

Project Deliverable D6.1


Method and Abstract Workflow

Project name:	Q-ImPrESS
Contract number:	FP7-215013
Project deliverable:	D6.1: Method and Abstract Workflow
Author(s):	Marco Masetti, Steffen Becker, Ivan Skuliber, Michael Hauck, Jan Kofron, Klaus Krogmann, Johannes Stammel, Cristina Seceleanu, Johannes Tysiak, Raffaella Mirandola, Danilo Ardagna
Work package:	WP6
Work package leader:	SFT
Planned delivery date:	M16
Delivery date:	15/05/2009
Last change:	14/05/2009
Version number:	1.0

Abstract

This deliverable defines an overall abstract workflow of the Q-ImPrESS working method. The main goal of the deliverable is to outline the working method, neither focusing on a specific change scenario nor taking into account a given domain. The perspective is on the overall workflow, divided in processes. Each process is described in terms of inputs/outputs and main actions. A proof of concept of the method is described by means of a basic example.

Keywords: method workflow, service architecture meta-model, quality prediction analysis, change scenario, trade-off analysis, quality annotations

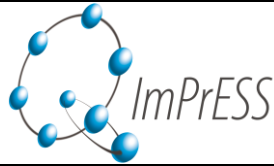
	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

Revision history

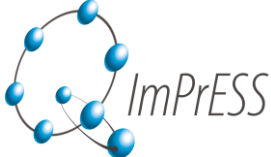
Version	Change date	Author(s)	Description
0.1	11/02/2009	M.Masetti	Draft version
0.2	27/02/2009	M.Masetti	Minor updates
0.3	05/03/2009	M.Masetti	Minor updates.
0.4	10/03/2009	M.Masetti	Deliverable structure.
0.5	14/03/2009	M.Masetti	Deliverable structure revised after partners comments
0.6	21/04/2009	M.Masetti	First peer review
0.7	22/04/2009	M.Masetti	Included updates from Michael Hauck to reference table process 3.3
0.8	22/04/2009	M.Masetti	Updated chapter 1.1
0.9	24/04/2009	M.Masetti	Chapter 4.5.3 added
0.10	27/04/2009	M.Masetti	New process diagram.
0.11	30/04/2009	M.Masetti	Updated sections 3.4.1, 4.x
0.12	11/05/2009	M.Masetti	Several minor updates
0.99	12/05/2009	M.Masetti	Pre-final release, last updates from Steffen and Johannes Tysiak
1.0	14/05/2009	M.Masetti	Final release

Table of contents

1	Introduction	5
1.1	<i>Purpose of this document.....</i>	5
1.2	<i>Status of this document</i>	5
1.3	<i>Related Documents</i>	5
1.4	<i>Motivations for a working method.....</i>	5
1.5	<i>Q-ImPrESS Project overview.....</i>	6
2	Users and change scenarios targeted by the project.....	7
2.1	<i>Q-ImPrESS tool landscape</i>	7
2.2	<i>Change scenarios.....</i>	8
3	The working method	9
3.1	<i>Gathering new requirements.....</i>	9
3.2	<i>Defining change scenarios.....</i>	10
3.3	<i>Modelling a change scenario.....</i>	10
3.3.1	<i>Modelling Non-Functional Requirements.....</i>	11
3.3.2	<i>Deciding on Model granularity</i>	12
3.3.3	<i>Managing the models of each single component.....</i>	12
3.3.4	<i>Modelling system assembly</i>	17
3.3.5	<i>Modelling system deployment</i>	17
3.3.6	<i>Modelling system usage.....</i>	17
3.3.7	<i>Adding quality annotations</i>	17
3.3.8	<i>Importing an external model into SAM.....</i>	17
3.3.9	<i>Validating SAM</i>	18
3.4	<i>Predict System Quality.....</i>	18
3.4.1	<i>Performance Prediction</i>	18
3.4.2	<i>Reliability Prediction.....</i>	18
3.4.3	<i>Maintainability Prediction.....</i>	19
3.5	<i>Performing a Trade-off Analysis.....</i>	21
3.6	<i>Implementing Service Architecture Model change scenario.....</i>	22
3.7	<i>Validating the Model by Measurements.....</i>	22
3.8	<i>Deploying the System.....</i>	22
4	Proof-for-concept of the method	24
4.1	<i>The basic use case.....</i>	24
4.1.1	<i>Client-Server Example</i>	24
4.1.2	<i>Client-Server Interaction</i>	25
4.2	<i>Process 3.1: Gather New Requirements for the test system.....</i>	26
4.3	<i>Process 3.2: Define implementation scenarios</i>	26
4.4	<i>Process 3.3: Model Implementation scenarios</i>	27
4.5	<i>Process 3.4: Predict system quality</i>	28
4.5.1	<i>Predict system performance</i>	28
4.5.2	<i>Predict system reliability</i>	28



4.5.3	Predict system maintenance	29
4.6	<i>Process 3.5: Perform trade-off analysis</i>	30
4.7	<i>Process 3.6: Implement change scenario</i>	30
4.8	<i>Process 3.7: Validate Model by Measurements</i>	31
4.9	<i>Process 3.8: Deploy updated test system</i>	31
5	Workflow Process Reference table.....	32
5.1	<i>Main workflow processes</i>	32
3.1	32
3.2	33
3.3	33
3.4	33
3.5	33
3.6	33
3.7	33
3.8	34
5.2	<i>Process 3.3: Model Implementation Scenario processes</i>	35
3.3.1	35
3.3.2	36
3.3.3	36
3.3.4	36
3.3.5	36
3.3.6	37
3.3.7	37
3.3.8	37
3.3.9	37
5.3	<i>Process 3.4: Predict SAM quality processes</i>	38
3.4.1	38
3.4.2	38
3.4.3	39
3.4.4	39
5.4	<i>Process 3.3.3: Manage Single Component's Model</i>	39
3.3.3.1	40
3.3.3.2	40
3.3.3.3	41
3.3.3.4	41
3.3.3.5	41
6	Glossary	42
7	Conclusions	43

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

1 Introduction

1.1 Purpose of this document

This deliverable defines an overall abstract workflow of the Q-ImPRESS working method. The main goal of the deliverable is to outline the working method, not focusing on a specific change scenario nor taking into account a given domain.

The perspective is on the overall workflow, divided in processes. Each process is described in terms of inputs/outputs and main actions.

A proof of concept of the method is described by means of a basic example.

This deliverable will guide method validation performed in WP7 and will be used as a basis to derive the operating guidelines for the specific domains.

The mapping between the processes and the tools adopted is not part of this deliverable but will be described in the following D6.2, D6.3 and D6.4.

This deliverable is organized as follows: chapter 1 provides an overview of the deliverable; chapter 2 describes the users and the change scenarios targeted by Q-ImPRESS, chapter 3 describes the generic working method; chapter 4 describes the workflow applied to a basic example; chapter 5 provides a reference table of all the processes listed in the workflow; chapter 6 lists a glossary of common terms. Conclusions are finally drawn in chapter 7.

1.2 Status of this document

Final.

1.3 Related Documents

- D1.1 – System Requirements
- D2.1 – Service Architecture Meta – Model
- QImPRESS_BasicExample.doc WP2/WP3 technical note

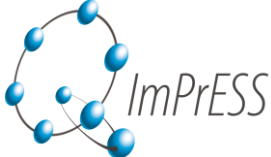
1.4 Motivations for a working method

The main goal of the working method is to guide the end-users of the Q-ImPRESS toolkit getting the results he/she is interested in. The Q-ImPRESS toolkit can be successfully used to tackle a broad range of change scenarios that cover most of the events that can happen during system life time. Each different change scenario will need a slightly adapted workflow to be tackled successfully by the Q-ImPRESS toolkit. Anyhow all these workflows can be seen as specialisations of the abstract method outlined in this deliverable.

The path the user will follow to reach his goal can be divided in several processes, the method clarifies how these processes are connected one each other, their requirements and what they do.

Moreover different users may approach the Q-ImPRESS toolkit with different goals in mind. As a result, the use of the Q-ImPRESS toolkit may require the application of some sort of guidelines that will help users reaching their goals.

Another consideration is time: applying the Q-ImPRESS method can be time-consuming, as some phases could require large computational efforts or several iterations involving users

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

with different capabilities (i.e.: system architects, system engineers, product managers); a method with accompanying guidelines should help tracking the current status and providing a rough estimation of the time to finish.

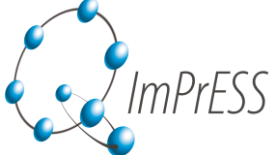
1.5 Q-ImPrESS Project overview

The main project goal is to facilitate the adoption of service-oriented architectures in large, complex and critical environments. To achieve this goal a set of tools to assess software quality with different metrics and to predict impacts of software changes is provided.

The Q-ImPrESS project will validate the toolkit in three different domains: Telecommunications, Automation and Enterprise.

The vision of the proposed project is to enable service oriented architectures with predictable end-to-end quality using analysis and simulation techniques for quality impact analysis.

This deliverable describes the generic working method that should be used in order to reach the desired goals.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

2 Users and change scenarios targeted by the project

This working method has been derived taking into consideration Deliverable D1.1 (Requirements Document) and Deliverable D2.1 (Service Architecture Meta-Model (SAMM) specification). The Q-ImPrESS toolkit aims at assessing the quality of complex, service oriented software systems, such as automation or telecommunication systems. These systems are also characterized by long life expectations, so usually a deployed system is kept for several years before being replaced with a brand new one. Usually software systems are replaced when system maintainability or performance degrade below an acceptable threshold. Such big systems usually involve the work of several people with different roles: Product Managers (business level decisions), Software Architects (responsible for the whole system - or a single sub-system- high-level architecture and work planning), Software Engineers (responsible for component design and implementation).

The Q-ImPrESS toolkit achieves its goal by building a Service Architecture Model (SAM) of the target system and performing analysis on the model. The adherence of the Service Architecture Model with the system it describes is crucial, as the predictability of the results heavily rely on this. The SAM creation process is described in details in the working method.

The system is embedded in a runtime environment that is not fixed but usually changes during system life-time; as runtime environment we consider the runtime platform the system is deployed upon, the system users and third-party service providers the system interacts with; these changes lead to new requirements, which produce change demands for the system.

The runtime environment is not the only responsible for change requests: for example new requirements can come from the Product Manager, as the competitors' solutions evolve in time and the product under analysis has to cope with new features required by the market.

Q-ImPrESS addresses the impact of these changes on the system quality, such as system performance, reliability and maintainability.

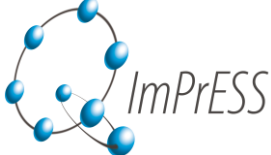
No matter what caused them, changes can affect the topology of the software system (assembly change scenarios), the deployment environment (allocation change scenarios), or the user behaviour (usage change scenarios). Refer to chapter 2.2 for a (not exhaustive) list of use cases targeted by the Q-ImPrESS platform.

2.1 Q-ImPrESS tool landscape

The Q-ImPrESS method is accompanied by different tools. Most of these tools will be integrated within a GUI application.

The tools landscape is represented by the following tool classes:

1. Reverse engineering tools (SISSy/SoMoX/ArchiRec) to analyse and inspect software source code (of white-box components) in order to derive a standard, vendor-independent partial service architecture model;
2. Tools that derive system behaviour from system execution;
3. Model editing tools to update system models;
4. Tools to derive and solve prediction models for quality attributes of services (i.e., performance, reliability and maintainability);
5. Tools for model transformations between the different tools provided including

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

- methods to ensure consistency and traceability between tools specific models;
6. Tools for analysis and simulation to make predictions on the quality attributes of services;
 7. Tools for trade-off analysis;

2.2 Change scenarios

The Q-ImPRESS method targets changes in service-oriented software systems. These changes can be divided in *assembly change scenarios*, which affect the topology of the software system, *allocation change scenarios*, which affect the deployment environment and *usage change scenarios*, which result from evolving user behaviour. The following examples illustrate concrete change scenarios. They can also be found in Chapter 4.4.3 of Deliverable D1.1 (Requirements Document).

Assembly change scenarios affect the implementation and topology of the software system itself. This covers new services being designed and implemented from scratch. Changing requirements may also impose the need to change existing services. Another possible concrete change scenario is the change in service granularity. One example for this is a service that is partitioned into several fine-grained services. Also the wiring between services may change. For example a single instance of a service is replaced by multiple service instances and a load balancer.

