REMICS Supervise Activity Area Overview

7.7.2 Activities

Identify withdrawal procedures

With the description and the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design) define the withdrawal procedures to be applied in case of stop of the service.

In the end, the output of this work will be withdrawal procedures for the system.

Analysts or programmers are those in charge of the definition of the withdrawal procedures to stop the system.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model definition.

Implement withdrawal procedures

With the description, the models of the systems (including requirements, the SOA model of the architecture, the Cloud model of the deployment, and implementation design), the withdrawal procedures implement the withdrawal procedures.

In the end, the output of this work will be running withdrawal procedures and the withdrawal information for the system.

Analysts or programmers are those in charge of the establishment of the running procedures infrastructure and the withdrawal information.

This activity has no automation possibilities.

From REMICS perspective, this activity is supported by specific tools developed in the REMICS for model transformation.

7.7.3 Tools

- UML modelling tools
- SoaML modelling tools
- PIM4Cloud modelling tools
- Code generation tools

7.7.4 Inputs

⇒ Complete Requirements (RDL)

- Use case model
- Iteration model

⇒ Soa Model (SoaML)

- Component Model

⇒ Cloud Model (PIM4Cloud)

⇐ Deployment Model

⇒ Component Design

⇒ Code stubs

⇒ Glossary

7.7.5 Outputs

⇐ Complete Requirements (RDL)

- Use case model
- Iteration model

⇐ Soa Model (SoaML)

- Component Model

⇐ Cloud Model (PIM4Cloud)

- Deployment Model

⇐ ImplementationDesign

⇐ Code

7.7.6 Support Material

- Requirement Template

- UML system model template

7.7.7 Workers

- Analyst
- User
- Programmer
- Tester
- Project Manager

7.7.8 Tips

☑ No tips in this version.

8 Lifecycle

The REMICS will be based on an iterative lifecycle with a similar philosophy of OpenUP. Depending on the phase of the project the methodology will provide different kind of iterations where the first ones will be focussed in the requirement identification and the last ones in the evaluation and supervision of the final system. The following figure shows the six types of phases expected in REMICS methodology: Requirement, Recover, Migrate, Test, Supervise and Withdrawal.

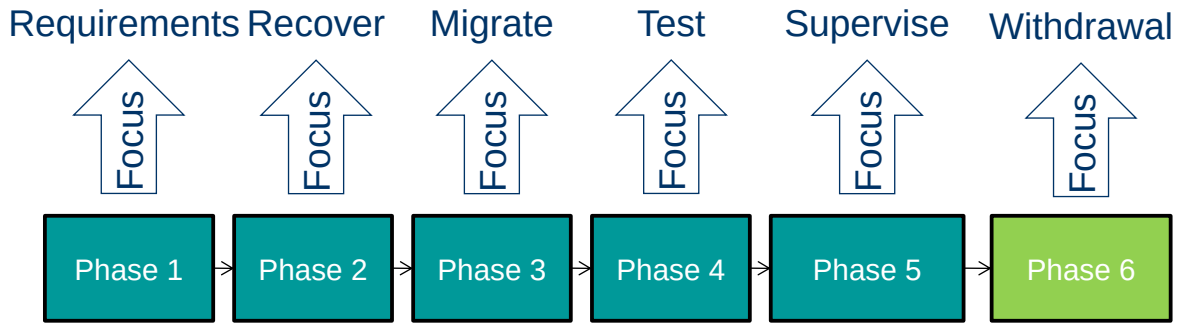

Figure 12 – REMICS Methodology Lifecycle Phases

Internally, each phase may implement all the activity areas of the methodology but with different priorities.

