D4.6: Use Case Specific DP & Holistic Escrow

W4 – Processes and Methods for Digitally Preserving Business Processes

Delivery Date: 31/03/2013

Dissemination Level: Public

TIMBUS is supported by the European Union
under the 7th Framework Programme
for research and technological development and demonstration activities (FP7/2007-2013)
under grant agreement no. 269940

Deliverable Lead		
Name	**Organisation**	**e-mail**
Stephan Strodl	SBA	sstrodl@sba-research.org

Contributors		
Name	**Organisation**	**e-mail**
Stephan Strodl	SBA	sstrodl@sba-research.org
Konstantin Hobel	SBA	khobel@sba-research.org
Elisabeth Weigl	SBA	eweigl@sba-research.org
Tomasz Miksa	SBA	tmiksa@sba-research.org
Stefan Pröll	SBA	sproell@sba-research.org
Rudolf Mayer	SBA	rmayer@sba-research.org
Severin Winkler	SBA	swinkler@sba-research.org
Marco Unterberger	SBA	munterberger@sba-research.org
Johannes Binder	SBA	jbinder@sba-research.org
Barbara Kolany	ITM	barbara.kolany@uni-muenster.de
Hecheltjen Martin	ITM	martin.hecheltjen@uni-muenster.de
Yankova Silviya	ITM	silviya.yankova@uni-muenster.de
Gonçalo Antunes	INESC-ID	goncalo.antunes@ist.utl.pt
Ricardo Vieira	INESC-ID	rjcv@ist.utl.pt
Daniel Draws	SQS	Daniel.Draws@sqs.com
Daniel Simon	SQS	Daniel.Simon@sqs.com
Sven Euteneuer	SQS	Sven.Euteneuer@sqs.de

Internal Reviewer		
Name	**Organisation**	**e-mail**
Thomas Molka	SAP	thomas.molka@sap.com
Carlos Coutinho	CMS	carlos.coutinho@caixamagica.pt

Disclaimer

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADE	Adverse drug events
BP	Business process
CI	Continuous Integration
ADL	Architecture Description Language
LOC	Lines of Code
NCSS	Non-commenting source statements
PMD	PMD Java Source Code Scanner (http://pmd.sourceforge.net/pmd-5.0.2/meaning.html)
DP	Digital Preservation
DSO	Domain-specific ontologies
IERM	the Intelligent Enterprise Risk Management Module
OAIS	Open Archival Information System
QA	Quality Assurance
QC	Qualitas Corpus[1]
UrhG	*German:* Urheberrechtsgesetz; *translation:* copyright law
Rz	*German:* Randzahl; *translation:* margin note
ABGB	*German:* Allgemeines bürgerliches Gesetzbuch; *translation:* Civil Code of Austria
ASP	Application Service Provider
IP	Intellectual Property
IPR	Intellectual Property Right
BGBI	*German:* Bundesgesetzblatt; *translation:* Federal Law Gazette
BGH	*German:* Bundesgerichtshof; *translation:* Federal Court of Justice of Germany
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SW	Software
VM	Virtual machine
atIO	*German:* österreichische Insolvenzordnung; *translation:* Austrian Insolvency Act
gInsO	*German:* deutsche Insolvenzordnung; *translation:* German Insolvency Act
WP	Work Package

[1] http://qualitascorpus.com

1 Executive Summary

The focus in the area of Digital Preservation (DP) so far has been predominantly on static objects, such as data sets, documents and media. However, there is an increasing demand on preserving whole processes, e.g. scientific workflows and critical business processes for the long term. The TIMBUS research project aims at providing a methodology, guidelines and tools to preserve processes. The first part of this report presents the TIMBUS process framework that is used to digitally preserve a business process. The framework defines the required processes and actions needed for planning, for executing the preservation, and for redeploying a process in a new environment at some point in the future. The TIMBUS process framework is domain independent and can be adjusted and applied to different scenarios. The framework describes the information flow between the steps of the TIMBUS preservation process, provides recommendation for potential methods and tools needed for performing the preservation, and defines responsibilities of the involved stakeholders. The framework guides the implementation of a preservation solution for a business process. The TIMBUS process consists of three phases: plan, preserve and redeploy. TIMBUS adopts a risk management approach that considers Digital Preservation as a risk mitigation strategy on an enterprise wide perspective for business processes. In TIMBUS, Digital Preservation is used to make a business process, including data, procedures, used services, infrastructure, legal and organisational aspects available and useable in the long run. This deliverable specifies the supporting services for the processes in accordance with the TIMBUS architecture. Examples of practical implementation and adoption are presented by using the TIMBUS use cases.

Business processes are an orchestration of activities to achieve a specific goal. The activities are supported by various services that are provided by different systems including customised software or external services such as web services. In order to preserve the functionality of the business processes, the problem of how to make external services available in the future needs to be solved. A similar problem is faced in the case of commercial or customised software that is available at the customer, but only in the shape of object code. In case the developer is not available in the future (e.g. due to bankruptcy), the object code remains with the user but he has no handle on demanding the source code for further adjustments or new requirements. The second part of this deliverable thus describes Holistic Software Escrow methods, which ensure the access to deposited material for example source code under specific circumstances. It provides a potential solution to maintain external dependencies for long term preservation while providing additional protection of investments. We present a process to plan and execute an escrow agreement addressing technical as well as legal aspects of the procedure. From the legal point of view, the escrow contract specifies the obligations of the involved parties including notification, information, release and deposit conditions. From the technical perspective, the selection of the material to be deposited and the quality evaluation of the material are critical parts of the process to ensure usability in the future. The TIMBUS Technical Framework for Software Escrow was developed to support the verification process of Software Escrow. It implements measurements facilitating the assessment of the quality of software artefacts with focus on Software Escrow specific concerns. The software prototype is implemented as an extensible framework based on a plug-in

infrastructure. It that can be extended and adjusted for the specific requirements defined in the escrow agreement. The feasibility of the different plug-ins has been tested with experiments using different software projects. Existing software quality metrics as well as newly adopted measurements were used for the experiments. The results show that the Software Escrow verification process can be supported by providing software quality measurements and highlighting software artefacts or sections of potential interest for further manual reviews.

2 Introduction

Digital Preservation is the management of digitally information over the time. While the research on Digital Preservation traditionally followed a data centric view of information, the long term preservation of complex systems and processes received less attention. Currently there is an increasing interest in the preservation of complete processes over time, going beyond the preservation of data. The TIMBUS project aims at providing methodology, guidelines and tools to capture, document, and preserve processes for the long term. This deliverable presents the TIMBUS process framework that describes the process steps of the three phases TIMBUS approach: plan, preserve, and redeploy. This document further describes Software Escrow as a potential preservation solution for dependencies to external business partners.

Business processes are an orchestration of activities to achieve a specific goal. The preservation of such as process requires sufficient details of the process and the context it is embedded in. This context is needed in order to redeploy the process with its original behaviour in the future. Moreover, detailed planning and testing of potential preservation strategies and controlled execution of the required preservation actions is needed for a correct redeployment of the process in the future. Process preservation needs to address the different layers of a process, including business, application, and technology layer. The relevant context of the process has to be captured including technologies both on hardware and software levels, dependencies between these components, the use of services operated by external providers, data, and the high-level concepts of the business layer. The business layers address organisational aspects of the process including its business logic, the involved stakeholders and legal obligations. Business processes are supported or implemented by various services that may be provided by different systems. The services can be provided by external service operators. As these services are beyond direct access, it is not easy to capture and preserve them. In the long run, the availability of current technology and services cannot be guaranteed, especially if they are provided by third parties. The functionality of business processes can be threatened by missing software services or obsolete technology. Legal concerns constitute further threats such as non-compliance with contracts, licences, or SLAs (Service Level Agreements). Moreover, domain specific laws and regulations such as data protection set requirements and obligation for a preservation solution.

The presented TIMBUS framework guides organisations through the three phases of the TIMBUS preservation approach. The approach is driven from an enterprise risk management perspective. Digital Preservation is a potential risk mitigation strategy derived from the threat of potential loss of availability of information over time. Risk management is used to identify and evaluate different risks associated with business processes in a structured and well defined manner. Different preservation alternatives to preserve an adequate set of information have to be identified and evaluated. Within the planning phase, the context of the process is captured, including all relevant aspects that are needed for preservation and redeployment. A preservation plan is created defining relevant actions that need to be executed for extracting the relevant process data from the productive environment, and subsequently preserving and archiving the process for the long term. The relevant components implementing and documenting the significant properties of the process need to be identified and maintained over time. Processes may not be limited to a single system but often make use of distributed services across platforms that need to be identified and captured for preservation. The used components need to be described and preserved for future use. Based on cost-risk considerations, suitable preservation strategies are selected. The preservation phase executes the preservation strategies and prepares the process for long term archiving. The redeployment of the business process defines the re-execution of the preserved process in a new environment at some point in the future. This redeployed process needs to be verified for its authenticity and correctness. An evaluation is required to compare the significant properties of the redeployed process with reference values of the original process. The TIMBUS framework provides guidance through the process with references to implementations and related work. The TIMBUS process describes the required actions that need to be performed and the expected outcome. Responsibilities on the process level are defined and involved stakeholders are specified. The framework we developed is not limited to a single use-case but provides the flexibility to be applied in various settings considering the specific requirements and implementation. The TIMBUS framework needs to be instantiated and adapted with respect to specific requirements of the business process to be preserved. Examples for implementations are shown in this deliverable by using the TIMBUS use cases. The presented framework describes the theoretic principles for the preservation process. The architectural support for the processes by the TIMBUS architecture is identified and presented in this deliverable. Tool support for the architecture is developed as part of WP6 within the TIMBUS project providing implementations for the process steps.

The use of external services within a business process presents a significant challenge for the long term preservation. Limited or no access to the implementation leaves the services vulnerable to obsolescence and in worst case disappearance over time, which promotes dependencies on external vendors. Escrow provides a business model to ensure access to a third party implementation under specific circumstances. This deliverable thus introduces Holistic Software Escrow, a concept for setting up escrow agreements for third party software with a special focus on technical and legal aspects. Software can be essential for business processes in companies providing critical services to deliver products and services. In many cases the software solutions are custom made and delivered by external vendors. It is a common practice in software vendors to deliver only the binary code of the software to the customer, as the maintenance and adaption of the software is also provided by the vendor as part of his business model. This dependency becomes critical for the customer in particular when the vendor stops the maintenance of the software system or goes out of business. Without access to the implementation of the software, further development or adoption of the software for new requirements is hardly possible. In order to prevent such situations a Software Escrow agreement can be established. Within such an agreement, the implementation of the software project is deposited with an independent software escrow agent. When predefined events occur, such as bankruptcy of the vendor, the deposited material is released to the customer. This deliverable guides through the process of Software Escrow, examining technical and legal aspects of the agreement. The presented Holistic Software Escrow extends the scope of existing approaches. So far only the source code was deposited including a wide range of relevant software project artefacts e.g. including documentation and design documents. A key challenge of the procedure is the verification of the deposited material to check whether it can actually be used in the future. A technical framework is introduced in this deliverable to support the semi-automated evaluation of deposited software projects regarding completeness and sufficient quality of the artefacts. The aim of the framework is to decrease the time and cost of the evaluation of deposited software artefacts by decreasing the need for manual tasks to be carried out by experts. The framework combines a number of existing and adopted quality metrics to identify elements that need further manual review. A set of software projects has been used to test the feasibility of the framework, and the significance of different quality metrics. The principles of Software Escrow can be applied to any third party software service, e.g. external web services. It provides a practical solution to overcome external dependencies for the long term preservation.

The remainder of this document is structured as follows. Section 3 presents the TIMBUS process framework. In Section 3.3 the fundamental concepts of the framework are specified such as stakeholders, artefacts and environments. The processes to digitally preserve a business process are described in Section 3.4 describing the process steps of the three phases. The architectural view of the process framework is shown in Section 3.5 where the support of the TIMBUS architecture is described. The application of the framework on the TIMBUS use cases is presented in Section 3.6 for the Civil Engineering and the e-Health use case. The process description ends with a summary in Section 3.7. Holistic Software Escrow is discussed in Section 4. The escrow procedure is grouped into three phases, each section discussing the technical and legal, respectively contractual aspects of each phase. The Technical Framework presented in Section 4.3.3 discusses the components needed for a holistic evaluation with focus on maintainability. An introduction of the prototype implementation is given in Section 4.6, followed by the experiments executed by using different software projects. In Section 4.7 we provide conclusions on the proposed the Holistic Software Escrow. Section 5 provides a summary of this deliverable and provides an outlook of future work within the TIMBUS project.