Allocation changes cover changing hardware and middleware next to the allocation of services on the hardware or middleware. The used hardware resources may change due to evolving technology. A middleware may be replaced by a newer version. It may also be necessary to change the allocation of software system components on the hardware or middleware resources, e.g. deploying two services, which previously ran on the same machine on two different machines.

Usage change scenarios are also targeted by the Q-ImPRESS method and tools. An emerging IT enterprise might experience a sudden rise in the number of users which concurrently use the software system. Also, the characteristics of the usage may change. One example for change in usage characteristics may be an audio and video download service, where users previously concentrated on downloading smaller audio files, but now more prominently download large video files.

The above list of examples is not intended to represent a complete enumeration of change scenarios. There may be further examples for change scenarios which are also covered by the Q-ImPRESS method and tools.

3 The working method

In this chapter the working method is thoroughly investigated and explained.

The main workflow can be divided in processes. Each process encompasses a closely related list of activities.

The following picture depicts the main workflow; each box represents a process, grey boxes tag activities not taken into consideration by the project, box numbers match chapters ones:

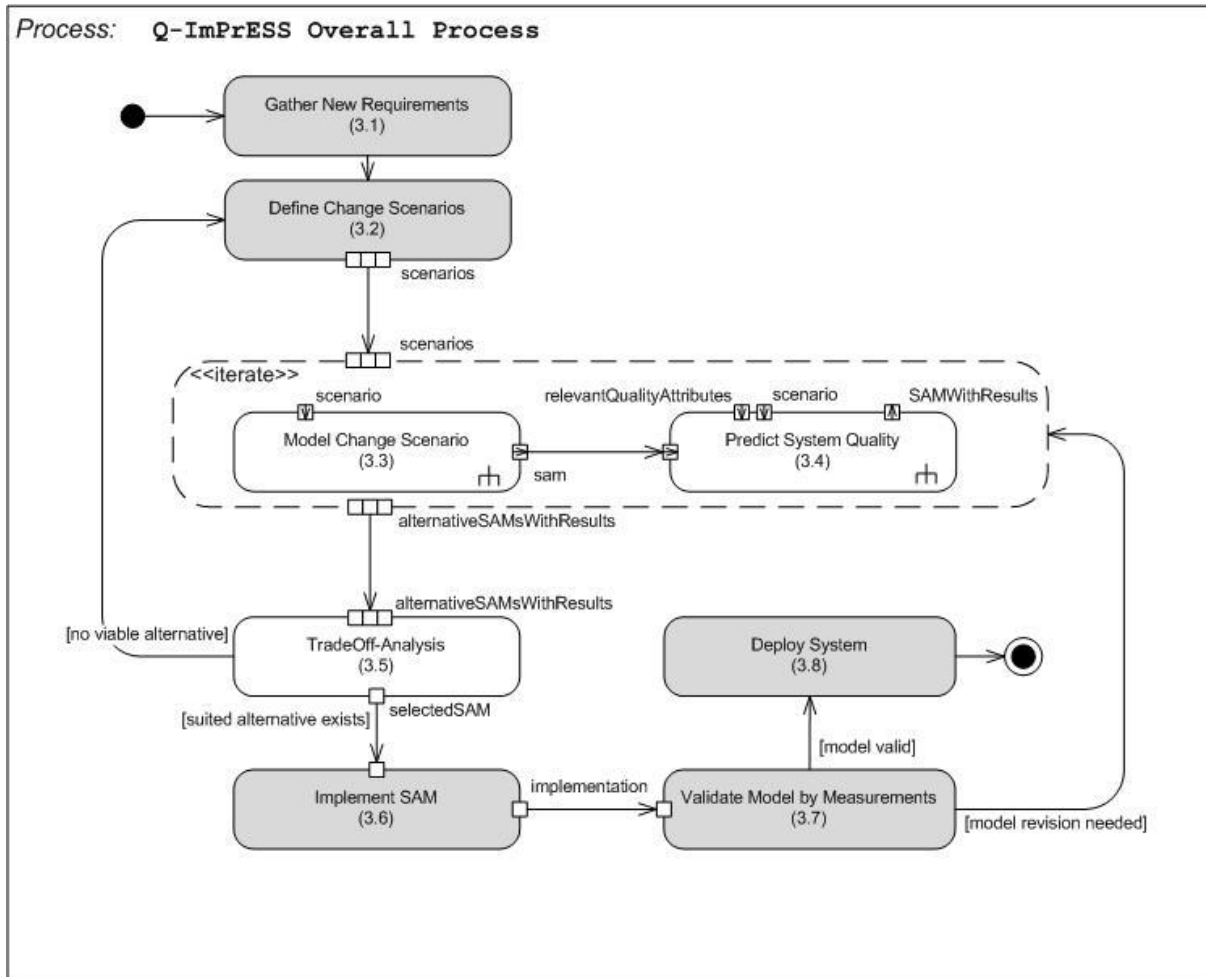


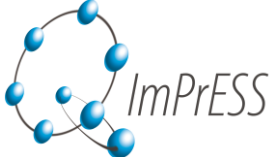
Figure 1 - The Q-ImPRESS Overall Process

3.1 Gathering new requirements

The workflow starts with the collection of new requirements. Usually the product manager collects new requirements. These requirements could for example come from the customers or could be defined after a comparison with similar products from competitors.

Any new requirement leads to a change in the system. When the impact of the change on the system quality is not predictable, or when more than one potential change scenario can be followed, the impact of the solution can be assessed first using the Q-ImPRESS method.

The Q-ImPRESS method does not support with any tool the collection of requirements, but this process triggers anyway the entire workflow.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.2 Defining change scenarios

A brief analysis of these requirements leads to the selection of the change scenarios to assess. This activity is performed by the product manager with the help of the System architect as deep knowledge of the system is needed. The System architect proposes some alternative solutions to meet the requirements. These alternative solutions will be modelled and their impact analysed by the Q-ImPRESS toolkit. These first introductory activities are performed several times during a product life cycle and, up to here, do not require any use and knowledge on specific Q-ImPRESS tools.

3.3 Modelling a change scenario

The Q-ImPRESS toolkit can be used to easily identify the implementation alternative that fits best the required quality of service, without having to proceed with the implementation of each of them. This is the most prominent added value in using the Q-ImPRESS working method compared to proceeding the usual way.

Even if the System architect does not have to know all Q-ImPRESS platform internals to use the Q-ImPRESS toolkit, the knowledge of the main abstractions and ideas at the base of Q-ImPRESS is necessary in order to understand the overall workflow.

To predict the effects of a change scenario on quality attributes, Q-ImPRESS analysis tools use a model of the system, the Service Architecture Model (SAM).

Any change scenario (either usage, assembly or allocation) leads to an update of the Service Architecture Model of the system under analysis.

Actually, the modelling of the system is split in two levels. Each analysis tool has its own internal representation of the system information, specialized for the tool goal. This specialized system model representation can be seen as the result of a transformation over a common Service Architecture Meta-Model (SAMM). Model-to-model transformation from SAMM to a tool prediction model is automatic and performed behind the scenes (using a standard model transformation engine like QVT). Automatic Model-to-model transformation in the other direction (from tool prediction model back to SAMM) is not foreseen at the moment.

Deliverable D2.1 (Service Architecture Meta – Model) explains in details the SAMM specification.

Figure 2 represents the workflow to be followed to model a change scenario:

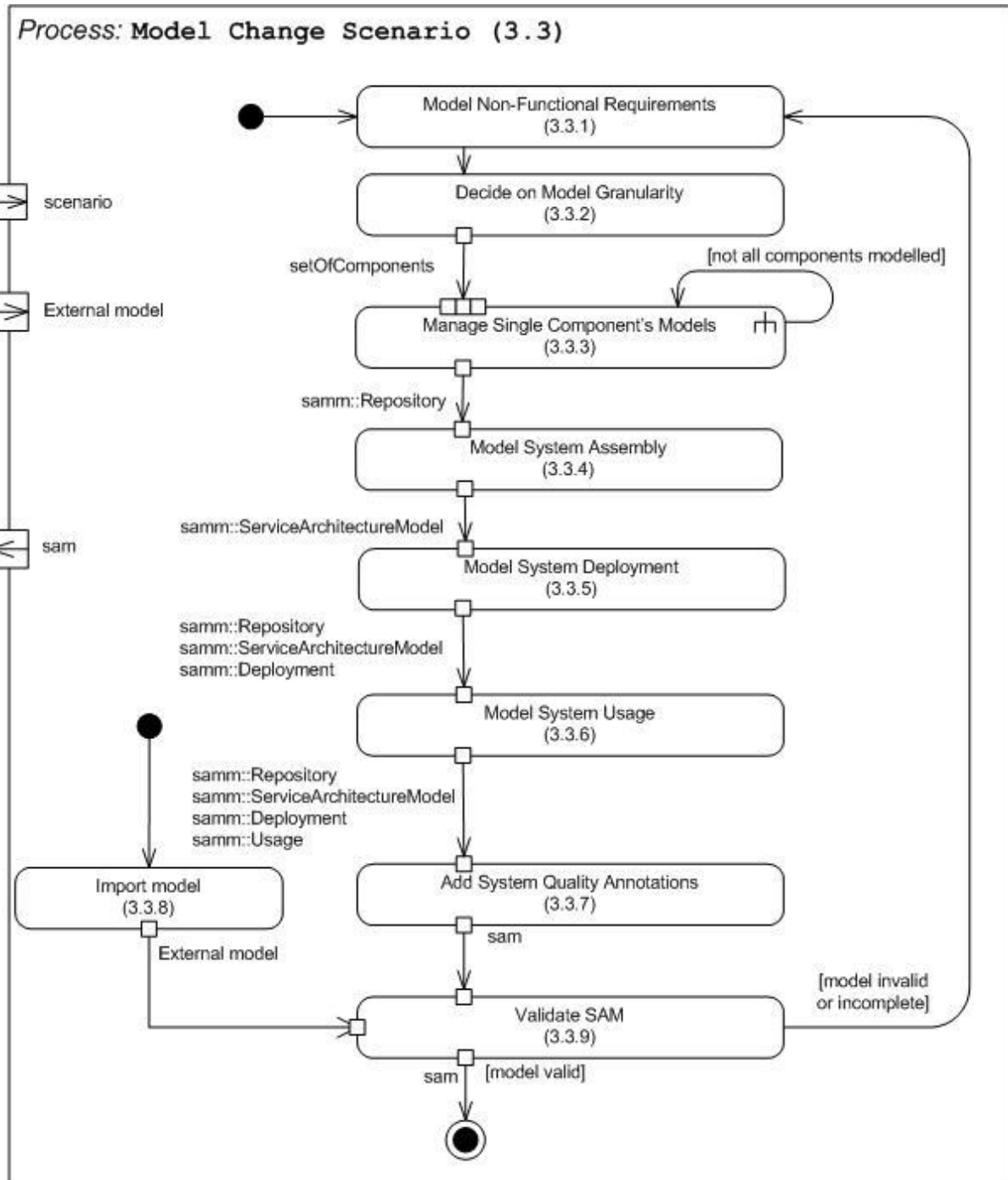
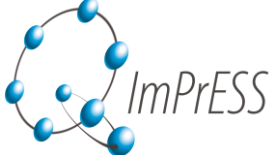


Figure 2: Modelling change scenario workflow

This process will be detailed further in the following sub-chapters.

3.3.1 Modelling Non-Functional Requirements

Updating the service architecture model to adhere to an implementation scenario means starting with listing some non-functional requirements.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.3.2 Deciding on Model granularity

The system architect has to decide on the granularity of the model: there is of course a trade-off between the costs of modelling and the accuracy and predictability of the results we get back from the system. Systems architecture can be modelled as a composition of components, connected by connectors (see the glossary for a definition of terms). Systems addressed by Q-ImPRESS are composed by a large set of components. An exhaustive description of each component of a complex system could be simply useless if, from the perspective of the prediction analysis to perform on the system, some of them could be modelled as black-box components *without affecting the behaviour of the components under analysis*.

For complex systems as the ones addressed by Q-ImPRESS, it is envisaged that the product manager starts the workflow giving a high level description of the system, detailed further by the system architect. If reverse-engineering is adopted to generate white-box models of components, this process can be iterated. In each iteration, the SoMoX tool tries to detect more abstract structures in composed components guided by the user's input.