- **Requirements phase:** the focus in this first phase is the identification of the final system vision, the identification of the additional requirements, and establishment of the development (development, testing, preproduction, production) environment. It may also include the implementation of few critical features of the system, which may compromise the final system vision.
- **Recover phase:** the focus of this phase is the recovery of the knowledge of the components to be reengineered. This may include the implementation of features of the recovered system in the new infrastructure.
- **Migrate phase:** the focus is the delivery of all the features of the initial system plus the implementation of the additional system.
- **Validate phase:** the focus is to develop validation procedures to ensure that the developed components fulfil the application requirements.
- **Supervise phase:** the focus is to implement the supervise procedures and components. This may involve the introduction of additional requirements, components and validation procedures.
- **Withdrawal phase:** the focus is to implement the withdrawal procedures and components. This may involve the introduction of additional requirements, components and validation procedures. Withdrawal phase has a different colour as far as it will not be supported by the REMICS project as stated in 5.2.

For the implementation of each of the phases the REMICS methodology proposes a SCRUM or XP like approach where some features are chosen for implementation. In each of the Phases it will be possible to carry out one or more sprints. A sprint is an SCRUM concept that represents a closed period of time, usually two weeks, where some features of the final system are chosen to be implemented and are finalized. Anyway, this duration can be adjusted to the context of each migration project.

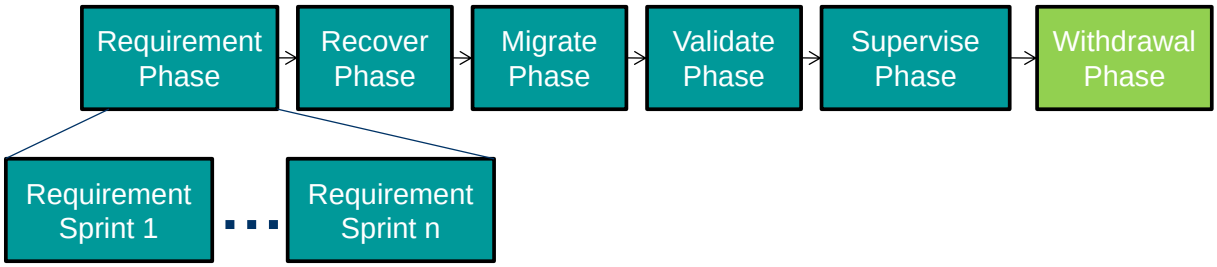

Figure 13 – REMICS Methodology Lifecycle Sprints

9 EPF Implementation

The REMICS Methodology is implemented in EPF following a layout similar to openUP. It is structured in packages to control the publication of the methodology and packages to gather the actual content of the methodology.

9.1 Presentation

Two packages control the publication of the methodology, these are core and publish. Core defines a general template for methodology presentation, containing information about the navigation of the methodology, the copyright information and EPF in general. While publish extends that publication schema adapting it to REMICS elements.

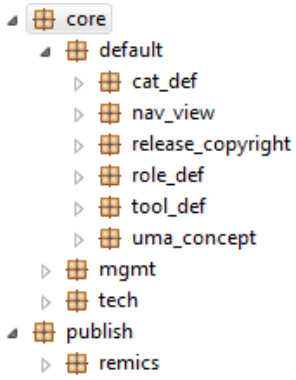

Figure 14 - Presentation template

Inside the nav_view package is where the navigation elements of the left of the screen are described.

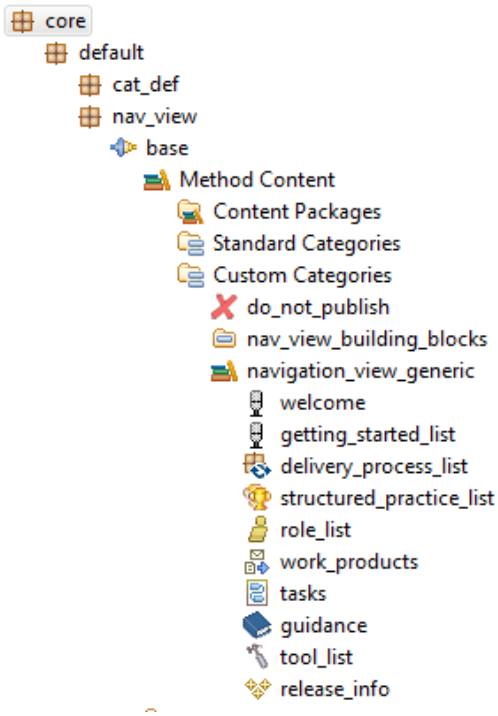

Figure 15 - Navigation template

The publish package redefines some of the navigation categories of the core presentation template to adapt it to the remics methodology.