3 TIMBUS Digital Preservation Process Framework

3.1 Introduction

The TIMBUS Digital Preservation Framework defines the process to digital preserve a business process (BP). It describes the three phases of the preservation: planning, preserve and redeploy. The framework describes the process steps of each phase specifying input, output, methods and responsibilities. It should guide organisations through the process of preserving business processes. The framework is domain independent and can be applied to different settings. It can be adjusted for the specific needs and requirements of a setting. For the process steps of the framework, methods and specific responsibilities of involved stakeholders are suggested. The instantiation of the framework needs to be done for specific settings depending on the implementation of the process, requirements, obligations and involved actors. The wide range of processes that are supported by the framework allow for a flexible implementation of the process steps. It provides a clear specification of required actions and information for preservation and redeployment of a business process, but is flexible enough to be adjusted according to be requirements of specific settings.

The framework was designed by taking into account the use case scenarios such as the civil engineering use case LNEC [1] and the former use case of scientific experiments [2] as a reference points. Moreover the TIMBUS framework was influenced by a number of other models, procedures and frameworks in the area of the Digital Preservation and other related disciplines. A variety of related work has been considered for the TIMBUS DP Process Framework, starting from the fundamental concepts of OAIS [3] and risk management [4] to the work of former DP research projects such as CASPAR[2], SHARMAN[3] and PLANETS[4] ranging to work of information security, verification and authentication [5]. Due to the wide range of different business processes and implementation that are supported by the framework, it defines the basic concepts and methods for the three phases of the TIMBUS approach.

The remainder of this section is structured as follows. Chapter 3.2 introduces related work of the process framework for Digital Preservation of business processes including process modelling techniques and related work in Digital Preservation and other disciplines. Section 3.3 describes the preservation process, starting with the definition of the fundamental concepts such as stakeholders, artifices and environments. The involved processes of three phases of the TIMBUS framework are shown in Section 3.4.2 for planning, Section 3.4.3 for preservation and Section 3.4.4 for redeployment. The monitoring and evaluation process for the business process is described in Section 3.4.5. Section 3.5 provides an architectural view of the process that identifies the support by the TIMBUS architecture [6]. Section 3.6 completes the process description with use case specific examples of the TIMBUS framework.

[2] http://www.casparpreserves.eu/

[3] http://shaman-ip.eu/

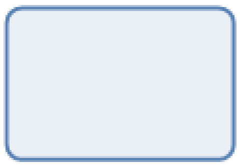
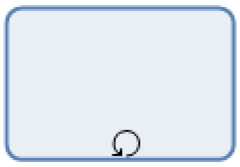
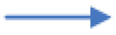
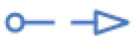
[4] http://www.planets-project.eu/

3.2 Related work

This section points to related work of the TIMBUS process framework in the area of Digital Preservation and related disciplines. The methods used for process and architecture modelling (BPMN and Archimate) in this deliverable are introduced in this section.

3.2.1 BPMN 2.0

The Business Process Model and Notation (BPMN) specification was created by the Business Process Modelling Initiative and published in 2004. The model was then adopted by the Object Management Group, with version 2.0 being the latest update [7]. BPMN provides for all stakeholders an intuitive notation while being able to represent complex processes. Thus it builds a bridge across the gap between different business stakeholders, from business analysts to developers and managers. Business process operations are visualized by Business Process Diagrams (BPD). A list of frequently used symbols can be found in the Figure 1 below. A complete overview of BPMN symbols and diagrams is available online[5]. BPMN is used for the graphical representation of the TIMBUS processes in this document.

Symbol	Description
	**Task / Activity**
	**Loop Task** The task is repeated.
	**Sequence Flow** shows the order in which activities in a process will be executed
	**Message flow** Sending and receiving a message between flow participants

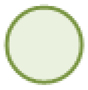
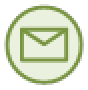
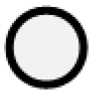
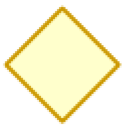
Symbol	Description
	Start Event
	Start Message Event Receiving messages and triggering the Event
	End Event
	Gateway (decision)

Figure 1: BPMN 2.0 symbols

[5] E.g. http://www.bpmb.de/images/BPMN2_0_Poster_EN.pdf

3.2.2 ArchiMate

ArchiMate is a domain-independent modelling language for enterprise architecture and provides tools to describe, analyse, and visualize relationships within and across business domains. As a The Open Group[6] standard, ArchiMate has been widely adopted for real-world use in the commercial and educational sectors, based on practical experience of many years. It fully aligns with the TOGAF standard [8]. Thus, results from The Open Group forums and work groups in this domain are directly incorporated. Therefore ArchiMate follows the terms provided by the TOGAF standard instead of defining its own terms. TOGAF is a framework that enables developing enterprise architectures. More information on ArchiMate can be found in Deliverable 4.9 [9]. ArchiMate is used to show the architectural view of the TIMBUS processes with the support of the TIMBUS architecture in Section 3.5.

3.2.3 TIMBUS methods, tools and architecture

3.2.3.1 Context model

The TIMBUS context model [10] [11] provides a framework to systematically model relevant context parameters and dependencies. It is sufficiently comprehensive in order to address different Digital Preservation relevant parameters of the business process context. The goal of the context model is to capture all aspects of a business process that are relevant for risk management, preservation planning, preservation, archival storage and redeployment. It is implemented as ontology. The model consists of a domain-independent structure that represents the core concepts, which are aligned with the ArchiMate concepts. The context model can be extended by domain specific ontologies (DSO) in order to capture concerns that are of particular relevance to a specific domain. The TIMBUS context model already defines a set of such domain specific ontologies that serve as a reference implementation.

3.2.3.2 TIMBUS architecture

The TIMBUS architecture represents the complete solution for performing preservation of modern business processes. An overview about the main components is show in Figure 2. Its goal is to provide a comprehensive architecture to cover the various aspects and requirements for the long-term preservation of complex digital objects such as different data types, entire business processes and the execution environment they run in. A detailed description on the TIMBUS architecture can be found in [6]. The support of the TIMBUS processes designed in this deliverable and the TIMBUS architecture are presented in Section 3.5.

[6] http://www.opengroup.org/

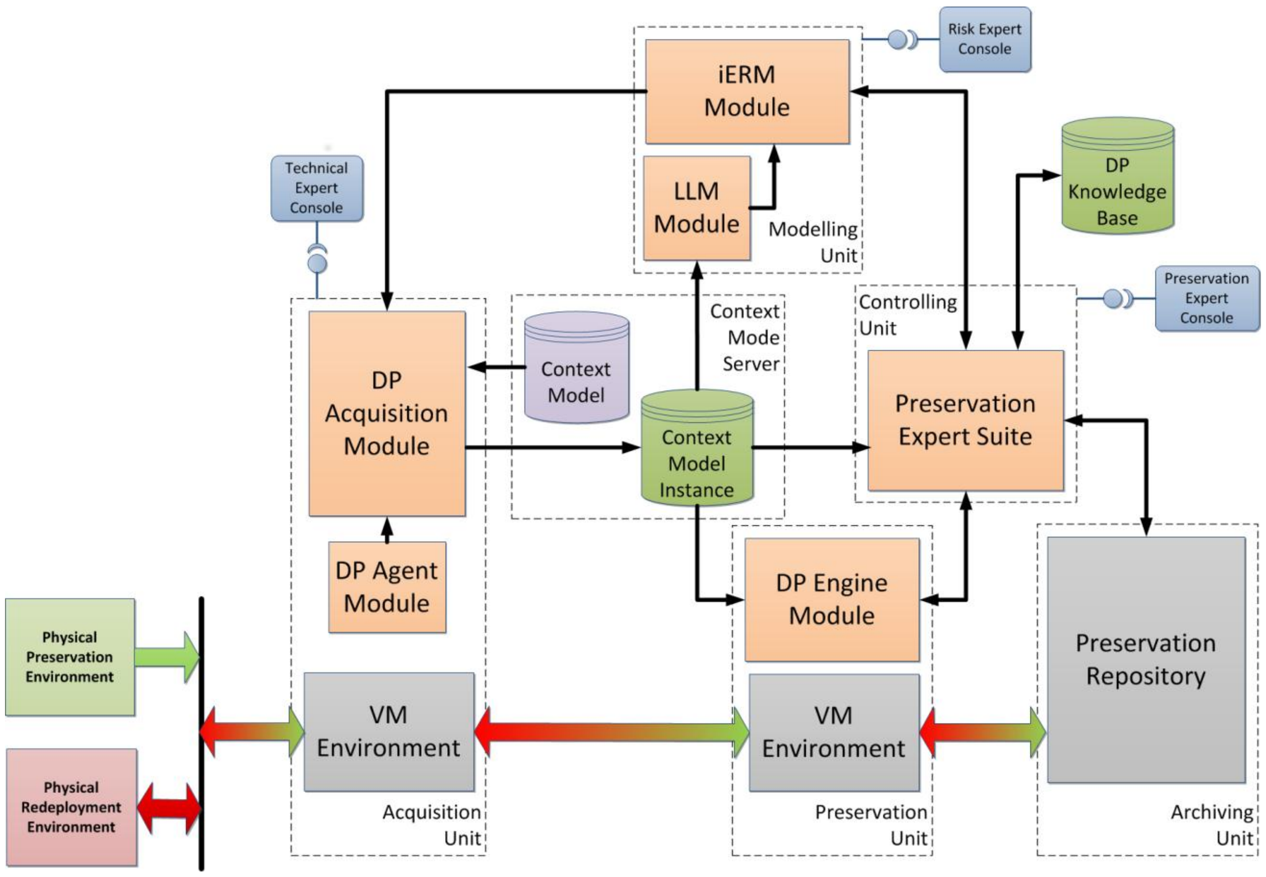


Figure 2: High level view of the TIMBUS DP architecture

3.2.4 Digital Preservation

Although Digital Preservation has been traditionally driven by memory institutions and the cultural heritage sector, it is increasingly recognized that it is a problem affecting all organizations that manage information over time, and as such it affects most of contemporary organizations where information systems provide important support to the business. Although the OAIS Reference Model [3] remains an important source of concepts to the field, it lacks directives and guidelines to address complex preservation scenarios with multiple business support systems and complex digital objects in place. In such scenarios, Digital Preservation requires a holistic view, acting as a combination of organisational and business aspects with system and technological aspects, so that all the contextual aspects surrounding a complex digital object can be captured and the objective of rendering it in the future in the same or in similar conditions can be attained.

With this holistic concern in mind, digital information life cycle models have been designed, among which the DCC Curation Life Cycle Model [12] and the SHAMAN Information Life Cycle [13] are noticeable examples. The DCC Curation Life Cycle Model elongates the traditional scope of preservation to include curation. It addresses two phases: a Curation phase, which might involve the creation of new information or the access and reuse of already existing information and its appraisal and selection; and a Preservation phase, which involves the ingestion of the information into the archive, the application of preservation actions, and the storing of that information. During the two phases, community watch and participation and preservation planning take place in order to keep descriptive metadata and representation information up to date. The SHAMAN Information Life Cycle, besides including the Archival phase already addressed by the OAIS model, suggests two additional pre-ingest phases and two additional post-access phases. The pre-ingest phases Production and Assembly aim at the capturing of the context of production of the object and its assembly into an information package, respectively. The production and the assembly are important aspects for business processes and are part of the planning phase in the TIMBUS process framework. The post-access phases Adoption and Use of the SHAMAN Information Life Cycle concern the preparation of the retrieved package so that its information contents can be used. The access to an archived business process is realised as redeployment phase in the TIMBUS framework.

The Preservation Networks [14] of the CASPAR project are a relevant reference for the capturing of the dependencies of complex digital objects through the usage of entity-relationship-like models, although business and organisational aspects are left out of it. The TIMBUS context model captures the context including the dependencies and relationships of business processes. It is extensible for domain specific needs.