Different components can be modelled with different granularity depending on their impact on the quality attributes of the system. For example, some core and crucial components could require to be modelled in more detail than others.

3.3.3 Managing the models of each single component

Modelling the system means modelling each single component and updating the model of those components affected by the assembly change scenario currently under analysis.

This can be a time consuming process (some hints on processes time requirements are listed in the reference section), although the service model can be saved and reused in the future, this means that, after a while the Q-ImPRESS method is adopted, this process should be limited to updating the model of the few components affected by the change scenario, this should greatly limit the overall workflow time consumption.

Basically, the workflow followed for the modelling of a component depends on the availability of the component's source code.

Figure 3 outlines the workflow for component modelling:

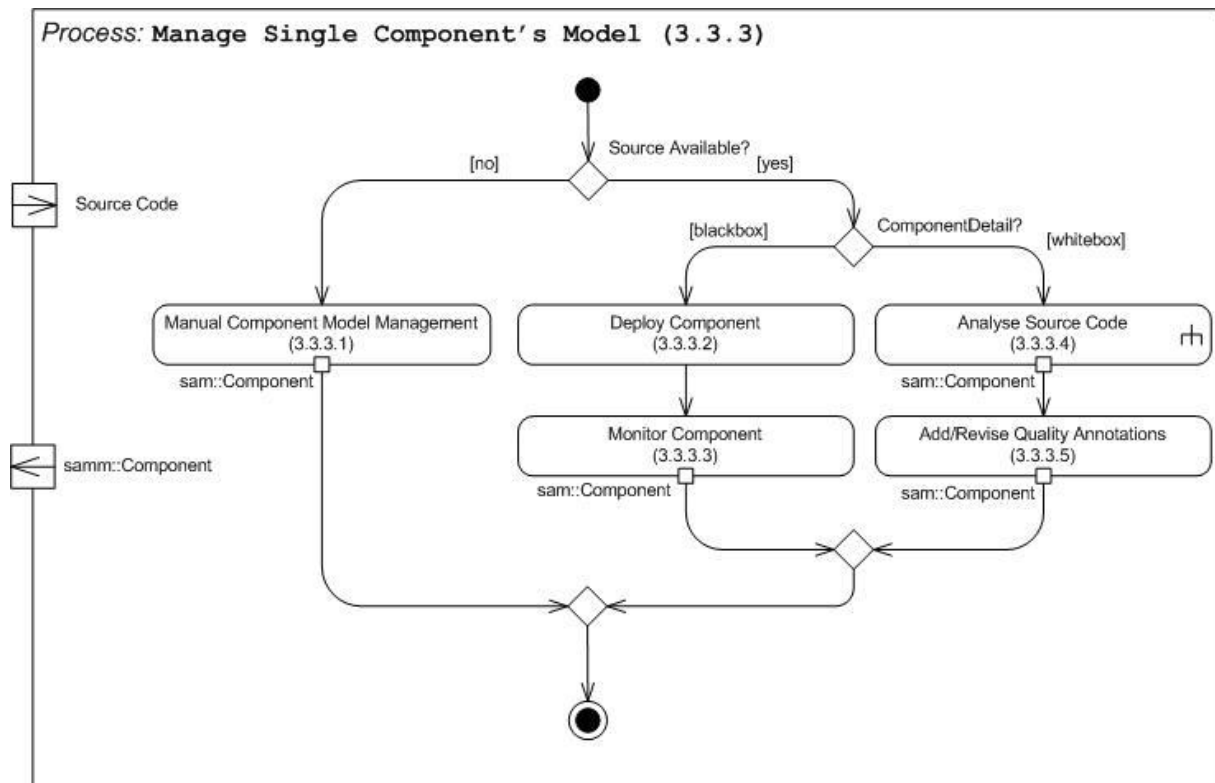


Figure 3: Managing single Component's Model

Workflow splits depending if component's source code is available or not.

If source code is not available then the component model has to be managed manually (process 3.3.3.1).

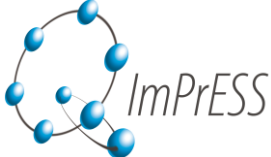
If source code is available then the system architect has two options depending on the level of details he wants to have for the component: he can deploy (process 3.3.3.2) and monitor (process 3.3.3.3) the component in order to semi-automatically get a model of the component useful for the prediction analysis (but with no details on the component's structure), otherwise he can perform a source code analysis (process 3.3.3.4), getting information on the component static structure as well, but with potentially the need to manually adjust the quality annotations (process 3.3.3.5). Q-ImPrESS will provide a textual editor for editing quality annotations (with improved functionalities like contextual syntax highlighting, code completion and context awareness).

Each process is described in detail in the next sub-chapters.

3.3.3.1 Manual Component Model Management

If the source code of the component is not available, the system architect has to model the component manually. The system architect has to know the system and components very well and has to have some experience on how the components behave to successfully perform this process.

In order to create component models manually, the system architect used editors provided for SAMM instances in the Q-ImPrESS toolkit. These editors allow modelling of component details like provided or required interfaces, component behaviour, etc.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.3.3.2 Component Deployment

For creation of some non-functional attributes of a black-box component, the application components have to be deployed and executed. System architect then uses a monitoring framework provided by Q-ImPRESS or his own monitoring infrastructure, which may not be, however, supported in the sense of conversion of the measurement data into SAMM quality annotations. Depending on the monitoring framework and type of the non-functional attributes, before deployment, the components of interest need to be instrumented with the monitoring frameworks to allow the planned quality analysis. For example, for performance prediction, the data on amount of loop iterations and the requested CPU time are to be collected and a corresponding instrumentation is required. Afterwards, the component (usually entire component application) needs to be deployed and executed. This can be done either in a test-bed environment with an artificial workload (i.e., component parameters and number of concurrent accesses) or using the real workload from a real running system. Getting the correct instrumentation points may be a time demanding task and expertise in performance analysis a requirement on the system architect. However, the instrumentation can be refined later and the monitoring being re-run if the model is found to be inaccurate. The deployment process is, however, not directly targeted and supported by the Q-ImPRESS tools.

3.3.3.3 Component Monitoring

Monitoring methods get information from software systems at runtime. Tools available for component monitoring follow different methods: code instrumentation, runtime monitoring, logging, or trace analyses. The system architect can use any monitoring framework, some glue have to be implemented to import monitoring results into SAMM (this is out of the project scope).

3.3.3.4 Source Code Analysis

If the component source is available, the system architect can decide to use reverse engineering techniques to extract the component model.

Reverse engineering integrates into the overall Q-ImPRESS as an alternate model creation step. It can also be applied for some of the available components or in scenarios where not for all components source code is available. In these cases, the remaining components need to be modelled manually.

The process is detailed in figure 4:

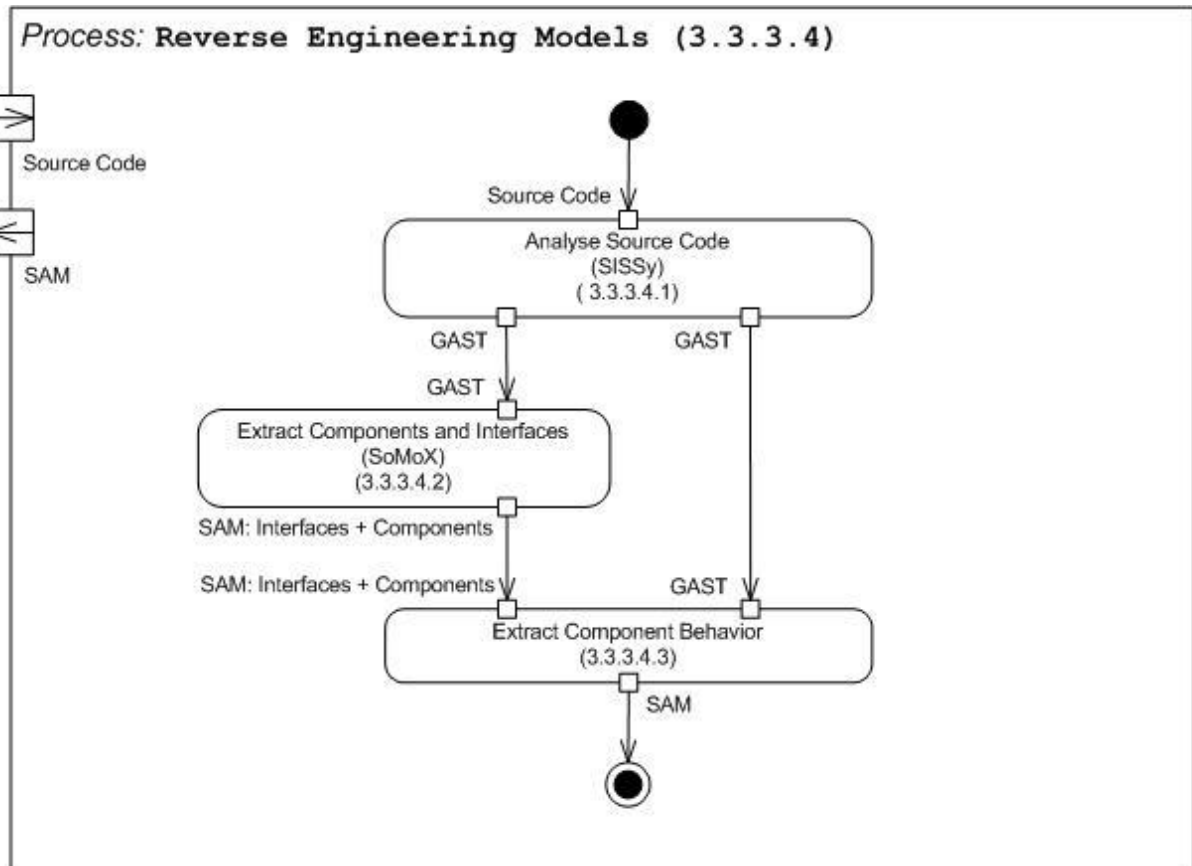


Figure 4: Reverse Engineering Models

3.3.3.4.1 Source Code Analysis with SISSy

The tool SISSy (Structural Investigation of Software Systems) will be integrated into the Q-ImPRESS platform and used for source code analysis. SISSy is a platform for problem pattern identification in OO source code written in Java, C++ or Delphi. SISSy produces a G-AST, a language independent model of the source code explained in detail in Deliverable D2.1 (Service Architecture Meta-Model (SAMM)).

3.3.3.4.2 Component and interfaces extraction with SoMox

SoMoX (Software Model eXtractor) is another tool provided with the Q-ImPRESS toolkit. It identifies the static structure of components, including their nesting and interconnections. It takes as input the output of SISSy and produces a SAM instance containing the static structure of the software. To detect parts of the static structure, such as components, various metrics are used.

3.3.3.4.3 Component Behaviour extraction

This process aims at extracting a component behaviour, in terms of interaction flow with other components.