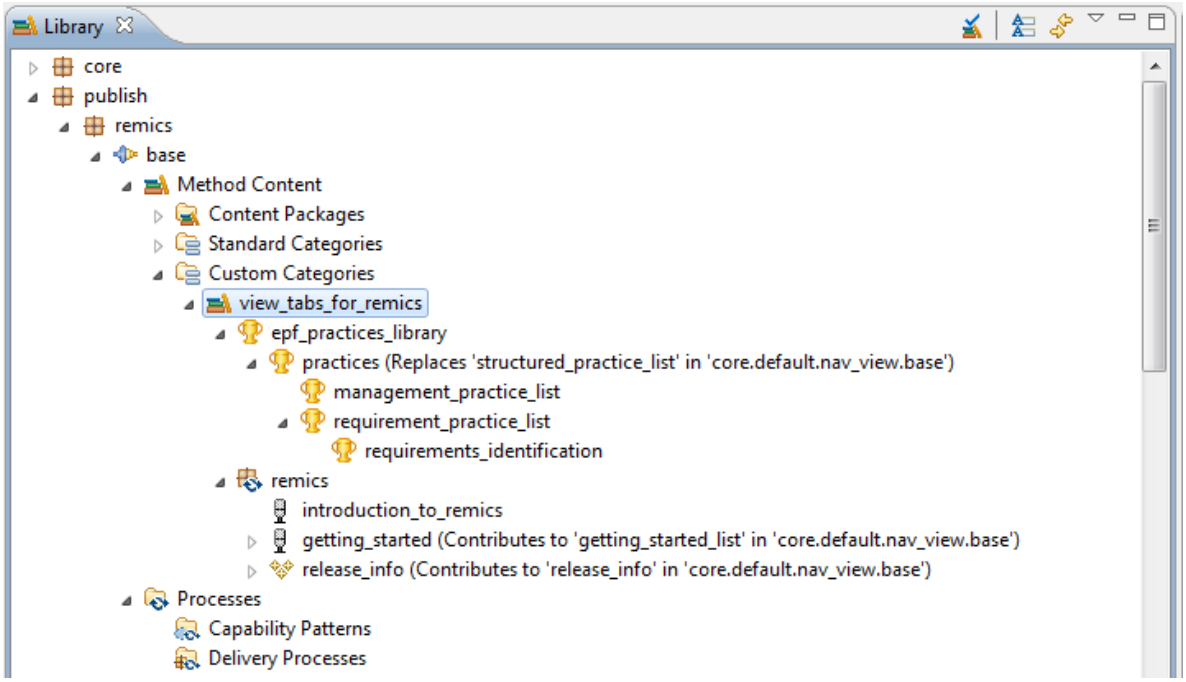

Figure 16 - Publish template

For example, practices replace the structured_practice_list of the general template to provide the ordered list of practices of the methodology.

9.2 Content

The actual content of the remics methodology and the contribution methodologies are provided in different packages. There will be one high level package that provides the integrated view of the methodology and multiple individual packages that provide information for its different parts.

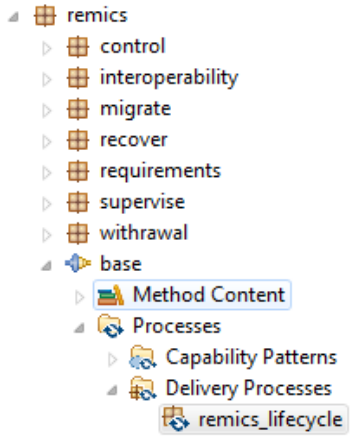

Figure 17 – High level remics package template

The remics.base is the package that contains the integrated view of the remics methodology. It is represended in the remics_lifecycle and it takes elements from the other packages (i.e. remics.requirements.base).