The processes defined in the framework can be implemented and supported by a range of existing work in the area of Digital Preservation. The research on preservation strategies has predominantly focused on migration and emulation. Migration involves the converting of objects to operate in a different technical environment (in terms of hardware and software) than originally intended [15]. Research on migration can be found within the SCAPE project[7], which evaluated 40 migration tools, ranging from image converters to database migration suites, of which the ones considered most adequate for SCAPE's large scale requirements were chosen [16]. Emulation is a widely used concept in information technology. For software engineering it is often used as development aid if the system architectures for development and application of the software differ (e.g. software for hand-held systems is usually tested in an emulated environment). With emulation it becomes possible to run legacy software in production environments [17] [18] [19]. Both strategies can be used to preserve parts of the information system supporting a business process, but new approaches need to be envisaged to address problems of external dependencies such as web services. A discussion about potential strategies is given in Section 3.4.2.4.2. Work on the evaluation of preservation strategies with focus on static data objects has been done with the PLANETS[8] and SCAPE project. The resulting preservation planning tool is called Plato, a publicly available web-based decision support tool with access to a distributed architecture of preservation services [20].

[7] http://www.scape-project.eu/

[8] http://www.planets-project.eu/

The on-going research project wf4ever[9] addresses approaches to publish and share scientific workflows by resolving execution platform dependencies. The aim of the project is to make the workflows preservable and reusable. Their approach is to create an abstract representation of the workflow that is represented in Open Provenance Model (OPM) [21] as Linked Data[10]. The project shows a practical approach to abstract and describes scientific workflows so that they can be easily shared across organisations. Nevertheless, the abstraction approach introduces some limitations for the preservation of significant properties that are not represented by the abstraction layer. For business processes a more comprehensive approach is required taking into account the context of a process including legal aspects as well as hardware specific properties e.g. the use of specific sensor hardware.

3.3 TIMBUS process framework

This section introduces the fundamental concepts of the TIMBUS process framework. The three-phase TIMBUS approach is introduced in Section 3.3.1. The involved stakeholders are described in Section 3.3.2. Three different process scenarios are considered in the framework as described in 3.3.3. This section further defines core artefacts of the preservation possess and environments of the process.

[9] http://www.wf4ever-project.org/

[10] http://linkeddata.org/

3.3.1 TIMBUS approach

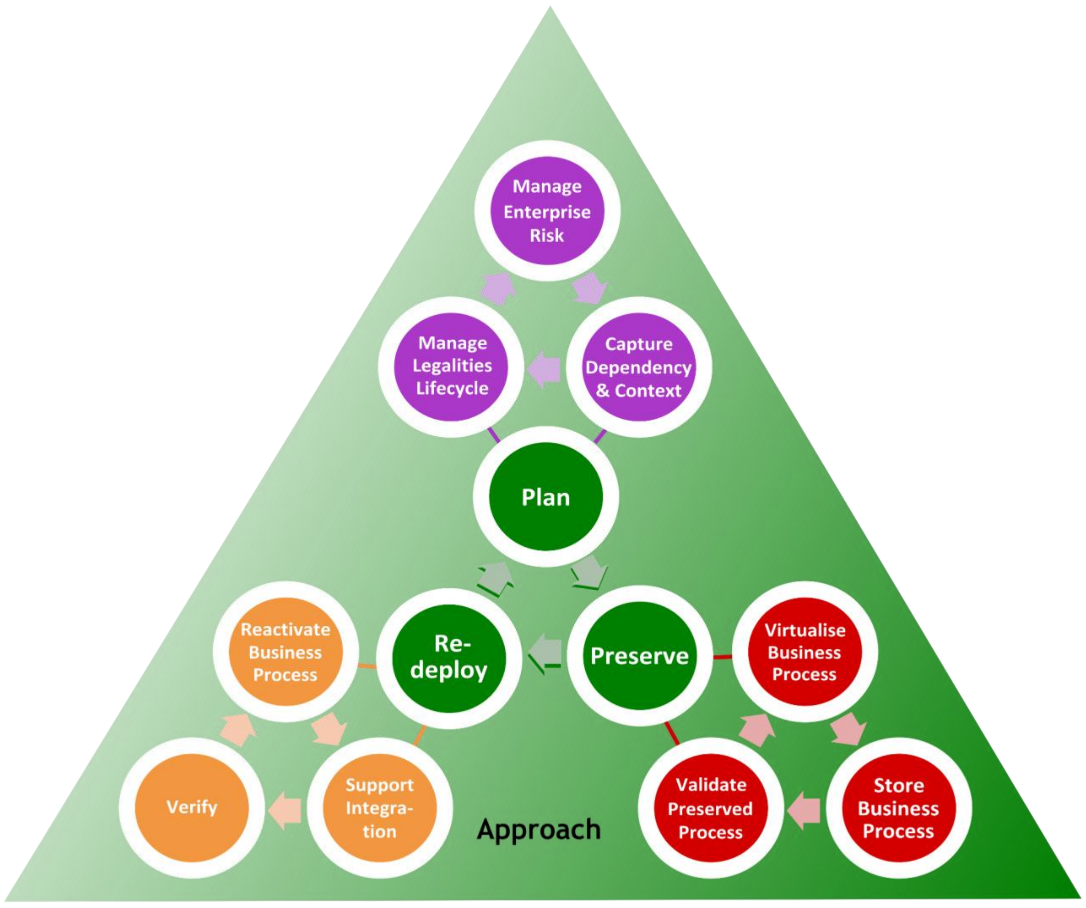

Manage Enterprise Risk
Manage Legalities Lifecycle
Capture Dependency & Context
Plan
Reactivate Business Process
Re-deploy
Preserve
Virtualise Business Process
Verify
Support Integra-tion
Approach
Validate Preserved Process
Store Business Process

Figure 3: TIMBUS Three phase approach

The TIMBUS approach to digitally preserve business processes can be divided into three phases: plan, preserve and redeploy (as shown in Figure 3). Digital Preservation is seen from a risk management perspective within the TIMBUS project. In the planning phase, a risk assessment of existing business processes is performed, the context of the process is captured and potential preservation solutions are evaluated. The preservation phase executes the preservation strategy based on the capturing of the business process and context from source systems and its preparation for archival storage. The last phase – redeployment – retrieves the process from the archive at some time in the future and redeploys it for re-execution.

Planning

In TIMBUS, the preservation of business processes is based on a risk management approach. In the planning phase a risk analysis of existing business processes is performed. The risk analysis is done on different levels within the organisation, starting from an enterprise wide identification of business processes critical in the future, supported by reviewing contractual and legal obligations, drilling down to detailed analysis of technical risks of current implementations. The risks that jeopardize the long term availability of business processes are identified and evaluated. For a detailed analysis of business process, the relevant components and dependencies are captured in the TIMBUS context model [22] [10] [11]. The model defines technical, organisational and legal aspects of a process. The legal and contractual aspects of a business process can be administered by the legality lifecycle management tool [23] provided by TIMBUS.

Potential preservation approaches for the business process are identified and evaluated against the specific requirements of the settings. The preservation includes the capturing of required process data from the source environment, application of preservation actions and migration of the process data into an archival format.

Within our risk management approach, Digital Preservation is considered as one possible risk mitigation strategy. The risk management is responsible for continuous monitoring of risk factors for active business processes as well as for archived business processes.

Preservation

In the preservation phase, the business process is captured from the source system. In order to ingest a business process into the archive, preservation actions have to be applied. Such actions are for instance emulation of external software or hardware, migration of data or the virtualisation of systems. The dependencies and relationships between the components of the business process need to remain intact over time. In order to verify the characteristics, behaviour and performance of the captured process in the future, validation data is captured. These will become useful whenever the process has to be redeployed. They are important for testing the success of the redeployment. Also monitoring parameters are specified and stored together with the process data in the archive.

Redeployment

The redeployment phase defines the reactivation of the preserved business process in a new environment at some time in the future. The redeployment procedure needs to be adjusted to the new environment. The required services need to be redeployed on a new (hardware and software) infrastructure and the process needs to be deployed. The access and usage of the redeployed services and retrieved data need to be adjusted to the new organisational structures and legal conditions. Then, the redeployed business process is validated against the metrics that have been defined by the original process.

3.3.2 Stakeholders

Different stakeholders, having different concerns regarding the preservation of the business process, are involved in the preservation process. Table 1 lists these stakeholders and provides a definition of their concerns. The stakeholder identification was driven by consolidating common role models of business processes, preservation models and identified concerns of the TIMBUS process steps. Furthermore, the TIMBUS use cases were used to verify the stakeholder definition. These concerns need to be realized by actors in the organisation - there has to be a mapping between the TIMBUS stakeholders and actors in the organisation, although this might not be a 1:1 relation. In order to ensure that all relevant concerns for the preservation are considered in the process, the relevant actors should be identified. Depending on the process and the structure of the organisation, stakeholders can be represented by several actors. An example of the stakeholder match can be found in Section 3.6.1 for the civil engineering use case.

Table 1: Stakeholders

Process Owner	The Process Owner has interest in the preservation of the business process. He is responsible for the business process, business data and the business rules.
Process Operator	The Process Operator performs the work necessary to archive the objectives of the business process. The operator represents the customer reusing the preserved process in the future.
Technology Manager	The Technology Manager is responsible for the technological system continuity and the deployment of technological infrastructure of the institution.
Technology Operator	The Technology Operator is responsible for the operation of the IT services and infrastructure archiving service level agreements.
Preservation Manager	The preservation manager (also repository manager) controls the long term preservation and retrieval of digital assets in the institution, defines strategies and sets goals for preservation in the organisation.
Preservation Operator	The Preservation Operator is responsible for the operation of the preservation and redeployment process, and the archive.
Executive Manager	The Executive Manager's responsibilities constitute the strategic decision making on an organization level. The Executive Manager creates the enterprise-wide business plans and associated resourcing plans.
Operational Manager	The Operational Manager is responsible for the operational decisions in the day-to-day business. The role is responsible to manage the resources (including budget) and processes in the organisation.
Risk Manager	The Risk Manager identifies, assesses, and risks. This includes analysing the value of assets to the business, identifying threats to those assets, and evaluating how vulnerable each asset is to those threats. The Risk Manager can be further specified as described in [24].
Legal Expert	The Legal Expert is responsible for the legal and contractually issues of the business process.
Auditor (external)	The auditor is responsible for certifying that the organization practices, the system's properties and the operational environment comply with established standards and regulations.
Regulator (external)	The regulator is an external entity imposing rules concerning the preservation of digital assets, such as legislation and standards. Example is the government enacting data protection regulations.

3.3.3 Process scenarios

In the preservation of business processes we are facing different scenarios depending on the implementation of the system and the requirements of the setting. Three different types of scenarios can be identified within TIMBUS: deployed, automated and orchestrated.

- In a deployed scenario, the processes run partially supported by information systems, and while it is possible to capture some of the software (SW) components and configuration data, it is not possible to capture all relevant information, such as the system state, or instances of the process.
- In an automated scenario, the processes run completely supported by information systems, and while it is possible to capture the SW components, configuration data, and other relevant data (e.g. system state), it is not possible to capture single executions (instances) of the process. This scenario is similar to the hibernate function existing in several information systems.
- In an orchestrated scenario singe executions of the process are identified and stored. Besides the possibility to capture SW components and configuration data, it is also possible to capture instances of the process (which dismisses the need for capturing the whole system state).

Table 2: Process scenarios

Processes	SW Components	Config Data	System State	Single instance
Deployed	**X**	**X**		
Automated	**X**	**X**	**X**	
Orchestrated	**X**	**X**		**X**

3.3.4 Artefacts

Different artefacts are used, processed, generated and exchanged by the different process steps within the TIMBUS framework. The core artefacts are listed and described in Table 3.

Table 3: Artefacts

Abbreviation	Artefact	Description
RAR	Risk Assessment Report	RAR is the result of the Risk Assessment Management process (as described in Section 3.4.2.1 and [25] [24]). It evaluates the risks of a specific business process setting according to their expected likelihood and impact.
CMI	Context model instance	CMI specified the context of a specific business process including technical, organisational and legal aspects. The context model is described and specified in [22].
PPP	Process Preservation Plan	The Process Preservation Plan (PPP) specifies actions to digitally preserve a business process.