Figure 5 depicts again the reverse engineering process to derive a component model from its source code:

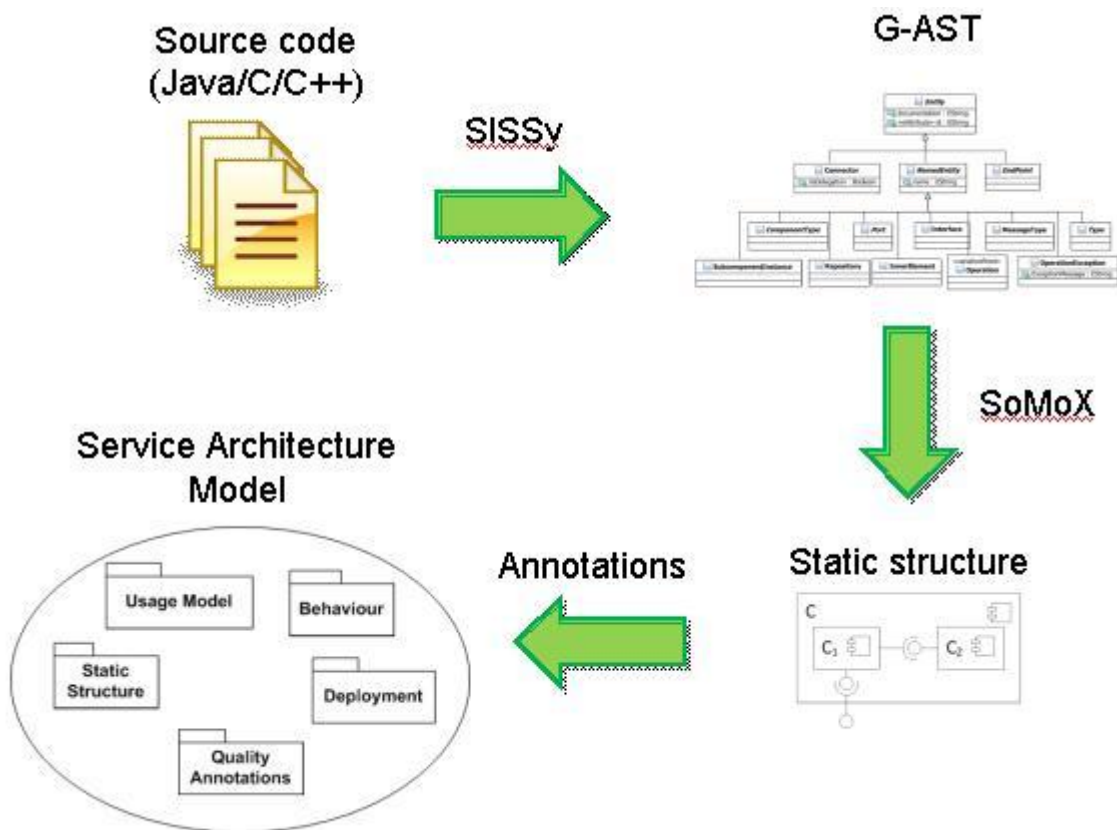
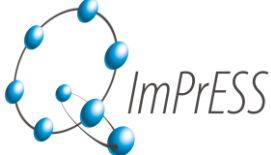


Figure 5: Source code Analysis with SISSy and SoMoX

3.3.3.5 Quality Annotations Management

Quality annotations model quality relevant properties of the system, e.g., resource demands, loop iteration counts, arrival rates. They include the type of the annotation, i.e., whether it is a requirement, an estimated value, or a value taken measured on the running system. The component developer or the system architect can provide the annotations manually by editing them in the dedicated editor. Alternatively, they can be collected by the monitoring tools. The quality annotations need to be attached to other elements of the SAMM. Therefore, they can only be added to model elements which already exist in the SAMM. Annotations can be updated as soon as the implementation or measuring of the system advances and new data is available.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.3.4 Modelling system assembly

At this point models for individual components are available. Now they need to be assembled to represent a Service Architecture Model.

Hence the system architect takes components from repository and plugs them together using connectors.

If assemblies are already present software architect can adapt them (e. g. exchange components).

3.3.5 Modelling system deployment

Before the system architect can proceed with the prediction analysis, two more processes have to be performed: a *model of the deployment environment* (in terms of allocation schema and resources) has to be provided along with a *model of system usage*.

The deployment model needs to be updated if the assembly change scenario involves the creation of new components (that need to be allocated), or in case of the HW resources are changed.

3.3.6 Modelling system usage

The workload intensity caused by the users of the service oriented system can be potentially extracted during the runtime monitoring process, anyhow the need for a manual update of this model is foreseen and therefore included in the workflow.

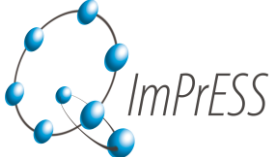
3.3.7 Adding quality annotations

Once all models are updated, quality annotations are added by the system architect.

Quality annotations are a type of information directly attached to elements in a SAM. This information is needed for the prediction of runtime properties (performance and reliability). Examples of quality annotations can be resource demands caused by certain steps in the architecture or loop iteration counts. Moreover quality annotations specifies the source of the annotation, i.e., whether the value is a requirement, a predicted value, an estimation, or a measured value. A predicted value stands for the result of the prediction process, i.e., it is an output of the method, while an estimation is a user's guess of the value, i.e., it might be an input parameter of the method. If it is a measured value it should contain information on the execution environment (see following paragraph) and usage model used to measure it.

3.3.8 Importing an external model into SAM

The Q-ImPrESS tools also enable the System Architect to reuse existing models. Existing SAMM instances which were saved externally, may be imported into the Q-ImPrESS toolset. It is also possible to import models which are instances of other meta-models (e.g. UML models or instances of Ecore-based meta-models). In order to import these models, the System Architect has to transform these models to an instance of the SAMM using external model-to-model transformation facilities, such as openArchitectureWare or QVT.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.3.9 Validating SAM

Before the start of the prediction process, the Q-ImPrESS platform automatically runs a consistency check and displays model inconsistencies that have to be fixed in order to proceed to the following activities.

3.4 Predict System Quality

The following process is the prediction of the quality metrics with respect to the updated Service Architecture Model (3.4). This process will be detailed further in a following chapter. Processes 3.3 and 3.4 will be iterated for each defined change scenario, results will be saved along with the model of the implementation alternative.

3.4.1 Performance Prediction

Having a SAM, the user can start a performance prediction using a transformation into the Palladio Component Model (PCM). For this, the SAM has to be complete, i.e., all components are modelled, the service architecture is available, it is allocated and its usage model is available. Furthermore, all model elements have the needed quality annotations. The Q-ImPrESS tool set will check whether the model is valid which mainly means that all model elements and all needed quality annotations are available. The transformation to PCM and the resulting system simulation are executed transparently for the Q-ImPrESS method's user.

As a result, the PCM's performance prediction annotates the SAM model with average response time distribution, resource utilisation distribution, and an estimated system throughput. Any of these predicted values can be used in an succeeding trade-off analysis.

3.4.2 Reliability Prediction

Q-ImPrESS toolkit uses KLAPER for the reliability prediction analysis.

KLAPER (Kernel LAnguage for PErformance and Reliability analysis) is a kernel language which can be used as starting point to carry out performance or reliability analysis. KLAPER adopts a single model of the system, which can carry different kinds of additional information to support the analysis of different attributes (of performance or reliability). In the Q-ImPrESS scope, KLAPER is used to support only reliability prediction. This information can be used by the system architect to verify if a given system architecture satisfies reliability constraints or to evaluate multiple candidate design alternatives for trade-off analysis.

Reliability is a measure of the continuous delivery of correct service, or, equivalently, of the time to failure and in the Q-ImPrESS framework is evaluated as the probability that the system performs its required functions under stated conditions for a specified period of time. Furthermore KLAPER allows working at a finer grain obtaining the probability that any subsystem or a set of them successfully complete a given service invocation (see Deliverable D3.1).

To support the reliability analysis, KLAPER uses information embedded in models expressed in SAMM, decorated with quality annotations such as usage profile and component failure

rates. At the end of the analysis KLAPER provides reliability values of components and systems nodes.

3.4.3 Maintainability Prediction

The maintainability prediction has the goal to guide a software architect to estimate the effort which is necessary to implement a given change request in the architecture described by an architecture model.

Therefore the maintainability prediction process takes as inputs the architecture model, i. e. an instance of the SAMM and the description of a change request. The description of change request is represented as scenario in the overall workflow.

The workflow of maintainability prediction process is depicted in Figure 6. The following paragraphs explain the process steps and sub processes in more detail.

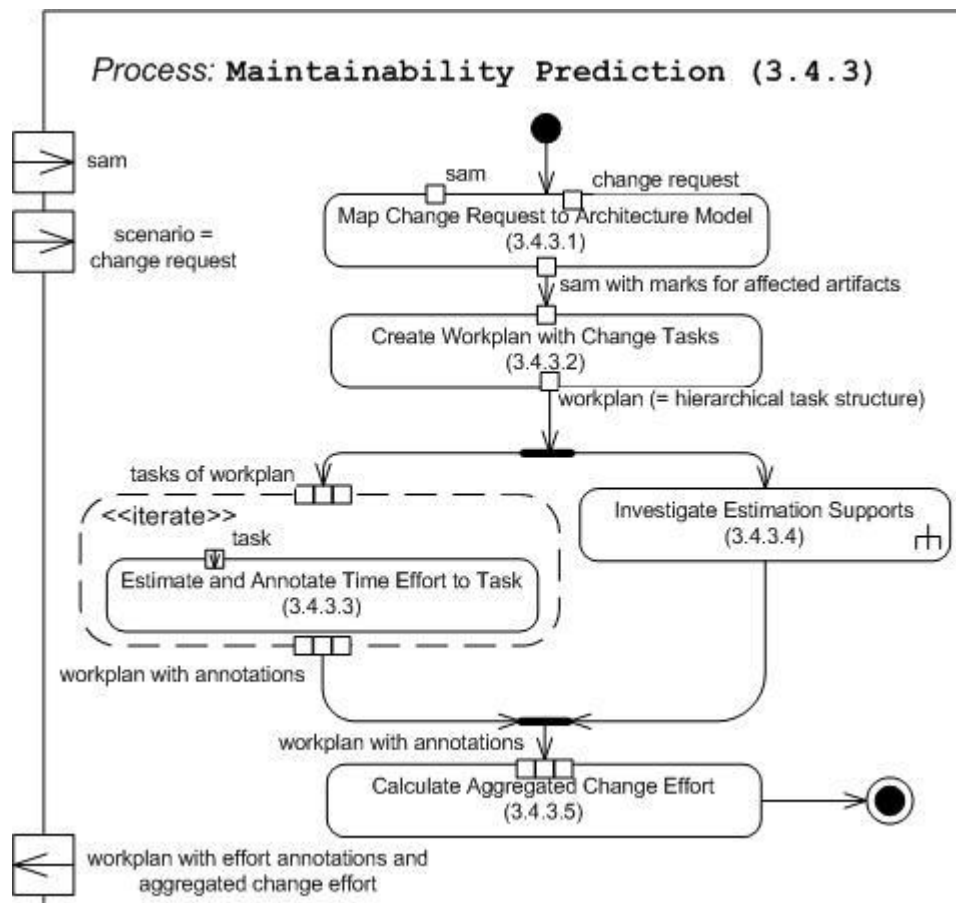


Figure 6: Maintainability Prediction Analysis

3.4.3.1 Map Change Request to Architecture Model

The software architect maps the given change request to the architecture. The software architect identifies parts in the architecture model which are affected by the change and marks them.

3.4.3.2 Create Workplan with Change Tasks

The software architect identifies basic steps which describe changes in the architecture that fulfil the change request. These basic steps are modelled as hierarchical tasks which form a workplan.

3.4.3.3 Estimate and Annotate Time Effort to Task

The software architect annotates each task with an estimation of work time necessary for execution of this task.

3.4.3.4 Investigate Estimation Supports

In order to facilitate and improve estimations the tool provides to the system architect some estimation supports. As illustrated in Figure 7 these consist of investigation of team organization, investigation of architectural properties and investigation of development environment.

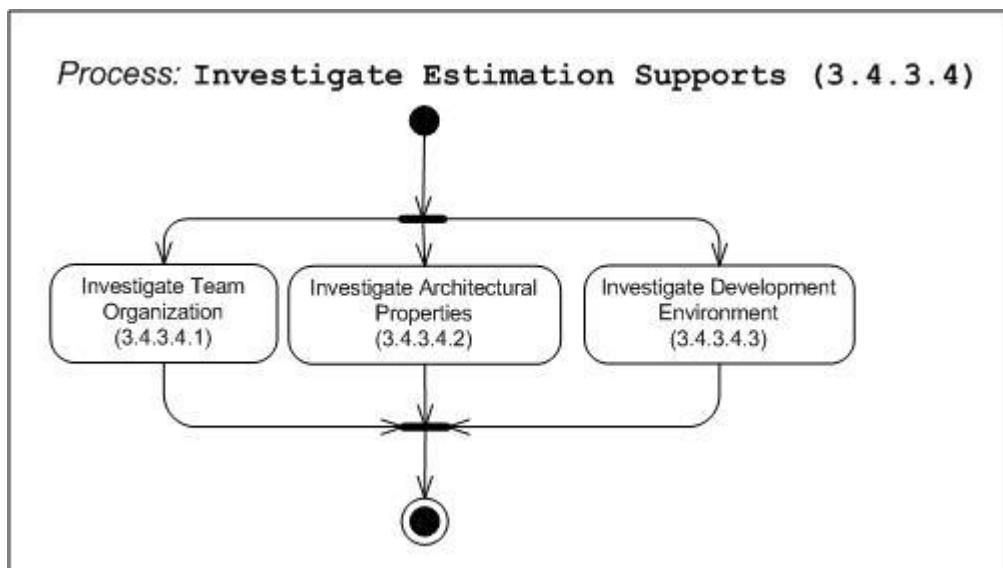
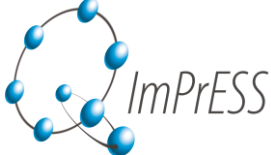


Figure 7: Investigate Estimation Supports

Investigate Team Organization: The tool guides the software architect in creation of a team organization model, which covers the department structure of a company, the team structure within departments, the roles which are present in these teams (e. g. software architect, software developers, etc.) and the association to people which perform these roles. Once a team organization model is created its parts are mapped to the work plan. This mapping

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

enables architects to see whether a workplan involves more or less complex parts of the team organization.