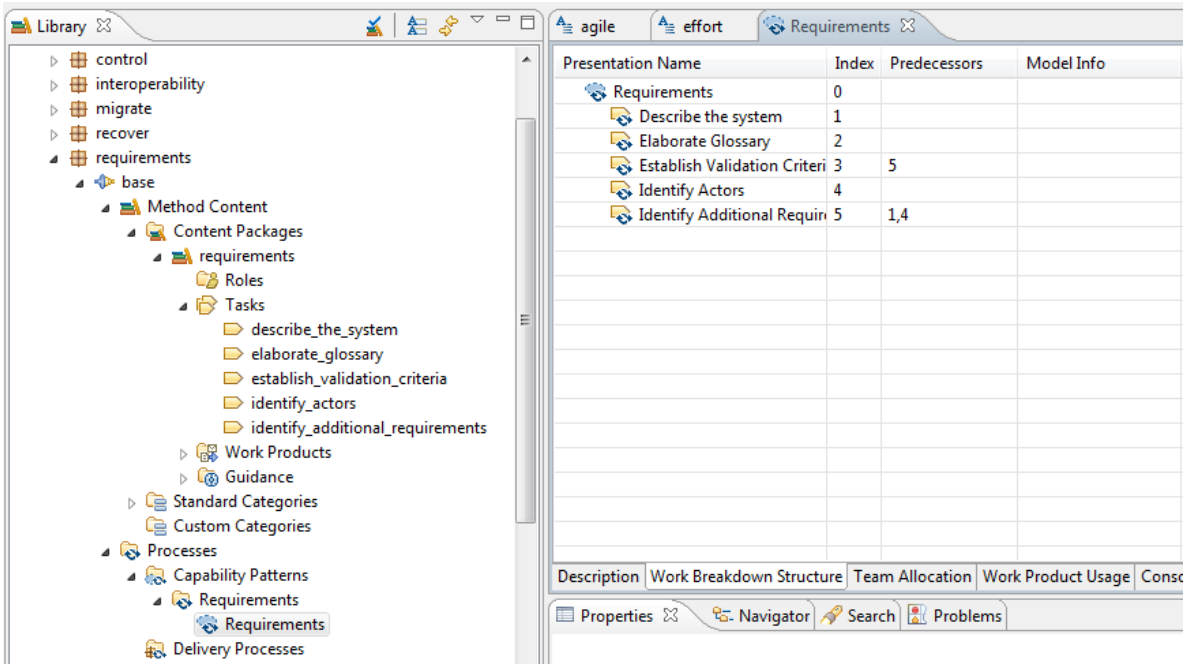

Figure 18 – Requirements package template

For example in the requirements package we define the requirements related activities and a sequence (Capability pattern) is provided for those activities.