		It includes • A specification of the significant properties of the process, • methods to acquire the relevant data from the source system, • preservation actions for the components of the business process • validation and verification mechanisms including measurement points and data • archival storage format for the business process • redeployment procedure • evaluation of the plan against the preservation requirements • monitoring events for review of the preservation plan.
V&V Plan	Validation and Verification Plan	The V&V plan specifies the validation and verification procedure for the preserved and redeployed business process. The Goal of the V&V Plan is to provide measurements to check whether the significant properties of the process were preserved over time.

3.3.5 Environments

The TIMBUS process interacts with two different environments listed in Table 4 (as defined in the TIMBUS architecture [6]).

Table 4: Environments

Environment	Description
Source environment	Execution environment of the original business process that will be preserved. The environment includes technical as well as non-technical resources (e.g. organisational aspects, legal aspects).
Redeployment environment	The target environment which is the combination of technical and non-technical resources forming an infrastructure to re-deploy the preserved business process.

3.4 Digital Preservation Process for Business Processes

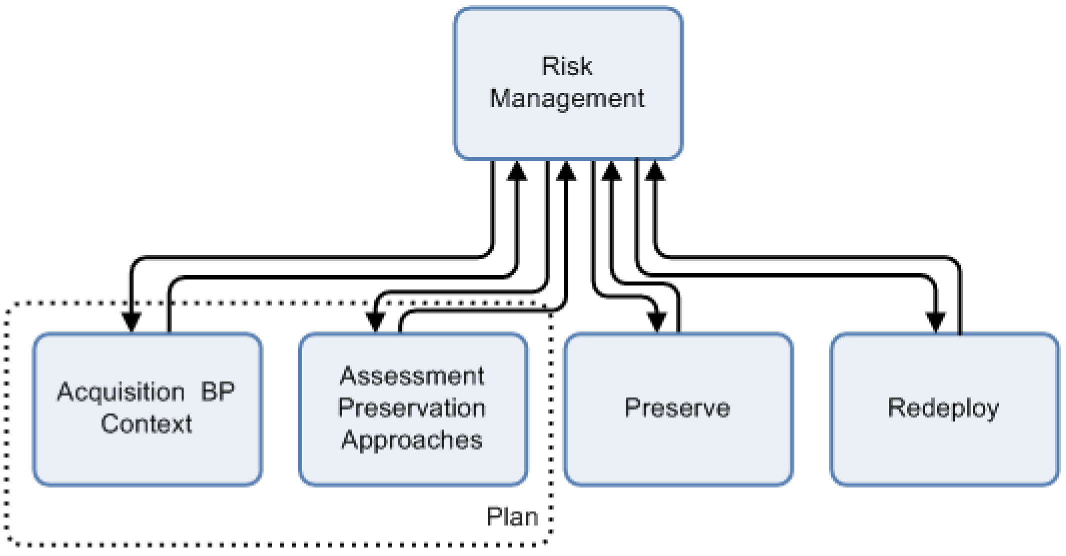


Figure 4: TIMBUS Process High Level View

Figure 4 shows the high level processes of the TIMBUS framework. The preservation of the business process is seen from risk management perspective. The TIMBUS process can be divided into three phases: plan, preserve and redeploy.

In the first phase the risk management performs a risk analysis of existing business processes including an assessment of processes that need to be accessible in the future. In order to assess the risks of a process the context and the dependencies of the business process are captured and modelled (*Acquisition business process context*). For processes with potential risks of becoming unavailable in the future potential preservation strategies are identified and evaluated (*Assessment Preservation Approaches*). This includes processes with external components and those which are beyond the influence of the process owner.

For risk mitigation, the most suitable preservation strategy, depending on the individual requirements is selected. The execution of the preservation strategies and transformation of the business process into an archival format is done in the *Preserve* process. The *Redeployment* process reactivates the archived business process in a new environment at some time in the future.

In the following sections, the process steps of the three phases are described in more detail. An overview of the detailed TIMBUS process framework is given in Figure 5 showing the process steps of all three phases. The figure shows that all processes of the TIMBUS framework are triggered by the Risk Management (described in Section 3.4.2.1). The planning phase consists of the *Acquisition business process context (*described in detail in Section *3.4.2.2*) and *Assessment Preservation Approaches* (see Section 3.4.2.3). The process steps of the preservation phases are presented in Section 3.4.3. The redeployment process is described in Section 3.4.4. For a better overview tables summarise the process steps with key information. For all steps the inputs and outputs are defined in the table. The support for each process step by the TIMBUS architecture [6] is presented as well. An overview of the architectural view of the TIMBUS process is given in Section 3.5 of this document. The stakeholders (as defined in Section 3.3.2) involved in the process are defined by using the RACI method. RACI[11] describes the participation of roles in completing task or processes. An overview about the assignment is given in Section 3.4.1. A short description of all process steps is given in the table.

Process		
Risk Management	**Input**	Source environment, expert input, business goals, regulations
	Output	Risk Assessment Report (RAR): Identified risks, evaluated risks, risk mitigation strategies, triggers for re-evaluation of risks
	TIMBUS Architecture	IERM Module
	Stakeholder involved	R,A: Risk manger I: Execute Manager C: Process Owner, Technology Manager, Preservation Manager, Operational Manager, Legal Expert, Auditor, Regulator
	Description	The risk management is a central component of the TIMBUS process. It is responsible to identify, analyse and evaluate risks. Starting from ISO 31000 [4] as a generic risk management approach the TIMBUS framework integrates Digital Preservation within risk management. The risk management compares different risk mitigation strategies and supervises the execution of risk mitigation strategies. The monitoring of risks is an on-going activity for the archived business process and running business process. It can re-invoke the risk analysis and evaluation process.
Acquisition business process context	**Input**	Source environment, process documentation
	Output	Context model Instance of the business process with different views of the stakeholders (CMI)
	TIMBUS Architecture	DP Agent Module, DP Acquisition Module, Legal Life-Cycle Module, Preservation Exert Suite
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: all stakeholders I: Risk Manager
	Description	The context of a business process is captured and documented. The context includes technical, organisational and legal aspects of a process. Different views of the context are developed for specific concerns of the stakeholders.
Assessment	**Input**	RAR, CMI
	Output	Process Preservation Plans (PPP): including list of potential preservation

[11] RACI: R-Responsible, A- Accountable, C-Consulted, I-Informed

Preservation Approaches		strategies evaluated against identified requirements
	TIMBUS Architecture	Preservation Expert Suite, DP Engine Module
	Stakeholder involved	R,A: Preservation Manager C: Process Owner, Process Operator, Technology Manager, Operational Manager and Legal Expert I: Risk Manager
	Description	Potential preservation strategies for the business process are identified and evaluated against the preservation requirements. That includes procedures for capturing the process from the source system, applying required preservation actions, and the migration into the archival format. The resulting Process Preservation Plan also includes procedures for redeployment and validation of the preserved process.
Preserve	**Input**	Process Preservation Plans (PPP)
	Output	Business process stored in the archive (with performed preservation action, additional metadata, procedure for redeployment and verification) and monitoring triggers for re-evaluation of DP strategy.
	TIMBUS Architecture	DP Agent Module, Preservation Expert Suite, DP Engine Module, Preservation Repository, Risk Management
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: Technology Operator, Operational Manager I: Process Owner, Risk Manager
	Description	In this step data of the business process is captured from the source environment and prepared for archival storage. The required preservation actions defined in the Process Preservation Plan are executed. Quality assurance ensures the correct output of the performed actions. Verification data used during a later redeployment is captured. The business process is stored in the archived and monitor triggers for re-evaluation are submitted to risk management.
Redeployment	**Input**	business process stored in archive
	Output	Redeployed business process in new execution environment
	TIMBUS Architecture	DP Acquisition Module, Preservation Expert Suite, DP Engine Module, Preservation Repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Risk Manager I: Process Owner
	Description	In this step the archived process is redeployed in a new environment. The

		redeployment procedure is adapted to the redeployment environment. The preserved services and data need are installed. The redeploy process needs to be verified for correctness and completeness.

3.4.1 RACI – Responsibility assignment

The Responsibility Assignment Matrix also known as RACI allows to assign responsibilities of roles for processes. Table 5 shows the responsibilities of the roles for the high level processes of the TIMBUS framework (shown in Figure 4). The table shows that the preservation process requires a number of consulting roles that not executing tasks but contributing expertise and input. The Process Operator, Technology Manager, Technology Operator, Operational Manager, Legal Expert, Auditor and Regulator are consulting roles. Executive roles are the Preservation Manager, Preservation Operator and the Risk Manager who execute tasks of the TIMBUS process. Figure 5 shows the accountable roles for the process steps of the TIMBUS framework. The table shows that the Process Owner, Preservation Manager and the Legal Expert are involved in all phases of the process.

Table 5: RACI

	Risk Manage-ment	Acquisition business pro-cess context	Assessment of DP	Preserve	Redeployment
Process Owner	C	C	C	I	I
Process Operator		C	C		
Technology Manager	C	C	C		
Technology Operator		C		C	C
Preservation Manager	C	A	RA	A	A
Preservation Operator		R		R	R
Executive Manager	I	C			
Operational Manager	C	C	C	C	
Risk Manager	RA	I	I	I	C
Legal Expert	C	C	C		C
Auditor	C	C	C		
Regulator	C	C	C		C

R	Responsible	The person executing the task.
A	Accountable	The person accountable for signing off the work done by the responsible person.
C	Consulted	Persons that are consulted during the execution of the task. They are not executing the task but hold information that are required.
I	Informed	Those who are updated on the progress of the task.

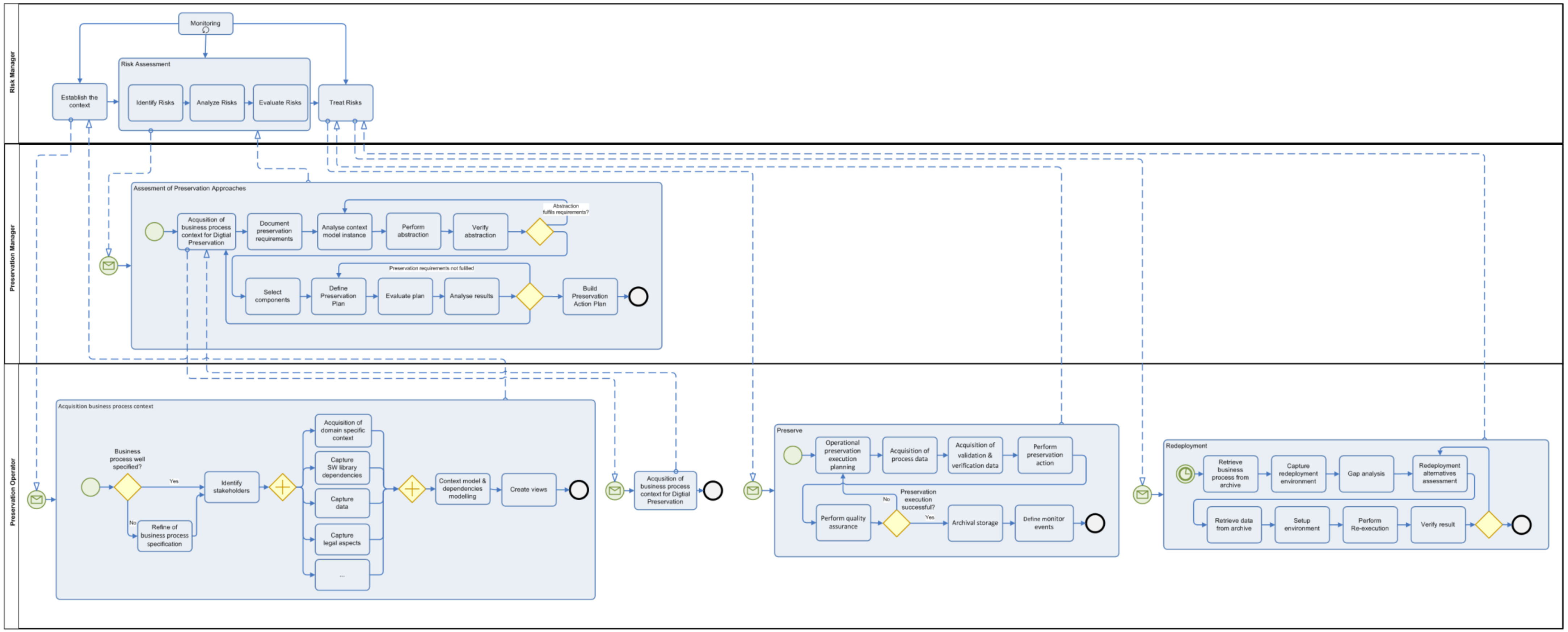

Risk Manager
Preservation Manager
Preservation Operator
Monitoring
Risk Assessment
Establish the context
Identify Risks
Analyze Risks
Evaluate Risks
Treat Risks
Assesment of Preservation Approaches
Acqusition of business process context for Digital Preservation
Document preservation requirements
Analyse context model instance
Perform abstraction
Verify abstraction
Abstraction fulfils requirements?
Preservation requirements not fulfilled
Select components
Define Preservation Plan
Evaluate plan
Analyse results
Build Preservation Action Plan
Acquisition business process context
Business process well specified?
Yes
No
Refine of business process specification
Identify stakeholders
Acquisition of domain specific context
Capture SW library dependencies
Capture data
Capture legal aspects
...
Context model & dependencies modelling
Create views
Acqusition of business process context for Digtial Preservation
Preserve
Operational preservation execution planning
Acquisition of process data
Acquisition of validation & verification data
Perform preservation action
Preservation execution successful?
No
Yes
Perform quality assurance
Archival storage
Define monitor events
Redeployment
Retrieve business process from archive
Capture redeployment environment
Gap analysis
Redeployment alternatives assessment
Retrieve data from archive
Setup environment
Perform Re-execution
Verify result