Investigate Architectural Properties: The tool guides the software architect to investigate properties of the architecture which influence the change effort. These consist of a couple of metrics, patterns, and antipatterns which are calculated or identified.

Investigate Development Environment: The tool guides the software architect to gather information about the development environment. The development environment usually influences the change effort because it determines which activities can either be automated or at least are facilitated by tool support or otherwise have to be done manually. It also determines the effort necessary for post-change activities (e. g. building, testing, debugging, deploying).

3.4.3.5 Calculate Aggregated Change Effort

Once a workplan with annotated effort estimates is present the aggregated total effort can be calculated.

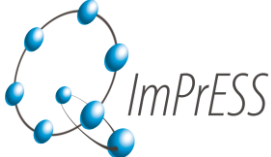
3.5 Performing a Trade-off Analysis

Quality prediction results are compared in the Trade-Off Analysis Process. To perform a trade-off analysis, the designer needs the following inputs: the architectural design alternatives, together with the predicted values of the QoS attributes. In addition, inputs pertaining to various usage profiles should be specified, such as utilization functions, size of data, loop counts etc., as well as the change scenarios. Under these inputs, a model builder generates an optimisation model, which is then fed to a model solver that generates the Pareto curves needed for determining the trade-offs between maintainability, performance and reliability obtained via approximated techniques. Based on such curves, the most appropriate architectural design alternative is chosen.

The result of the trade-off analysis could be that none of the proposed alternatives actually meets the requirements, and the overall process will have to continue again with 3.2, defining new implementation scenarios.

Let us assume that the prediction analysis gives us a set of QoS values as its outputs. We are interested in finding the optimal pair (\mathbf{A}, \mathbf{Q}) , where \mathbf{A} models the current architecture and \mathbf{Q} a vector of QoS attribute values. By changing elements of the architecture, like introducing or replacing a component, we will get a new architecture \mathbf{A}' . Also, \mathbf{A}' can be obtained by performing changes as, e.g., varying the range of a variable. In such cases, $\mathbf{A}' = \mathbf{A} + \Delta\mathbf{A}$, **where** $\Delta\mathbf{A}$ encodes the architectural change.

The trade-off analysis then subsumes two steps: (1) generate the Pareto curves that let the designer select the best architectural alternative(s) by determining the trade-offs between QoS attributes, and (2) find a ranking of the selected solutions by encoding the cost of the change

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

scenarios as a weighted sum of QoS attribute prediction values, in which the weights represent the ponder given by the designer to each QoS attribute.

If the trade-off analysis finds an implementation scenario fulfilling all the requirements, the system architect can proceed as usual, detailing the implementation scenario and guiding its implementation by the system engineers.

After having implemented the chosen implementation scenario the system architect has to check that the system actually matches the updated Service Architecture Model. If the validation is not positive, the system architect has to update again the Service Architecture Model.

If the model is valid, then the updated system can be deployed (Process 3.8). This ends the working method.

3.6 Implementing Service Architecture Model change scenario

After the system architect has selected the change scenario which best meets the requirements, he can proceed with the implementation. The system architect therefore creates basic implementation stubs for the new code. He has the choice to either do this manually or use model driven engineering techniques, where parts of the Service Architecture Model or other models can be used to generate code artefacts.

Having created these basic code artefacts, the system architect can delegate the responsibility for parts of the new system architecture to system engineers, who implement the necessary changes.

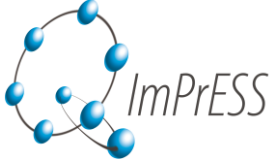
Should the system architect decide not to use model driven techniques for the implementation of the change scenario, he needs to make sure that the architecture represented by the source code actually matches the change scenario defined in the Service Architecture Model. This step can be omitted, if the Service Architecture Model is use to generate architectural code fragments.

3.7 Validating the Model by Measurements

To make sure the implemented model meets the requirements for which the change scenarios have been evaluated, it has to be validated by measurements. For example, it has to be tested if the Proxy component actually yields the performance that has been predicted.

3.8 Deploying the System

When the system architect has verified that the system meets the defined requirements, he can proceed by deploying the system. The used method for the deployment heavily depends on the domain for which the system is implemented. It may be as simple as manually copying

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

and executing the system on the target platform. In more complex scenarios, the deployment may be automated using deployment scripts using Make, Perl, Ant or other script based approaches, where the system architect simply executes the script for the system to be deployed.

4 Proof-for-concept of the method

A basic example system has already been used in WP2 and WP3 to get a common understanding on how the project partners expect the Q-ImPrESS method and its tools to interoperate. The same example is used again in this deliverable to illustrate the envisaged working method.

This basic example has been implemented and has been used as a proof-for-concept of the Q-ImPrESS method.

4.1 The basic use case

The developed case study illustrates a client-server architecture where a client communicates with a server offering a service to retrieve user data (e.g., login, password, etc.) stored in a database. Refer to Deliverable D2.1 for further details on the example.

4.1.1 Client-Server Example

The static structure of the example application is given in Figure 8.

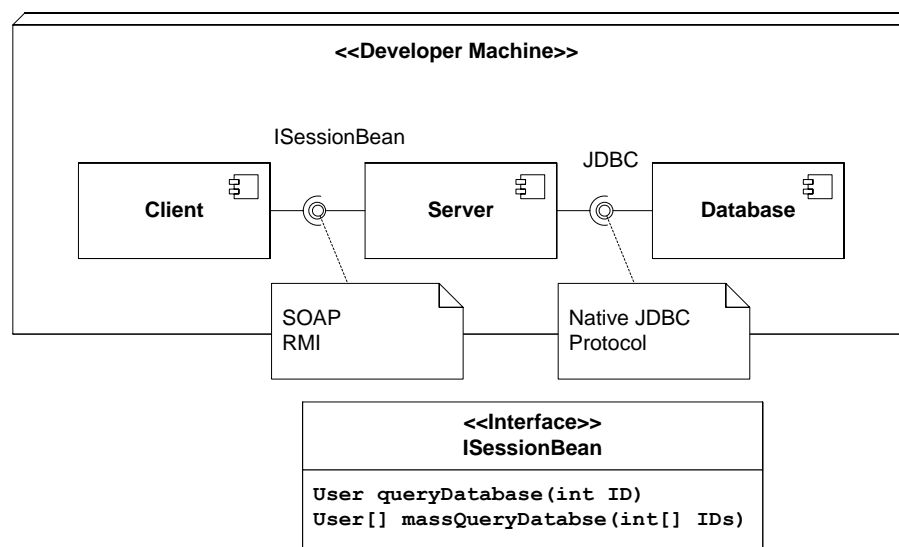


Figure 8 Basic Example Client-Server Architecture

The simple architecture consists of the following components:

1. Client – issues the requests for service
2. Server – provides the service by simple querying the database
3. Database – contains the data

Communication between components is annotated with possible communication style specification.

4.1.2 Client-Server Interaction

The client currently supports two types of interactions. A basic query (see Figure 9) where it simply sends a query to the server, and a mass query mode (see Figure 10) where it tries to retrieve a set of users from the database with a probabilistic number of users. In order to keep the example brief, please note, that Figure 9 and Figure 10 omit the quality annotations needed for the quality predictions (i.e., resource demands and failure rates).

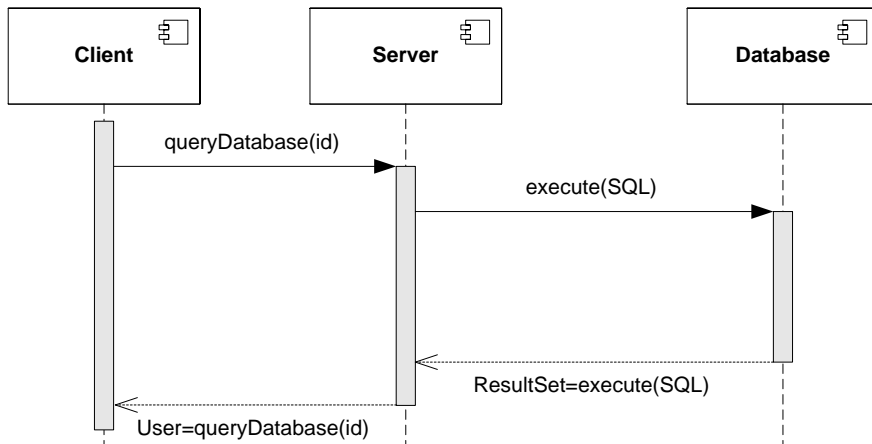


Figure 9 Sequence diagram basic query

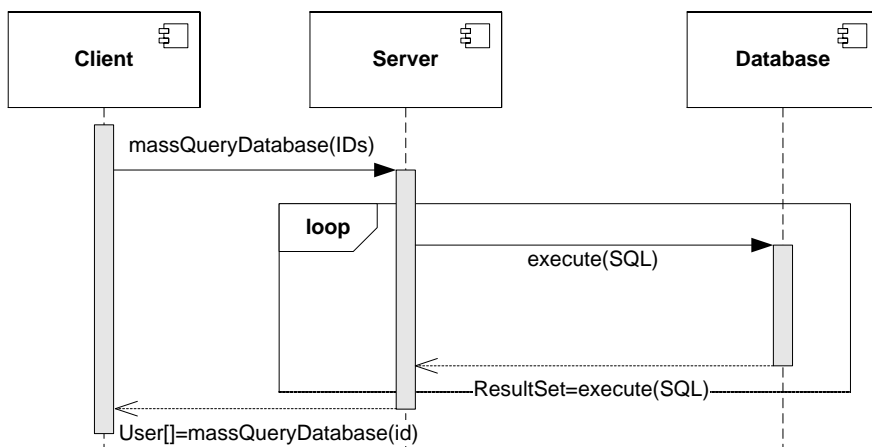


Figure 10 Sequence diagram mass query

If client server communication is done via SOAP the following picture describes the overall interaction:

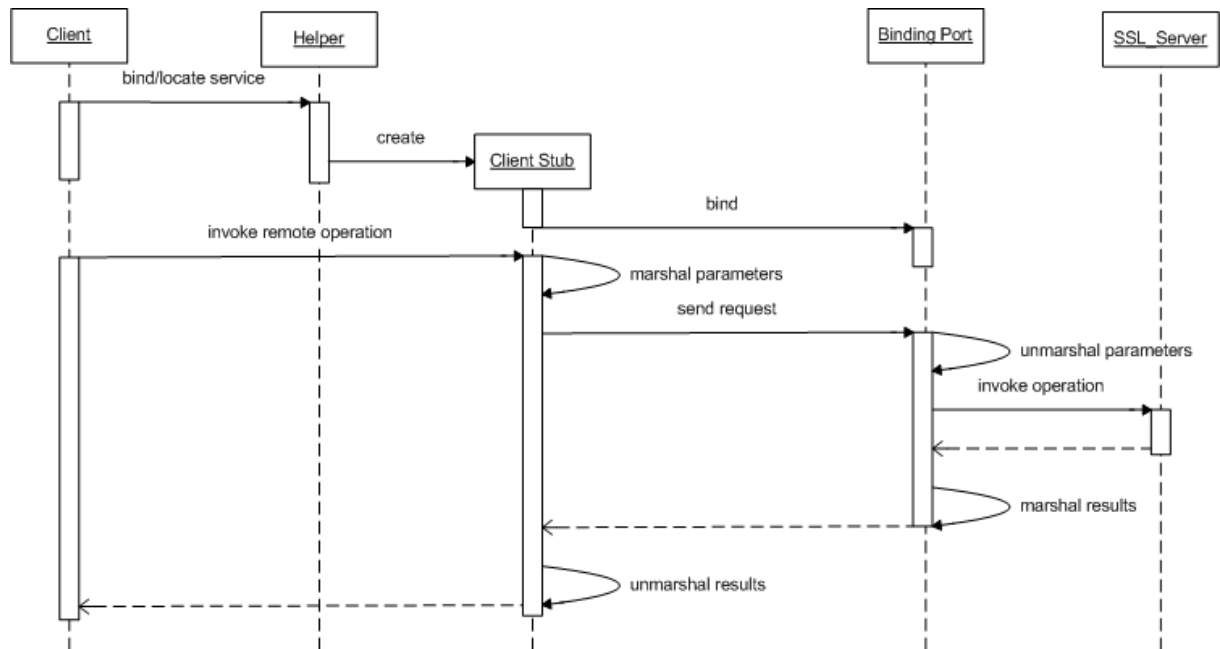


Figure 11 Sequence diagram Client-Server Interaction

Figure 11 outlines the sequence diagram describing the execution of both basic and mass query in case the SOAP protocol is used for server interaction; the RMI alternative could be represented in a similar way and is omitted

The following chapters describe in details the method applied on the basic example.