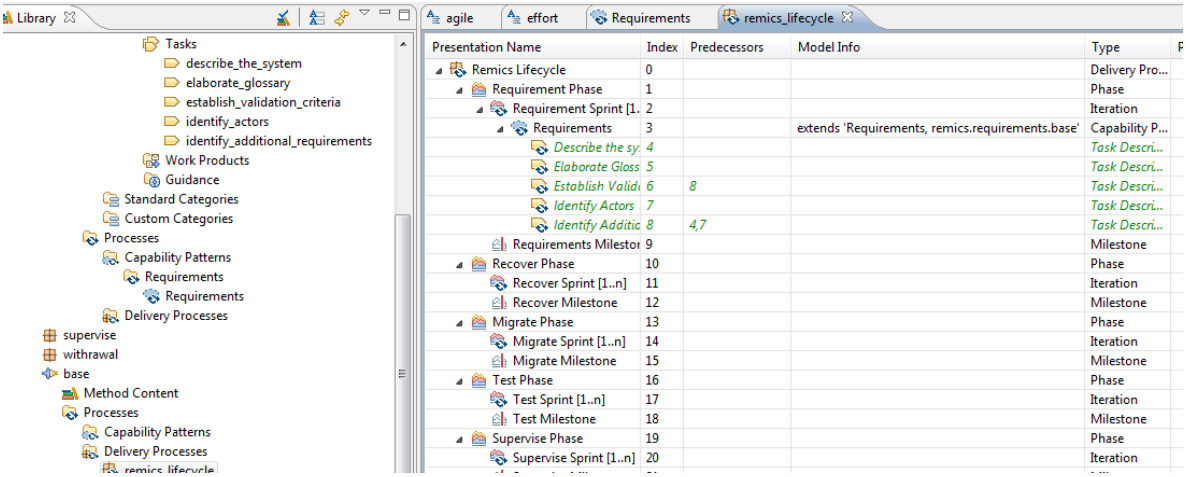

Figure 19 – Requirements capability pattern usage in the remics-lifecycle template

After that the requirement capability pattern can be reused in the definition of the remics-lifecycle.

9.3 Reused

When we want to reuse an existing methodology or methodology fragment inside the REMICS methodology, their content is included in a separate package. That package will include at least all the elements to be reused in the REMICS methodology, but in can also contain other elements if they are useful to improve the understanding of the reused methodology or methodology fragment.

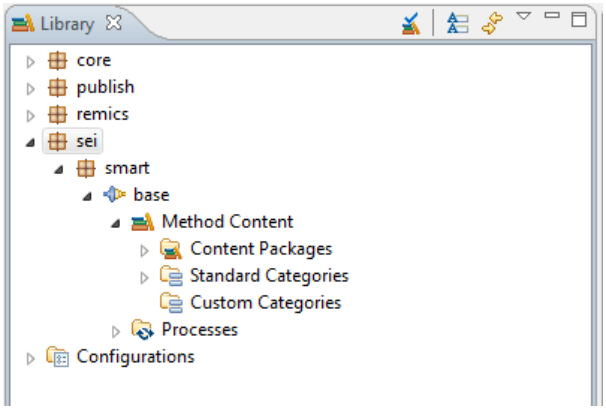

Figure 20 – Reused methodology fragment from SEI

10 Conclusion

The migration of legacy applications to cloud environment constitutes a new business need for software organisation in order to provide a better service to their customers. The usage of the cloud infrastructures may provide many benefits to the organisation, for example when increasing the system. Besides, the migration to a new deployment infrastructure is always a good opportunity for the organisations to update their legacy components to new trends in software development.

Within the REMICS project new techniques and tools are introduced to allow organisation to better deal with some of the challenges that these kind of project have to deal with. Unfortunately, there is no current methodology that covers the migration project in a complete way with a seamless integration of the later techniques and tools that may help in this task.

The current document presents a methodology to support the organisation in the adaptation of their legacy systems to the cloud. The methodology is characterised by a set of activities areas to be usually covered in this kind of initiatives and a recommended lifecycle that helps the organisation to manage with the lack of knowledge in this kind of projects.

The methodology covers seven activity areas:

- Requirements: focused in the additional requirements.
- Recover: focused in the recovery of the application logic from the legacy code.
- Migrate: focused in the definition and implementation of the migrated system, usually include the implementation of the SOA.
- Validate: focused in the implementation of validation activities over the migrated system.
- Supervision: focused in the implementation of monitoring and support features.
- Interoperability: focussed in the identification of interoperability issues and their solution.
- Withdrawal: focussed in the stop of the service in a managed way.

The methodology lifecycle provides an interoperability and incremental approach. It is organised in six main phases that may be split in one or more sprint in order to provide features in a periodic basis trying to avoid deadlock situations.

- Requirements phase: focused in the definition of the system and the identification of the additional requirements.
- Recover phase: focused in the recovery of the application logic of the components selected for reengineering.
- Migrate phase: focussed in implementation of the migrated system.
- Validate phase: focussed in the validation of the additional requirements and the main features of the system.
- Supervision phase: development of features to support the maintenance of the service.
- Withdrawal phase: development of features to stop the provision of the service.

This methodology will be later extended with guidelines on the usage of the different tools developed in the project to support the migration of applications. Besides, as the use cases are implemented it will get extended description of the activities and more tips for the different activity areas based on the lessons learnt from the application of the methodology and the tools in the use case.