Figure 5: Detailed TIMBUS Process

3.4.2 Planning

The first phase of the three-phase TIMBUS process is responsible for planning the preservation. Within the planning phase the business process in the institution is acquired. In order to preserve and later redeploy the process, the context needs to be captured and documented (e.g. technical, legal and organisational aspects). Suitable preservation strategies for the process are identified and evaluated in this phase. Figure 4 shows the high level view of the TIMBUS process, the planning phase consists of two steps: *Acquisition business process context* and *Assessment Preservation Approaches*.

The TIMBUS approach is risk management based. Risk management is a continuous process and interacts with all three phases of the TIMBUS process (as shown in Figure 6). Section 3.4.2.1 presents the core concepts of risk management. A more detailed description of the TIMBUS risk management can be found in [25] .

3.4.2.1 Risk Management

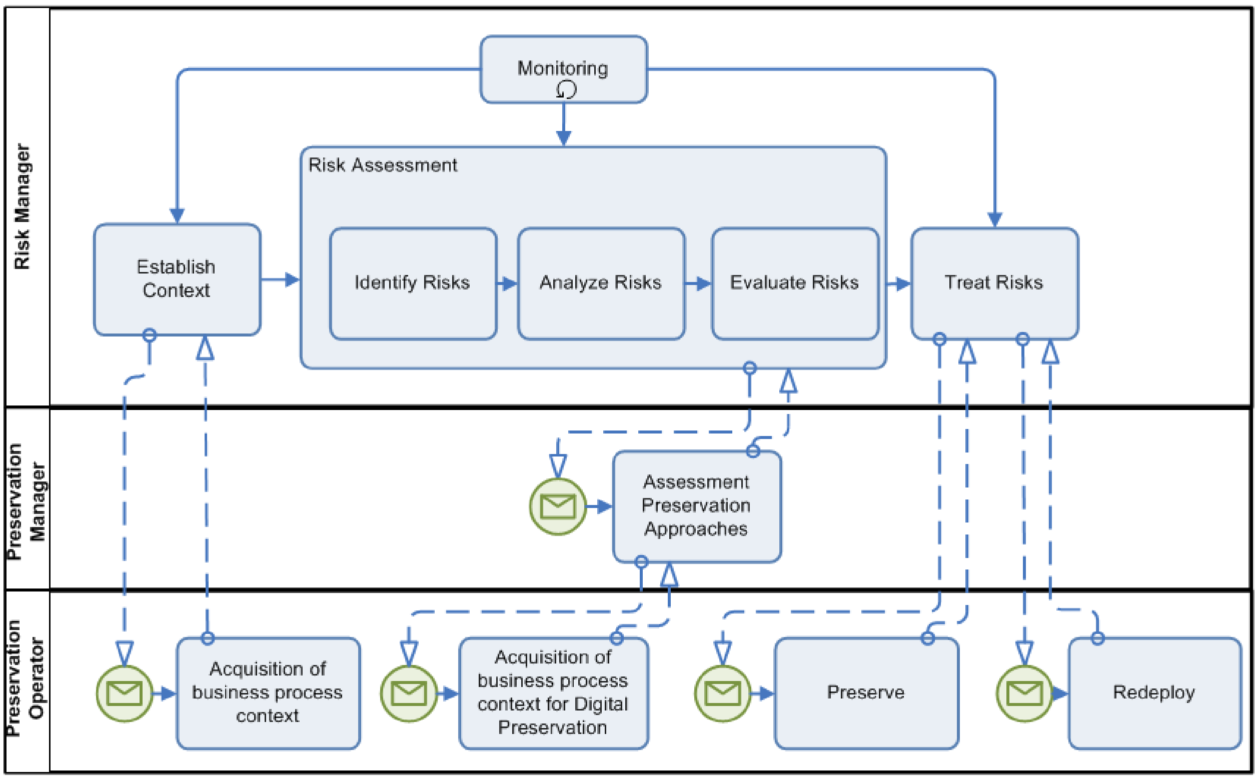


Figure 6: TIMBUS process

A risk can be characterized as the combination of the probability of an event taking place and its consequences [26]. Risk management is a well-established field with the goal of recognizing, preventing and controlling risks related with assets and activities. It is a common practice in managing organizations and projects, offering standardized methods for qualifying, quantifying and assessing risks, and deliver risk treatment. Digital Preservation is a possible mitigation strategy to treat risks affecting the long term availability and usability of business processes. The selection of a suitable risk treatment requires comparable evaluation of potential strategies under consideration of economic aspects.

TIMBUS uses a generic enterprise risk management process based on the ISO 31000 [4]. A detailed description about risk management within TIMBUS can be found in [25]. TIMBUS defines different procedural interfaces to connect Digital Preservation with enterprise risk management. The risk associated with a process can act as a driver prompting its preservation as a way of mitigating the threats that endanger the process. The risk management process identifies and evaluates different risks in a structured and well defined manner by determining the potential impact and likelihood of the risk. An example of the risk management process applied in TIMBUS can be found in [27] for the civil engineering use case and in [28] for the scientific process.

Figure 6 shows the process steps of risk management and the interface to the TIMBUS process. In a first step the context of the business process is established. The *Acquisition of the business process context* process (described in Section 3.4.2.3.1) supports the capturing. The result is a context model instance (CMI) of the process. During the *Risk Assessment* risks associated with a process are identified by using CMI. Potential preservation mitigation strategies are provided to the risk management by the *Assessment of Preservation Approaches* process (see Section 3.4.2.2). The enterprise risk management process then decides on the best matching solution. The risk management then triggers the preservation phase, which is described in 3.4.3.

The redeployment of preserved processes represents also a potential risk mitigation strategy. Examples are compliance of preserved processes with obligations and guidance or proofing process steps or process instances of preserved process. The redeployment process (described in Section 3.4.4) can be triggered by the risk treatment step as shown in Figure 6. The redeployment of a preserved process can also be triggered outside the risk management by demand of other tasks and process in the organisiation.

Establish Context	**Input**	Source environment, business process documentation
	Output	CMI
	TIMBUS Architecture	The task is implemented within the TIMBUS project as *Acquisition of the business process context* process (described in Section 3.4.2.3.1).
	Stakeholder involved	R,A: Risk Manager
	Description	In order to manage the risk of a business process, the context of the process needs to be specified. The TIMBUS context systematically specifies aspects of a process including technical, organisational and legal. The model instance (CMI) describes the process and the related context and is further used for the risk assessment.
Identify Risks	**Input**	CMI

	Output	List of identified risks for the business process
	TIMBUS Architecture	Risk Model Builder
	Stakeholder involved	RA: Risk Manager C: Process Owner, Technology Manager, Preservation Manager, Operational Manager, Legal Expert, Auditor, Regulator
	Description	In this step the risks of the business process are identified. Structured approaches to identify the source of the risks, the impact area, events and the potential consequences are used. Potential methods for the risk identification are described in [24].
Analyse Risk	**Input**	List of identified risks
	Output	Report about likelihood and impact of identified risks
	TIMBUS Architecture	iERM Module: Risk Annotation
	Stakeholder involved	RA: Risk Manager
	Description	In this step the identified risks are analysed according to their likelihoods and impacts.
Evaluate Risk	**Input**	Report of analyse risk
	Output	RAR, PPP
	TIMBUS Architecture	Risk Impact Assessment
	Stakeholder involved	RA: Risk Manager I: Executive Manager
	Description	In this step the risks and potential mitigation strategies are evaluated. Evaluation techniques are presented in [24]. The *Assessment of Preservation Approaches* process provides input to the evaluation step of the risk management. An important criterion for the evaluation are cost benefit analyses. Cost considerations for DP are also discussed in [24]. The output of this step is whether risks are acceptable/ tolerable or which risk treatments are required to mitigate the risks.

Treat Risk	**Input**	PPP
	Output	Preserved business process in archive
	TIMBUS Architecture	Within TIMBUS the risk treatment is executed as preservation phase, which is described in Section 3.4.3.
	Stakeholder involved	RA: Risk Manager I: Executive Manager
	Description	The risk treatment addresses the execution of risk mitigation strategies and the evaluation of their effectiveness. The TIMBUS process considers Digital Preservation as a risk mitigation strategy.

3.4.2.2 Acquisition of business process context

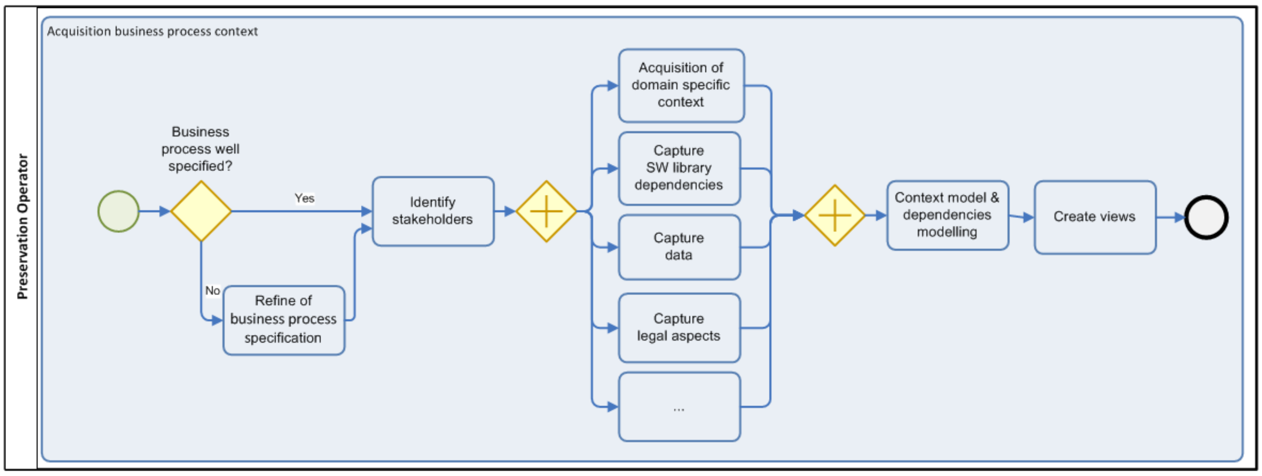

Preservation Operator
Acquisition business process context
Business process well specified?
Yes
No
Refine of business process specification
Identify stakeholders
Acquisition of domain specific context
Capture SW library dependencies
Capture data
Capture legal aspects
...
Context model & dependencies modelling
Create views

Figure 7: Acquisition of business process context

This task is responsible for acquiring the context of a business process. This process is triggered by the risk management process as shown in Figure 6. The captured information is modelled as an instance of the TIMBUS context model [10] [22]. The acquisition process is shown in Figure 7. The first step is to determine if the business process is defined and specified in a sufficient way for further processing. If not, the refinement of the business process specification is started. In many cases processes are vaguely defined, clear specifications are often absent. Processes in the organisation are continuously changing and improving, as a result the formal documentation is often out-dated. Business processes in organisations are strongly connected and report to each other. For their preservation, thus, a clear specification of the boundaries is required. The business process to be preserved is captured, specified and documented in this acquisition process. For modelling and documenting business processes a wide range of frameworks are available, e.g. BPMN [29] or ArchiMate [8], along with a number of tools supporting them. In cases where the process model is either not available or not up-to-date, experts are supported with techniques from the area of process mining in order to retrieve a formal model of the process [30].