4.2 Process 3.1: Gather New Requirements for the test system

The new requirement is the following: given a defined system usage model, we want to enhance the system performance for the mass query call. System performance is measured as system response time; on the reference deployment environment and usage scenario the average response time for the mass query call is 10 ms. The requirement is to lower response time to an average of 8ms.

4.3 Process 3.2: Define implementation scenarios

Once the new requirement is formulated, the system architect (who knows the system very well) comes into play and proposes a couple of change scenarios:

- First assembly change scenario (component addition): the system architect, based on his experience in client/server architectures, suggests to introduce a Proxy component between the server and the database. The Proxy component allows more servers to connect to the same database, keeps track in a cache of each server request and database response, sends back to servers cached responses if still valid or send requests to database if already expired, handle response expiration. Even for such an easy test system it is not so trivial to predict if the introduction of such a component in the sequence really boosts system performance, which is the performance gain, and how much performance differs with respect to different caching options;

- Second change scenario (component update): the system architect, based on the performance results of the implemented system, realizes that most of the system latency is produced by the massQueryDatabase query, therefore suggests a new implementation of the method which avoids the looping of database queries and instead prepares a single SQL statement to be executed over the database (eventually changing the interaction method). The throughput and response time of this new implementation can be modelled as strictly adherent with the ones of the DBMS.

4.4 Process 3.3: Model Implementation scenarios

For the first change scenario, the software architect could model the static structure as follows:

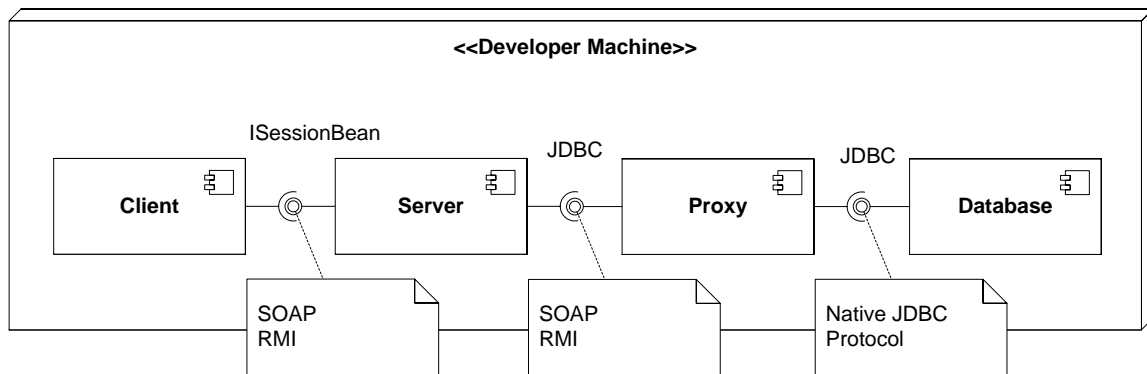


Figure 12 Model of the first change scenario

The connection between the Server and the Database component has been replaced by the intermediary Proxy component. The model of the Proxy component has to capture all relevant performance impacts that may arise when caching database requests or managing multiple server connections.

For the second change scenario, the Server component's behaviour model has to be changed to reflect that database queries do not occur in loops but in a single SQL statement. The following picture shows the sequence diagram of the new mass query implementation:

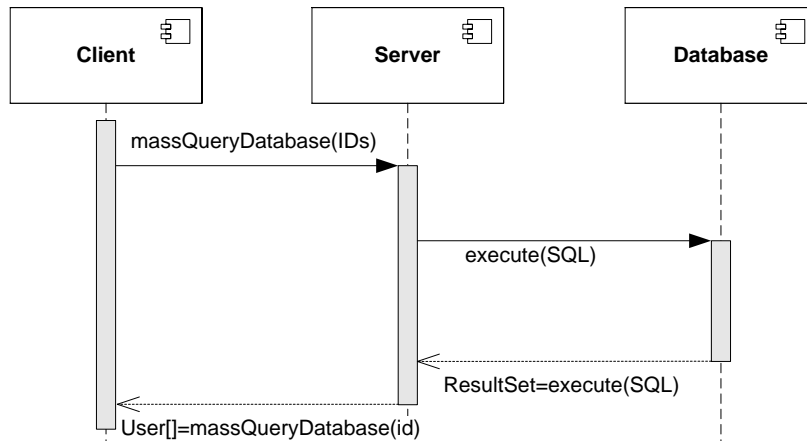


Figure 13 Sequence diagram of the second change scenario interaction

Based on the models for the change scenarios, the resulting system quality can be predicted.

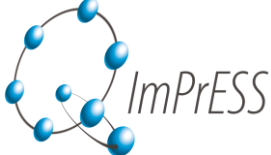
4.5 Process 3.4: Predict system quality

4.5.1 Predict system performance

Doing a performance prediction of the three design alternatives can result in three different tuples of figures, each tuple representing some characteristics of the design options. For example, consider a tuple (average response time, utilization). Then for the initial implementation, we get a prediction of average response time = 10ms and utilization = 70% for the defined usage profile, i.e., the tuple is **(10ms, 70%)**. For the scenario with the proxy, the response time is faster and the utilization lower, e.g., **(8ms, 55%)**. For the mass query, the response time is even higher, however, causing the same load **(6ms, 55%)**.

4.5.2 Predict system reliability

KLAPER tool allows evaluating the reliability of the whole system in the three considered design alternatives. The initial implementation, according to the defined usage profile and component nodes characteristics, provides an overall reliability equal to **0.9**. The second change scenario provides the best reliability value equal to **0.999**, which can be expected since the new design alternative reduces the execution time and hence the busy period of the DB component. The scenario with the proxy (first change scenario) provides an intermediate level of reliability equal to **0.95**, which is due by the fact that a further component is introduced and also the busy period of DB is reduced through caching. This is reasonable since, in general the higher is the number of tiers of a client-server application, the lower is the reliability. Indeed, assuming the same reliability values of components and failure independency, the reliability of the overall system is obtained as the product of the reliabilities in the execution path.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

4.5.3 Predict system maintenance

Applying Maintainability Prediction to this example means to determine cost effort estimations for implementation of the change scenarios. For each alternative the user has to describe the change steps (tasks) that lead from the current implementation to the implementation of the alternative. Therefore a workplan is determined which consists of a list of simple or hierarchical tasks. Each task represents some kind of change.

Maintainability Prediction of the first assembly change scenario:

Workplan “Introduce Proxy Component:

Task 1: create new component type “proxy”

Task 1.1: implement provided JDBC interface (from server to proxy)

Task 1.2: implement query delegation logic and required JDBC interface to DB

Task 1.3: implement cache

Task 1.3.1: caching logic,

Task 1.3.2: lookup logic,

Task 1.3.3: decision logic (simple string comparison)

Task 2: allocate “proxy” component on machine

Task 3: configure connection from server to proxy

Task 4: configure connection from proxy to database

The following time estimates are annotated to tasks.

Annotated Time estimates for workplan:

Task 1: 30 PersonHours

Task 2: 6 PersonHours

Task 3: 3 PersonHours

Task 4: 3 PersonHours

Aggregated Time Effort: **42** PersonHours.

Maintainability Prediction of alternative “massQueryDatabase improvement”:

For this alternative just the implementation of the massQueryDatabase() service function has to be changed.

Workplan “Update massQueryDatabase”:

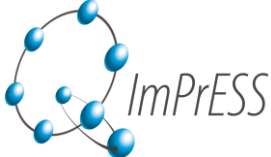
Task 1: Change implementation of massQueryDatabase()

Task 1.1: remove loop

Task 1.2: join data requests and build composite database query

Task 1.3: separate query results

Aggregated Time Effort: **6** PersonHours.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

Total results of maintainability prediction:

Proxy alternative costs 42 PersonHours, massQueryDatabase adaptation alternative costs 6 PersonHours. Hence last alternative is 7 times cheaper than the first one. Alternative 0 (Initial implementation) is already present (doesn't cost anything).

4.6 Process 3.5: Perform trade-off analysis

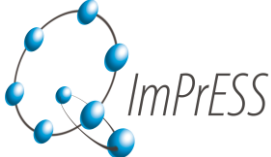
Let us assume that performance, reliability and maintainability are modelled by variables 'p', 'r' and 'm', respectively. Since in our example the aim of the analysis is to choose a design alternative that would improve the client-server system's performance, the designer can decide to weigh the performance higher than reliability and maintainability. Hence, suppose that the designer weighs the performance twice as much as reliability and maintainability: $w_p = 2$, $w_r = w_m = 1$. The weights can be normalized accordingly, such that they sum up to 1. In our example, the obvious best solution seems the reimplemented massQueryDataBase alternative, since it results in best performance, reliability and maintainability.

The trade-off analysis problem reduces to calculating the following sum: $cost_n = w_p * p_n + w_u * u_n + w_r * r_n + w_m * m_n$, n in $\{1,2\}$, for each design alternative and then pick the design alternative that has the least cost. Here, $w_u = 1$ encodes the utilization weight.

Instantiating the above formula with the actual values for the three QoS attributes, we get the following: for the proxy implementation scenario, $cost_{proxy} = 151201.516s$, and for the new mass query implementation, $cost_{massq} = 21601.561s$. Therefore, the new mass query implementation scenario is the 'cheapest' solution (measured in seconds), confirming the intuition.

4.7 Process 3.6: Implement change scenario

The results of the trade-off analysis are used to select the change scenario which should be implemented. The implementation of the Service Architecture Model can be done semi-automatically or manually. If it is done semi-automatically, code generation techniques have to be used that generate source code and configuration code based on the Service Architecture Model. Alternatively, the implementation of the components and the system provided in the Service Architecture Model can be done manually. For example, for every component of the Service Architecture Model, a corresponding Enterprise Java Bean (EJB) is being created which already contains some part, probably not complete, of the implementation logic of the component. As the models are still abstract the implementation cannot be fully generated. There may for example be placeholders to insert manually written code. Since the Service Architecture Model is more abstract than the final code, not all code can be generated. Therefore, manually written code has to be inserted into corresponding placeholders.

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

4.8 Process 3.7: Validate Model by Measurements

To make sure the implemented model meets the requirements for which the change scenarios have been evaluated, it has to be validated by measurements. For example, it has to be tested if the Proxy component actually yields the performance that has been predicted.