Once the business process is specified, the next step is to identify the stakeholders that have interest in the preservation of the process. A list of stakeholders for the preservation of business processes is presented in Section 3.3.2. The different stakeholders are interested in specific subsets of information about the process, which are called concerns [31]. These concerns are captured and are useful information for identifying the aspects that need to be captured from the business process. The identified concerns are used to create different views of the business process and its context in the last step of the business process acquisition process. A view represents the business process from a perspective of a set of concerns.

After the stakeholders and their concerns are identified, domain specific context information is gathered from the source environment. The *Acquisition of the business process context* is executed twice during the TIMBUS approach. As shown in Figure 6 the process is triggered once by the *Risk Management* and the second time by the *Assessment Preservation Approach* process*.* The two iterations of the process address different concerns of stakeholders. The context model instance of the first iteration is used to identify risks and analyse risk of the business processes. The second iteration is used for the planning and assessment of preservation approaches. An example of these different concerns is shown for the civil engineering use case in Section 3.6.1.

The different aspects of the business process can be extracted in parallel. All relevant context information that is required to address the concerns of the stakeholder is captured. Context information includes amongst others organisational aspects such as actors involved, technical aspects such used technologies, system components both on hardware and software levels, terms of service of external service providers and legal obligations. The context is often captured in domain specific languages such as database models, organisational charts, models of software components that are installed on a system or hardware models. Automatic tools can support the extraction of information from source systems, e.g. detection of software components. The tool support for populating the context model within the TIMBUS project is described in [32].

In a next step, *Modelling of Context Model & Dependencies* the domain specific models are consolidated and integrated into the TIMBUS context model. TIMBUS developed a model to systematically capture all relevant aspects of a process. This model is based on the structure and concepts of ArchiMate. It provides a domain-independent core structure, while domain-specific ontologies can be used to extend the basic model. Depending on the domain of the business process, different specific models can be integrated into the basic structure. An important aspect of the model is the relationships between the components. For successful preservation, the relationships and dependencies between the components need to be maintained over time. The model is implemented as an ontology, which allows both for the hierarchical categorisation of aspects into classes and subclasses, and for relating these aspects to each other. More details about the TIMBUS Context model are described in [10] [11] [33].

The process to instantiate the context model is a combination between bottom-up and top-down approach. Starting from high level descriptions of a business process (e.g. BPMN-Models) the process can be described in more detail (e.g. resources the steps are using, communication between steps). The bottom-up approach starts from detailed models captured in the previous step, e.g. technical infrastructure and resources that are used for executing operations. The detailed models are aggregated to larger components and artefacts, and the relationships between the artefacts are established. The combination between the two approaches helps to achieve a good capturing coverage of relevant artefacts and sufficient details for further processing. The goal is to capture all aspects of a business process that are relevant for risk management, preservation planning, preservation, archival storage and redeployment. Formal checks and pattern checks on the context model can help to indicate missing aspects. These that will be developed as part of the upcoming deliverable *D4.9 Refined Business Process Contexts*.

For some required aspects, existing tools can be used to support the automated context acquisition for domain specific context. Examples are CUDF [34] that describes technical dependencies of packages of computer system. More information about the automatic capturing of context can be found in [33].

In the last step of *Acquisition of business process* context, views of the context model are created. A view is a concrete representation of the process for specific concerns of a stakeholder. The view needs to be complete and address the concerns of its stakeholder adequately. The different views consist of different information, representation and different levels of detail. In different views detailed information can be summarized to more abstract concepts, e.g. detailed component model of a computer can be visualised as a single entry. A set of predefined views are available as template and defined by ArchiMate [8]. In a first iteration the context model is created for the assessment of risk as shown in Figure 6. The key concerns for the risk assessment are involved components, business goals, obligations and involved stakeholders [27] [28].

The outcome of the *Acquisition business process context process* is a context model instance (CMI) of the business process. The models instance describes all relevant information of the process which is needed to analyse and preserve it.

Decision: business process well specified?	**Input**	Documentation and specification of business process (models, descriptions)
	Output	Decision if business process is well specified and sufficient documented
	TIMBUS Architecture	The decision is a manual step without specific need of tool support.
	Stakeholder involved	R: Preservation Operator A: Preservation Manager
	Description	In many cases the business process is only briefly described. All steps of the process need to be specified and documented sufficiently. Business processes are often integrated and connected to other processes. Hence well-defined processes with clearly defined boundaries of the processes are

		required If the business process is not sufficient document the refinement process is triggered.
Refine business process specification	**Input**	Source environnement, documentation about business process
	Output	Specification of business process
	TIMBUS Architecture	Refinement of business process is not core focus of the TIMBUS architecture, techniques from the area of process mining can support the task [30].
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: Process Owner, Process Operator
	Description	The business process is defined and documented including the behaviour, performance, stakeholders involved and boundaries in sufficient detail. Process mining tools and methods can support this takes.
Identify stakeholders	**Input**	Documentation about business process
	Output	Requirements of the stakeholders (concern)
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: Process Owner
	Description	The concerns and interests of involved stakeholders are identified and documented. The information is used to apply different viewpoints on the context model instance (CMI). The different views represent the whole business process from different concerns. A set of predefined views is available for the TIMBUS context model. In a first iteration the context model is created for risk assessment (as shown in Figure 6).
Acquisition of domain specific context	**Input**	Source environment, available documentation about the business process
	Output	Domain specific information of the context
	TIMBUS Architecture	Agent Module: Metadata Capturer Acquisition Model: Dependency Extractor LLM: Legality Impact Assessment
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: all other stakeholder

	Description	Domain specific context information of the business process is captured and modelled. For example this could be the used resources, involved IT, organisation and regulation aspects. The output of the process is a number of domain specific models representing the context of the business process.
Modelling of context model & dependencies	**Input**	Domain specific models
	Output	Context model instance (CMI)
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R: Preservation Operator A: Preservation Manager
	Description	The domain specific information is consolidated into a CMI of the business process. The TIMBUS context model provides a basic structure that can be extended by domain specific models [10] [11] [33].
Create views	**Input**	CMI, requirements of the stakeholders (concern)
	Output	CMI with views
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R: Preservation Operator A: Preservation Manager I: Risk Manager
	Description	Creation of views for the different concerns of the stakeholders.

3.4.2.3 Assessment of Preservation Approaches

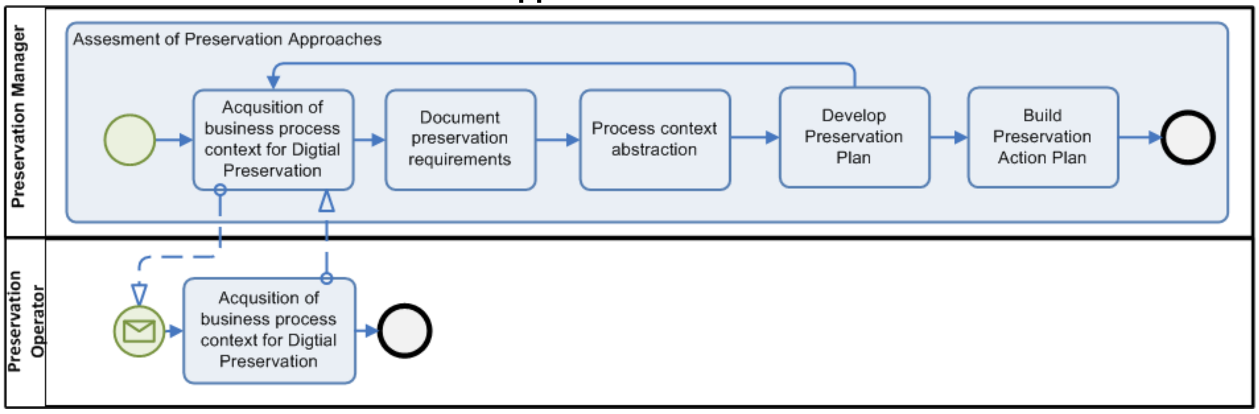

Preservation Manager
Assesment of Preservation Approaches
Acqusition of business process context for Digtial Preservation
Document preservation requirements
Process context abstraction
Develop Preservation Plan
Build Preservation Action Plan
Preservation Operator
Acqusition of business process context for Digtial Preservation

Figure 8: Assessment of Preservation Approaches

The *Assessment of the Preservation Approaches* is responsible for the specification of the preservation requirements, the identification of potential preservation strategies, and the evaluation and comparison of the strategies. The process consists of five steps as shown in Figure 8. The assessment is triggered by the risk management, asking for a risk mitigation strategy for specific identified risks of the business process. The result of the process is a list of preservation plans specifying different preservation strategies for the business process.

Acquisition of business process context for Digital Preservation

The assessment starts with a refinement of the context model instance that was created for the risk management. The context acquisition process follows the same process described in Section 3.4.2.2, but this time, the context model addresses the concerns of Digital Preservation rather than the risk management. The *Acquisition of the business process context for Digital Preservation* will be described in Section 3.4.2.3.1 in detail and the process steps will be shown in Figure 9.

The result of this process is a CMI that support the development of preservation plans for the business process. Detailed information about the technical implementation is required e.g. infrastructure, software.

Document Preservation Requirements

In the next step, the requirements for the preservation of the business process are identified. Based on policies and guidelines of the organisation and the input of the stakeholders, the requirements for the preservation solution and the preservation process are collected from a wide range of stakeholders that have an interest in the process, e.g. the Process Owner, Executive Manager, Operational Manager, IT department, and Preservation Manager. This broad stakeholder involvement helps to set the right priorities for the evaluation and leads to wide accepted solution. A number of influence factors define the requirements for a preservation solution. Examples are business goals, legal constrains, contracts, SLA (Service Level Agreements), organisational policies, re-use scenarios, etc. Involved standards, guidelines, and policies are specified in the CMI of the business process and can be extracted. The reuse scenarios present the predicted redeployment context for the business process. Based on the reuse scenarios requirements can be defined for a preservation solution.

The requirements consist of the significant properties of the process and the requirements for the preservation process. Significant properties are those essential attributes of a process, which describe its unique appearance, behaviour, quality and usability that needs to be preserved over time. The preservation process requirements define conditions regarding the preservation and redeployment process. They usually define budget and resource limitation, as well as, standards and guidelines that need to be fulfilled.

The requirements are used to evaluate different preservation approaches. Thus they need to be measurable and quantifiable. The evaluation is part of the *Develop Preservation Plan* process (see Section 3.4.2.4). Amongst others, two important aspects have to be considered specifically for business processes that are not sufficiently taken into account in existing work for preservation planning of static objects. First, requirements regarding the behaviour and performance of the process need to be specified. As a business process can have a quite complex behaviour, the separation of the business process into modules can help to set the requirements. The second aspect comprises legal and contractual obligations. The use of external products such as software, data, web services, and hardware can introduce limitations and obligations for the usage of the products. In order to develop a legally conforming preservation solution, SLAs, usage terms, contracts and laws need to be considered.

A recognised model for preservation requirements is an objective tree [35]. It provides a hierarchical structure of requirements where groups of requirements are further refined to become more detailed and measurable criteria on the lower levels. For complex systems such as business processes, the list of requirements can be very exhaustive. Hence, a hierarchical structure can help to organise the criteria so that all aspects and components of a preservation solution are addressed. This systematic approach further supports the discovery of discrepancies and contradictions in the requirements.

Work on significant properties for digital objects has been done in many Digital Preservation projects e.g. InSPECT[12] or Planets[13]. Significant properties for more complex objects have been defined for software in [36] and for interactive content [37]. Work on preservation requirements for behaviour of digital objects is presented in [38]. An overview about work on significant properties is given in [39] [40]. The preservation requirements of the civil engineering use case are specified in [1] and outlined in Section 3.6.1 of this document.

The result of this process is well defined requirements for a preservation solution of the BP. The requirements need to be specified as concrete and quantifiable criteria in a structured way.

Context Abstraction

The next step is the abstraction and generalisation of the concepts of the CMI. A detailed description of this step is provided in Section 3.4.2.3.2. The CMI contains a very detailed description of the captured business

[12] http://www.significantproperties.org.uk/

[13] http://www.planets-project.eu/

process. Not all implementation details are relevant for the preservation and the later redeployment and can thus be replaced by higher level concepts for the sake of simplification.

One example for generalisation of implementation details is the information of storage hardware implementation that can be abstracted to the storage interface. Another example could be or organisational aspects such as specific skill sets of employees that are involved in the business process. These can be generalised to role and skill models. The level of abstraction and the aspects that can be generalised depend on the specific setting, scenario and the preservation requirements. The abstraction of technical details can improve the development for preservation actions of the components (e.g. replacement through alternative implementations and use of emulators). In order to ensure trustworthy preservation, all changes and modifications in the CMI need to be documented. As the abstraction of details implies a loss of information, the performed modifications need to be verified with regard to the preservation requirements.

Develop Preservation Plan

Develop Preservation Plan is responsible for identifying potential preservation solutions, planning and testing the solutions and evaluating their plans against the previously defined preservation requirements. The detailed process is described in Section 3.4.2.4. The preservation plan consists of a procedure to capture the business process and its components from the source system, the required actions to maintain it for the long term and the procedure to redeploy and verify the process in a new environment.

Due to the complexity of business processes (e.g. distribution over different systems and use of external services), the preservation plan usually consists of combination of different preservation strategies. Examples for the technical aspects of a process are different migrations for static documents (e.g. the documentation of the process, contracts, etc.). For hardware component of the process, emulation approaches can maintain their functionality over time. The dependencies of the components need to be identified and documented, as modifications of components can propagate through the system and therefore interfere with the functionality of other components.

The evaluation of preservation plans determines whether the plans are able to fulfil the preservation requirement. If the developed plans fail in this regard, a new iteration of the assessment approach is started (as shown in Figure 8). The preservation of complex processes can require multiple iterations to find suitable preservation solutions. The iteration of the assessment can include:

- detailed context capturing if required artefacts were missing,
- changes of preservation requirements if the current requirement cannot be fulfilled for example through technical, financial or legal restrictions or
- other preservation approaches need to be evaluated.

The outcome of this process is a list of evaluated preservation plans.

Build Preservation Action Plan

The last step of *Assessment of Digital Preservation Approaches* is the creation of a preservation action plan. This plan specifies the workflow to execute the evaluated preservation plan, and defines conditions, triggers, and responsibilities for each step. Moreover, quality ensuring mechanisms for the execution phase are planned.

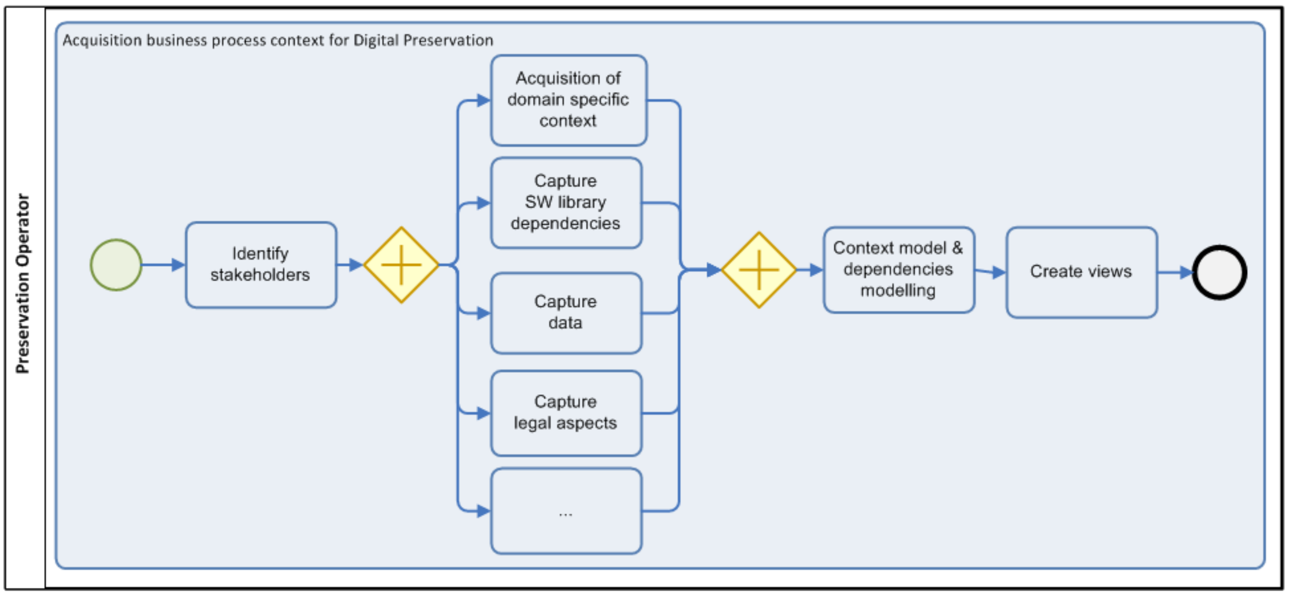
As a final step of the planning phase the evaluated strategies and the analysis are summarized. The report is submitted to risk management for decision making.

Acquisition business process context for Digital Preservation	**Input**	Source environment, CMI, RAR
	Output	CMI
	TIMBUS Architecture	DP Agent Module, DP Acquisition Module, Legal Life-Cycle Module, Preservation Expert Suite
	Stakeholder involved	RA: Preservation Manager C: all other stakeholder
	Description	The existing CMI of the business process is refined and extended for the assessment of DP approaches. For the preservation more detailed information about the implementation of the process is required.
Document preservation requirements	**Input**	CMI
	Output	List of preservation requirements for the business process
	TIMBUS Architecture	Preservation Expert Suite: Preservation Alternative Assessment
	Stakeholder involved	R,A: Preservation Manager C: all other stakeholder
	Description	In this step the requirements for a preservation solution are collected and organised. They are contributed by the different stakeholders of a business process. The requirements are derived from the redeployment scenarios, business goals, institutional policies and guidelines and legal obligations and include technical, organisational, legal and financial aspects. Requirements need to be measurable and define acceptable thresholds and knock-out criteria for preservation solutions. The requirements define the significant properties of the process. Those include all characteristics of the process that need to be preserved over time.
Abstraction of process context	**Input**	CMI, preservation requirements
	Output	CMI with abstracted concepts
	TIMBUS	Preservation Expert Suite: Model Weaver

	Architecture	
	Stakeholder involved	RA: Preservation Manager
	Description	The captured CMI contains implementation details of the business process. For later redeployment not all details are required. In order to facilitate further treatment, implementation details can be abstracted and generalised to higher level concepts. As the abstraction implies a loss of information, performed modifications of the CMI need to be verified against the preservation requirements. It needs to be ensured that all required information for the redeployment is preserved.
Develop Preservation Plan	**Input**	CMI, preservation requirements
	Output	Evaluated preservation plans
	TIMBUS Architecture	Preservation Expert Suite: Preservation Alternative Assessment, Preservation Planner, Validation and Feedback, Preservation Log Gap Detector DP Engine Module: Preservation Executer, Execution Monitor
	Stakeholder involved	R,A: Preservation Manager
	Description	This step is responsible for identifying potential preservation strategies, planning and testing the solution and evaluating the resulting plans against the previously defined preservation requirements. The preservation plan consists of the procedure to acquire the business process and its components from the source system, the required preservation actions and the procedure to redeploy the process in a new environment. The effects of applying the action are analysed against the preservation requirements. If the plan fails to fulfil the preservation requirements the assessment process is restarted to address the weak spots of the plan. The outcome of the step are evaluated preservation plans for the business process.
Build Preservation Action Plan	**Input**	Evaluated preservation plans
	Output	Evaluated Process Preservation Action Plans (PPP)
	TIMBUS Architecture	Preservation Expert Suite: Preservation Planner
	Stakeholder	RA: Preservation Manager I: Risk Manager

	involved	
	Description	The Preservation Action Plan specifies the required activities to execution the preservation approach. It defines responsibilities, conditions and triggers for the actions. Moreover quality ensuring mechanisms are planned for the execution phase. The evaluation results of the different approaches are summarised and submitted to the risk management for decision making.

3.4.2.3.1 Acquisition business process context for Digital Preservation



Preservation Operator
Acquisition business process context for Digital Preservation
Identify stakeholders
Acquisition of domain specific context
Capture SW library dependencies
Capture data
Capture legal aspects
...
Context model & dependencies modelling
Create views

Figure 9: Acqusition business process context for Digtial Preservation

The *Acquisition of business process context for Digital Preservation* follows the same procedure as the acquisition process described in Section 3.4.2.2. The only difference in the processes is that for the first acquisition the business process specification is checked for completeness (as shown in Figure 7). In this step, the existing CMI is then extended and refined. The addressed concern by the context model has changed. The first iteration of the CMI focused on the risk assessment of the process. The second iteration is used for the planning of the Digital Preservation solution for the business process. The change of the application area needs to be reflected in model instance.

Figure 9 shows the BPMN model of the *Acquisition of business process context for Digital Preservation* process that starts with the identification of the stakeholders. For the preservation two stakeholders are primarily responsible: the Preservation Manager and Preservation Operator. The concerns of the two stakeholders require detailed information about the implementation of the business process. In the next step domain specific context is captured. A detailed breakdown of the implementation of the business process is required for preservation, including software services, infrastructure, behaviour and organisational aspects. The level of implementation details that need to be acquired depends on the process scenario as defined in Section 3.3.3 and also on the preservation requirements. For deployed scenarios the infrastructure including software components and configuration can be captured. In this kind of scenario, it is not possible to capture the system state or instances of the process. The information system implementing the business process is seen as black box. The model of the business process can be preserved, but not single instances of the process execution. In an automated scenario it is possible to capture the entire system state, but not instances of the process. It includes software, configuration data, and other relevant data. The scenario is similar to hibernate and snapshot functionalities of existing BPM systems. An orchestrated scenario provides the highest grade of detail of the process. The software components, configuration data and the instances of the process can be captured. It allows full access to the components of information system. In an orchestrated scenario single process instances are identifiable and traceable through the steps of the business process. The captured information strongly depends on the domain of the process. For the civil engineering use case described in [1] specific information is required about the sensors that are used to take measurements in dams. Deliverable 4.3 [22] outlines the domain-specific ontology for sensors that is currently being developed to support the use case.

After all relevant information has been captured, the domain-specific models need to be integrated into a context model instance as described in detail in [22]. A major challenge for the preservation of processes is the maintenance of the relationships between the components over time. Changes to a component, e.g. by a preservation action can have unintentional effects on other components. The different kinds of relations between objects that were implemented in the TIMBUS context model are described in [10] [22]. The dependencies can be extracted from the source environment by tool support or established by the use of reasoning tools or they can be entered manually by an expert.

The final step of the acquisition is the creation of views in order to support the upcoming steps.

Identify stakeholders	**Input**	CMI
	Output	Requirements of the stakeholders (concern)
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: Process Owner
	Description	The concerns and interest of the stakeholders are documented. The context model instance is created for the specific concerns of stakeholders. This iteration of the CMI focuses on the preservation.
Acquisition context and dependencies	**Input**	Source environment, CMI
	Output	Domain specific models of the context
	TIMBUS Architecture	Agent Module: Metadata Capturer Acquisition Module: Dependency Extractor LLM: Legality Impact Assessment

	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: all other stakeholders
	Description	Domain-specific context information is captured and modelled for preservation of the business process. Examples are resources, involved IT systems, organisation and regulation.
Modelling of context model & dependencies	**Input**	Domain-specific models
	Output	CMI
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R: Preservation Operator A: Preservation Manager
	Description	The domain specific information is consolidated into a context model instance of the business process. Relationships between the components are identified and established in the model. The existing CMI is extended with the detail information gathered for further preservation of the business process.
Create views	**Input**	CMI, Requirements of the stakeholders
	Output	CMI including views
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R: Preservation Operator A: Preservation Manager I: Risk Manager
	Description	In this step views for the different concerns of the stakeholders are created.