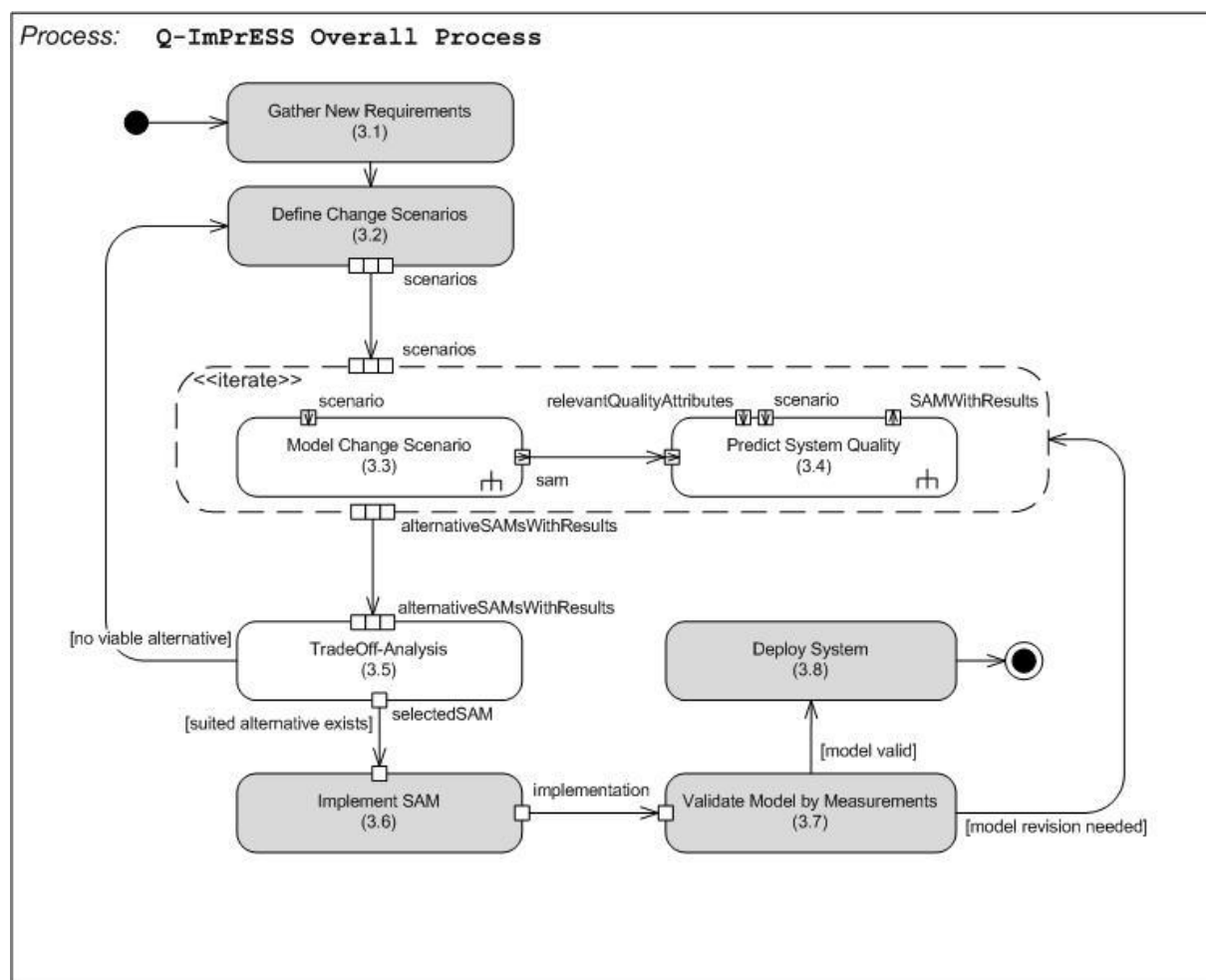
4.9 Process 3.8: Deploy updated test system

Finally, the test system has to be deployed. This involves allocating the components on the corresponding infrastructure and providing configuration data for the system. For example, EJB components have to be configured by means of deployment descriptors.

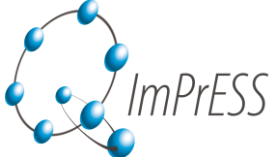
5 Workflow Process Reference table

This chapter contains a reference description of all the processes defined in the workflow.

5.1 Main workflow processes



3.1	
Process Name	Gather new requirements
Process input	None
Process output	A list of requirements.
Process description	The Product Manager collects new requirements for the system.
Expected duration	

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.2	
Process Name	Define Implementation Scenarios
Process input	Output of P1.1
Process output	A list of change scenarios to evaluate.
Process description	System architect, based on the list of requirements, defines a short list of implementation alternatives
Expected duration	

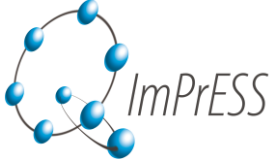
3.3	
Process Name	Model Implementation Scenario
Process input	A list of change scenarios to evaluate
Process output	The updated system SAM
Process description	The alternative change scenarios are described in the system SAM
Expected duration	

3.4	
Process Name	Predict System Quality
Process input	The updated system SAM with alternative scenario described
Process output	Prediction analysis results saved in SAM for each change scenario
Process description	Prediction analysis is performed for each change scenario and results are saved in SAM.
Expected duration	

3.5	
Process Name	Trade-Off Analysis
Process input	Prediction analysis results saved in SAM for each change scenario
Process output	The selected change scenario
Process description	The system architecture evaluates the different alternatives and selects the best one
Expected duration	

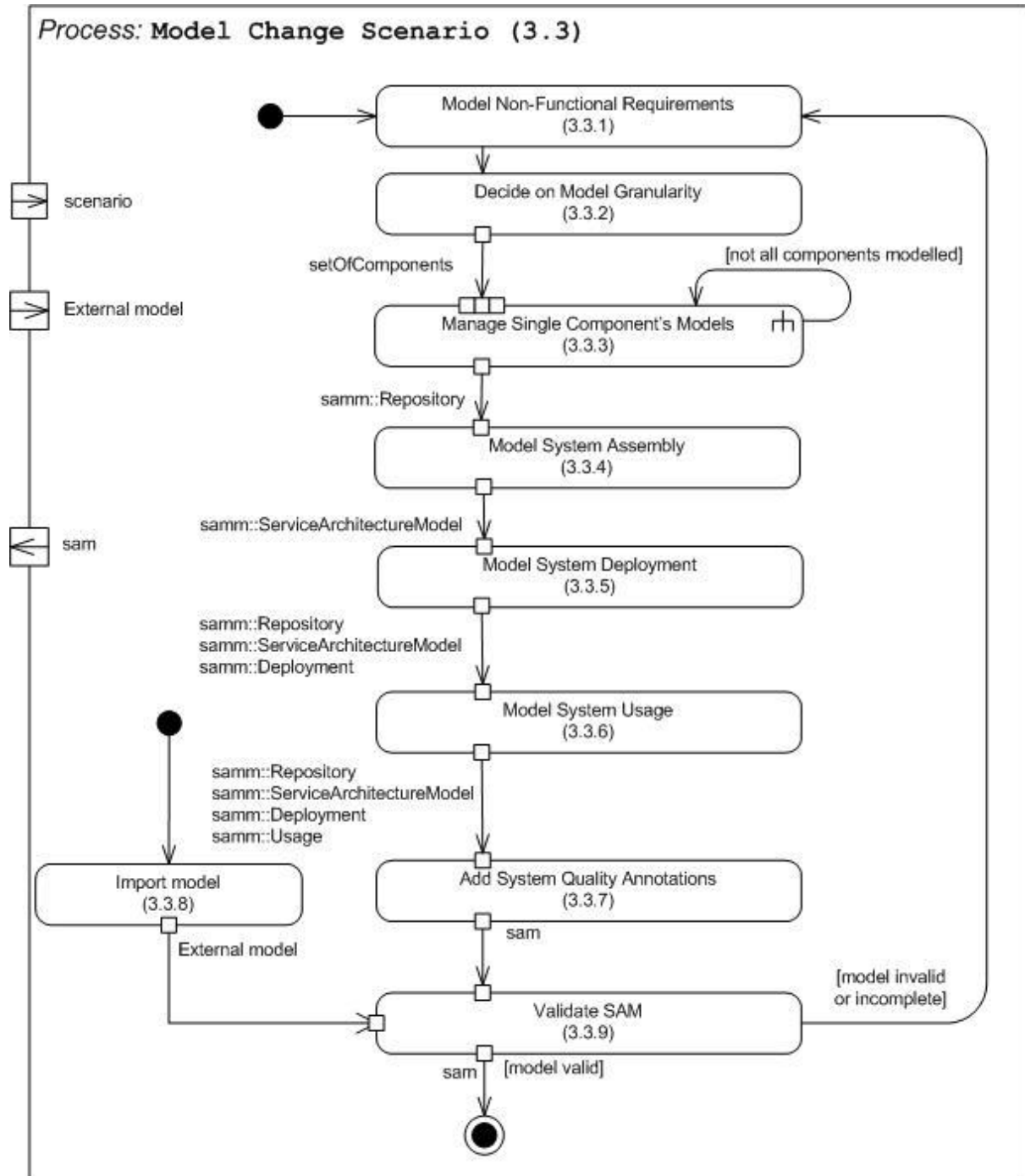
3.6	
Process Name	Implement SAM
Process input	The SAM of the selected change scenario
Process output	System updates implemented
Process description	The software engineer implements changes
Expected duration	

3.7	
Process Name	Validate Model By Measurements
Process input	Change scenario implemented
Process output	SAM validated
Process description	The software engineer analyses the implemented change and validate the model
Expected duration	

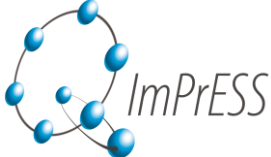
	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.8	
Process Name	Deploy System
Process input	Change scenario implemented
Process output	New system deployed
Process description	Software engineer deploys the new system.
Expected duration	

5.2 Process 3.3: Model Implementation Scenario processes



3.3.1	
Process Name	Model Non-Functional Requirements
Process input	Functional specification of the target application
Process output	Set of quality attribute and constraints for the target implementation
Process description	The system architect, according to the end-users requirements, define

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

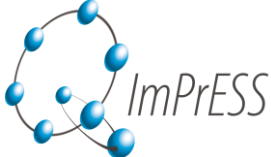
	the reliability, performance and maintainability constraints for the specific service components or for the whole architecture
Expected duration	Depending on the complexity of the system and the coordination effort needed it can take from hours to days

3.3.2	
Process Name	Decide on Model Granularity
Process input	Set of quality attribute and constraints for the target implementation
Process output	Decision for each component on which level, i.e., white box, grey box, or black box, it should be modelled
Process description	The system architect can hierarchically model the system and decide on the components granularity depending on the required accuracy of the output and on the impact of the specific component on the whole quality of the system
Expected duration	A couple of hours

3.3.3	
Process Name	Manage single component's models
Process input	Component characterized with the quality attributes of interest, Components' interfaces, human insight into application
Process output	SAM of a component
Process description	The system architect specifies the SAM of individual components, including the behaviour specification of the component and/or the component's services. See details of the process given starting from Table 3.3.3.1 up to Table 3.3.3.5.
Expected duration	Depends on the complexity of the component, but should be within hours. Depending on the number of components and measurements needed. See Process 3.3.3 tables for more details.

3.3.4	
Process Name	Model System Assembly
Process input	SAM of a target system components
Process output	SAM assembly model
Process description	The system architect details the service components and their connectors
Expected duration	Depends on the size of the target execution environment, but should be doable from hours to a few days

3.3.5	
Process Name	Model System Deployment
Process input	SAM of a target system components
Process output	SAM deployment model
Process description	The system architect defines the deployment of the service components to a candidate physical infrastructures
Expected duration	Depends on the size of the target execution environment, but should be

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

	doable from hours to a few days
--	---------------------------------

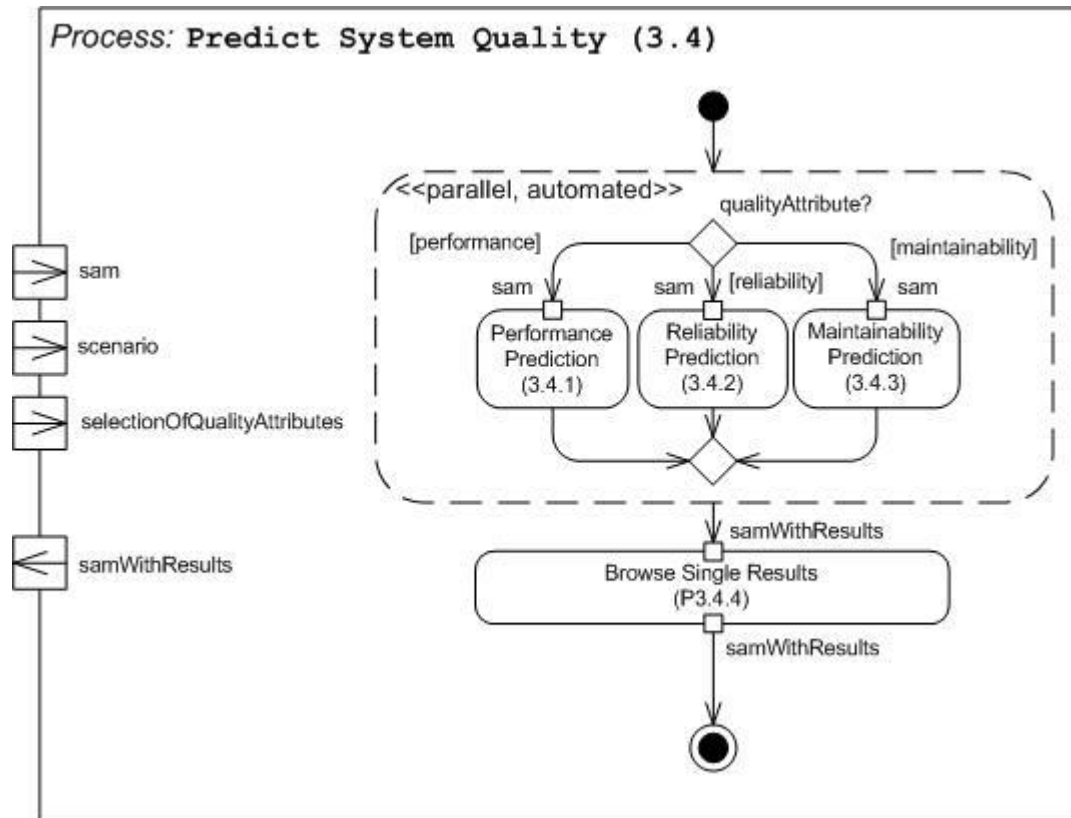
3.3.6	
Process Name	Model System Usage
Process input	Behavioural and deployment SAM models
Process output	SAM usage model
Process description	The system architect specifies the usage profile of the system according to monitoring data or to a prediction of the usage pattern of the system by end-users
Expected duration	Depends on the complexity of the usage but should be doable in hours

3.3.7	
Process Name	Add System Quality Annotations
Process input	Static, Behavioural, deployment, and usage SAM models
Process output	SAM with annotated quality attribute and constraints for the target implementation
Process description	The system architect, using the SAMM quality annotation package, defines the components quality annotations and introduces the constraints for the specific service components or for the whole architecture
Expected duration	Hours to a few days depending on the amount of missing annotations not yet modelled for components

3.3.8	
Process Name	Import Model
Process input	External model in a valid format
Process output	SAM with annotated quality attribute and constraints for the target implementation
Process description	The system architect, loads an external model.
Expected duration	Should take some minutes, depending on the model size.

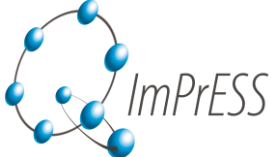
3.3.9	
Process Name	Validate SAM
Process input	SAM models for the whole architecture
Process output	Validated SAM models or set of detected inconsistencies
Process description	The Q-ImPRESS platform runs a consistency check in order to produce validated models which will be further analyzed by the Q-ImPRESS toolkit or to notify the system architect with a set of inconsistencies in the specifications to be fixed
Expected duration	Maximum a couple of seconds

5.3 Process 3.4: Predict SAM quality processes



3.4.1	
Process Name	Performance Prediction
Process input	SAM + Quality Attributes
Process output	
Process description	
Expected duration	

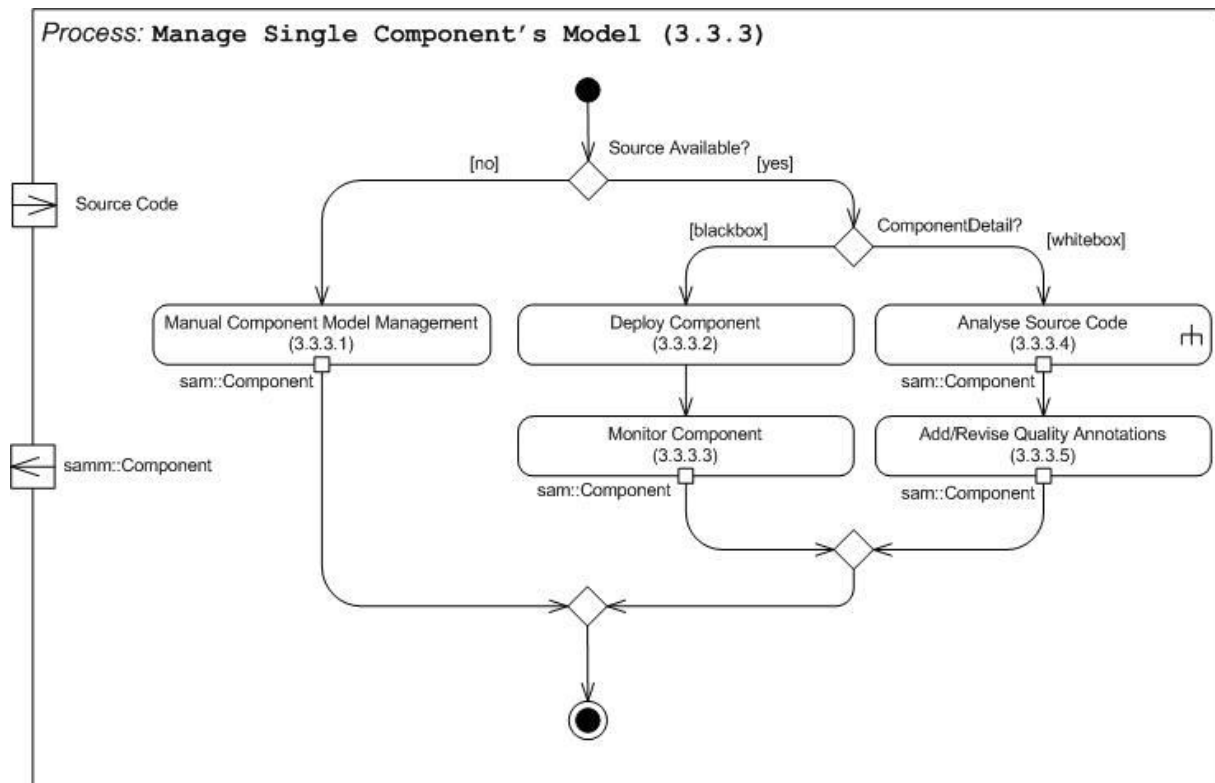
3.4.2	
Process Name	Reliability Prediction
Process input	SAM + Quality Attributes
Process output	Reliability of services and infrastructural components
Process description	The System architect can trigger what-if or trade-off analyses which include reliability quality metrics.
Expected duration	

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

3.4.3	
Process Name	Maintainability Prediction
Process input	SAM + Scenario (=change request)
Process output	Workplan model with annotated effort estimates and aggregated total effort.
Process description	<p>The maintainability prediction has the goal to guide a software architect to estimate the effort which is necessary to implement a given change request in the architecture described by an architecture model.</p> <p>Therefore the maintainability prediction process takes as inputs the architecture model, i. e. an instance of the SAMM and the description of a change request. The description of change request is represented as scenario in the overall workflow.</p> <p>The process of predicting maintainability can be divided in the following processes:</p> <ol style="list-style-type: none"> 1) Identify parts of the architecture model which are affected by the change, then identifying the basic 2) Identify basic steps to cope with the change scenario (workplan) 3) Estimate and annotate time effort for each step, in this step the tool will guide the system architect with some investigation supports 4) Compute aggregate change effort
Expected duration	Highly dependent on complexity of change request, architecture model, knowledge of architects. Approximately ranging from few hours to few days to few days.


3.4.4	
Process Name	Browse Single Results
Process input	Prediction Analysis results
Process output	
Process description	The Q-ImPrESS tool displays prediction analysis results, using text and charts, to the system architect
Expected duration	

5.4 Process 3.3.3: Manage Single Component's Model



3.3.3.1	
Process Name	Manual Component Model Management
Process input	Components' interfaces, human insight into application
Process output	SAMM::component description
Process description	This process step's execution leads to a manual created model of a single component (SAMM::component). The component developer has to model the component by using the provided concrete syntax of the SAMM. He is responsible for maintaining the model himself. Also he has to make sure the model complies with code which might be provided later.
Expected duration	Depends on the complexity of the component, but should be within hours

3.3.3.2	
Process Name	Deploy Component
Process input	Component implementation in form of source code, valid environment
Process output	A running system or testbed containing instrumented component
Process description	In this process step the system deployer takes the implementation of a black box component and installs it in an appropriate environment which resembles the final execution environment from a functional and non-functional viewpoint. Additionally the component has to be instrumented by the means of its environment to allow monitoring of the component. The definition of the monitoring places might need some preparation to collect the right data needed.

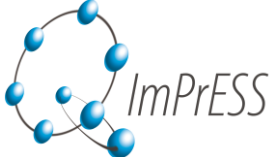
	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

Expected duration	Minutes to Hours, depending on the monitoring setup complexity
--------------------------	--

3.3.3.3	
Process Name	Monitor Component
Process input	A running system employing instrumented component
Process output	Abstract SAMM::component instance including performance and behaviour descriptions gathered by monitoring
Process description	This process includes executing the system and monitoring the installed components. Monitoring data is obtained automatically while the system is running
Expected duration	Minutes to days depending on the system and the desired accuracy of the gathered monitoring data

3.3.3.4	
Process Name	Analyse Source Code – static part
Process input	Implementation of the application in form of source code
Process output	Static structure of the application (identification of individual components and connectors among them) as SAMM::ServiceModel instance. The static structure extraction is a semi-automatic process which needs sometimes hints provided by the user.
Process description	The source code is transformed by tools into a language-independent representation (G-AST) which is in turn analyzed to identify components.
Expected duration	Tens of minutes to hours depending on the size, language and complexity of the application

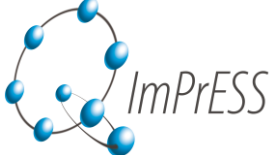
3.3.3.5	
Process Name	Add/Revise Quality Annotations
Process input	Output of P4.4. – automatically obtained static architecture and behaviour description of individual components
Process output	More precise application structure and behaviour and quality description
Process description	Analysis tools are limited and the implementation does not contain all design-level and usage information, so human insight is employed to improve the model. Especially, statically the resource demand of the implementation cannot be recovered by static analysis tools. Here human input or further monitoring is required.
Expected duration	Minutes to hours depending on the size of the system

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

6 Glossary

Term used within the Q-ImPrESS project, sorted alphabetically

Black-box component	A component having only interface information. Black-box components cannot be analysed using tools of static analysis, because they have no information about their internals, however, black-box components can be monitored. In order to apply static analysis, a black-box component needs to be turned into a grey-box component
Change scenario	A potential change in the system (assembly change scenario), the environment (allocation change scenario) or the usage (usage change scenario) to cope with a new requirement.
Component	A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition. (Szyperski)
Connector	Connectors are architectural building blocks used to model interactions among components and rules that govern those interactions. (Medvidovic)
G-AST	Generic Abstract Syntax Tree
Grey-box component	A component having interface information and additional model information (QoS annotations + behavioural aspects in case of composite components) needed for analysis
Non-functional requirement	A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours.
PCM	Palladio Component Model
SAM	Service Architecture Model, an instance of SAMM
SAMM	Common meta-model which contains everything to describe the information needed for quality prediction analysis. Serves as shared data-repository for all quality analysis methods.
Service	A deployed component
Service architecture	A set of services connected to each other via connectors. A subject to analysis.
SISSy	Structural Investigation of Software Systems. SISSy is a platform for problem pattern identification in OO source code written in Java, C++ or Delphi. SISSy produces a G-AST.
SOA	Service Oriented Architecture

	D6.1: Method and Abstract Workflow	
	Version: 1.0	Last change: 14/05/2009

Static Structure	Description of component and connector composition that captures the static architecture of a service at the time of deployment
White-box component	A component with available source-code

7 Conclusions

A generic method to follow in using the Q-ImPRESS toolkit has been proposed and explained. This method has been tested with a basic example. This method will guide the implementation of the demonstrators and will be used as a reference for the guidelines and cookbooks for the three specific domains tackled by the project.