3.4.2.3.2 Abstraction of business process context

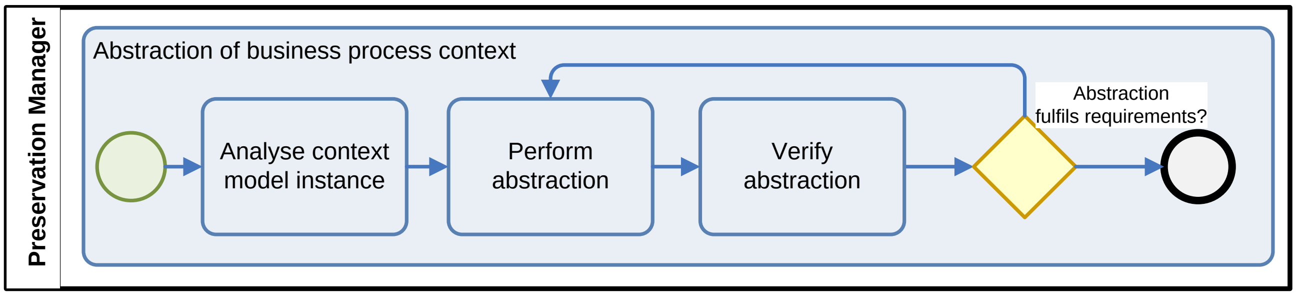

Preservation Manager
Abstraction of business process context
Analyse context model instance
Perform abstraction
Verify abstraction
Abstraction fulfils requirements?

Figure 10: Abstraction of process context

In this step the context abstraction of the business process model is performed. The context model describes the current implementation of the business process in the source system. The aim of the preservation of the business process is the maintenance of the significant properties (defined as requirements) of the process over time. For the preservation the exact implementation details might not be relevant in all technical aspects, as the focus is rather on preserving the logic and information flow of the process. Thus, implementation details may be abstracted to higher level concepts. The abstraction of technical details can facilitate the planning for preservation actions of the components (e.g. replacement through alternative implementations, use of emulators, encapsulation). The level of abstraction and the aspects that can be generalised depend on the specific setting and the preservation requirements. The abstraction causes loss of information, thus it is vital to ensure that no relevant information that is required in the future is lost during this step.

The first step is the analysis of the model to identify potential aspects for abstraction. The abstraction strongly depends on the preservation requirements. Potential abstractions are persons that can be replaced by roles with associated rights and obligations. Technical detailed implementation aspects can be abstracted to standards or used interfaces. Examples are the information of vendor specific storage hardware implementation that can be abstracted to the storage interface. Queries and patterns can be used to extract relevant aspects of the context model, but the task requires skilled personal to be executed.

The second step is the execution of the abstraction in the model. The abstracted artefacts are replaced by the new higher level concepts. Additional information about the replaced individuals such as attributes and restrictions of the replaced objects need to be reviewed and if relevant assigned to the new individual. Examples are technical characteristics such as size or performance. The relationships of the replaced individuals need to be assigned to the new artefact. All changes and modification of the context model need to be documented for later verification and traceability.

A risk of the abstraction step is the loss of information that is significant for the process. Thus in the next step the abstracted model is verified against the preservation requirements. The modified model must contain all information in all detail required to preserve and redeploy the business process. If significant properties of the process are missing after the abstraction the abstraction process needs to be re-executed.

All the modifications that have been applied to the model and the verification of the performed abstraction need to be documented and preserved for the authenticity of the preserved process. The outcome of the abstraction process is a context model where implementation details are encapsulated by abstraction. The abstraction performed in this task specifies the process that is preserved for future redeployment.

Analyse context model instance	**Input**	CMI, Preservation requirements
	Output	Suggestions for abstraction of CMI
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R,A: Preservation Manager
	Description	The context model represents the actual implementation of the business process. Not all implementation details are required for preservation. Based on the preservation requirements, the information of the context model can be abstracted and summarised to a higher level. It eases the preservation of the business process. Example: the actual instance of a hard disc (vendor, version and technical data) of a computer. In many cases only the storage interface is important other technical characteristics such as speed and cache are not significant, the hard disc card can be abstracted to more generic level e.g. SATA HD Storage with specific capacity.
Perform abstraction	**Input**	CMI, suggestions for abstractions
	Output	CMI with abstracted concepts
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	R,A: Preservation Manager
	Description	In this step the abstraction is performed. Detailed individuals are replaced by higher concepts. Attributes and relationships need to be assigned to new individuals. All modification applied to the model need to be documents to ensure traceable and trustworthy preservation of the business process.
Verify abstraction	**Input**	CMI with abstracted concept, original CMI, preservation requirements
	Output	Evaluation of the performed abstraction (yes/no)
	TIMBUS	Preservation Expert Suite: Model Weaver

	Architecture	
	Stakeholder involved	R,A: Preservation Manager
	Description	The abstraction of the model results in a loss of information. The verification against the preservation requirements ensures that all relevant information for preservation and redeployment are still available.
Decision	**Input**	Yes/No decision of *Verify Abstraction*
	Output	Trigger for *Perform Abstraction* or *Develop Preservation Plan* process
	TIMBUS Architecture	Preservation Expert Suite: Model Weaver
	Stakeholder involved	RA: Preservation Manager
	Description	The decision implements quality control and feedback loop. If the abstraction replaces information that is required in the future, the abstraction process is restarted.

3.4.2.4 Develop Preservation Plan

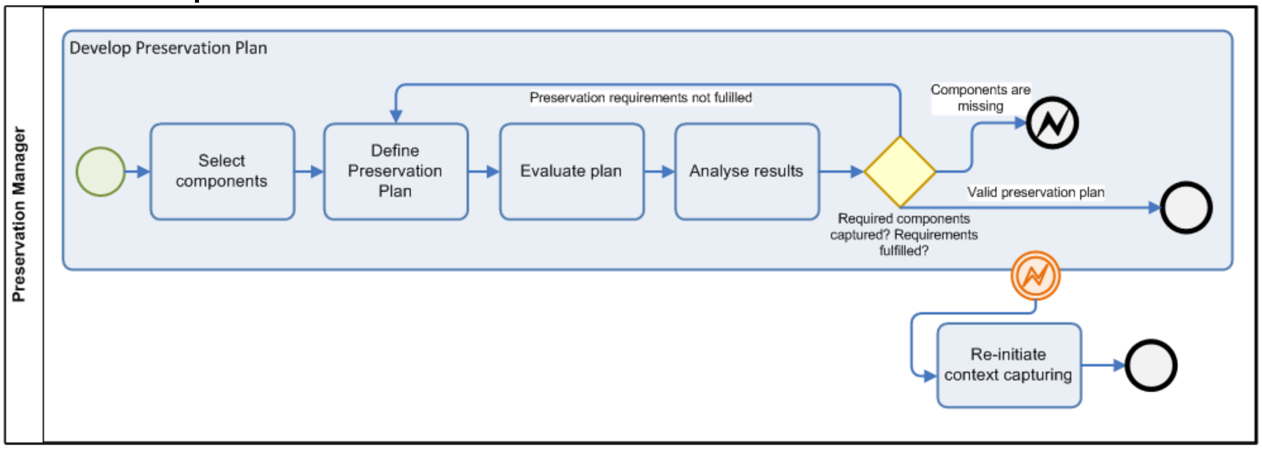

Preservation Manager
Develop Preservation Plan
Select components
Define Preservation Plan
Evaluate plan
Analyse results
Preservation requirements not fulilled
Components are missing
Valid preservation plan
Required components captured? Requirements fulfilled?
Re-initiate context capturing

Figure 11: Develop Preservation Plan

The *Develop Preservation Plan* process is responsible for identifying, testing and evaluating preservation strategies for the business process. The process needs to create a preservation solution that meets preservation requirements of the specific scenario. The preservation plan defines the capturing, preservation and redeployment of the process. Figure 11 shows the process steps for the development of a preservation plan. The steps are described in more detail as follows

In a first step the relevant components of the business process that need to be preserved are identified. A business process is a complex structure of services and activities. Not all components and functionality are required to be preserved for the future. The relevant components are selected and identified by using the context model.

A preservation plan for the selected components is created. Preservation actions to maintain the functionality, for example using emulation approaches, and to maintain the accessibility of digital objects (e.g. migration approaches) are identified. A preservation plan for the business process also includes access methods to redeploy the preserved process in a new environment. In order to ensure that a redeployed process works correctly and the restored information is authentic, validation and verification mechanisms are developed.

The preservation plan is evaluated against the previously defined preservation requirements of the setting. Multiple preservation plans, implementing different strategies for the preservation, can be developed and then be compared against each other. The process shown in Figure 11 is explained in the following sections in more detail.

3.4.2.4.1 Select components

The first step of the development of a preservation plan for a business process (shown in Figure 11) is the selection of the components to be preserved. All assets of a process (services, data, documents, etc.) that are significant for the process and are required to re-execute the process in the future need to be identified. The captured context model describes the implementation of the business process, but not all aspects of the implementation are relevant for preservation. Driven by the reuse scenario and the defined requirements, the significant artefacts can be identified. It might be possible exclude some parts of the implementation of the business process that are required for current operations and their performance, but not for the actual business logic. Examples for such parts are redundant components such as hot standby or load balancing server. Implementations of business processes are often integrated in larger information systems, where they are connected to other processes exchanging data for storing, reporting or processing information. In order to preserve a process, the boundaries of that process need to be clearly defined. Interactions with other systems and processes that need to be maintained for future reuse scenarios need to be specified. All services that implement required information and functionality of the process need to be part of the preservation solution and selected in this step. Interfaces and services for other systems that are out of the scope of the current business process can also be omitted from the preservation process.

Two approaches can be used to mark components in the context model: selection and exclusion. Components that are required to fulfil the preservation requirements are manually marked in the context model by human actors. The manual selection is done on an upper layer in the context model such as workflows, services and data sources. The low level implementation and dependencies of selected high-level concepts are addressed in the next step of the TIMBUS process.

Components can also be marked for exclusion from the preservation if they are part of the current implementation but not required in the future. The exclusion of such components reduces the effort for the preservation solution. In order to manually select and exclude components for preservation a deep understanding of the business process is required.

The outcome of this step is the context model instance of the business process with components that need to be preserved and components that can be excluded from the preservation marked within the model.

3.4.2.4.2 Define Preservation Plan

The definition of the preservation plan is a crucial step in the preservation. In this step plans for the preservation are identified to fulfil the preservation requirements. A preservation plan consists of:

- Actions to acquire the process data from the source system
- Preservation strategies to maintain the significant properties over time (e.g. emulation or migration for out-dated components and components that are at risk of becoming obsolete)
- Migration of the process into an archival storage format
- Access procedures for redeployment of the process in the future
- Validation and Verification procedure for the redeployed process

Different technical approaches can be defined and evaluated in this step. A comparison of different plans helps to identify the most suitable approach for a specific setting. A business process is an orchestration of tasks that are executed in a particular sequence to achieve a specific and well-defined goal. The process can be complex, involving various different services. For this reason, a combination of different preservation actions is applied to preserve a process for the long term. Examples are virtualisation and emulation approaches for preserving functionality of services, and migration for documents (such as guidelines, contracts, etc.). The overall goal is to maintain the significant properties of the process over time. Approaches and results from other preservation projects can be a vital source that helps to identify potential approaches for parts of the business process.

A challenging task for the preservation of complex processes is the preservation of relationships and dependencies between components over time. Knowledge of the dependencies is important for maintaining the functionality of the components. Broken dependencies that are not maintained over time can prevent the redeployment of the process in the future. Examples are manifold, such as missing libraries for software execution, missing databases for data input, incompatible hardware for operating systems or missing credentials for encrypted data. The dependencies need to be considered whenever changes are applied to components. Modification on components for preservation purposes for example can have undesired side effects on other components. Examples are the migration of data into other formats that cannot be processed further by other software components or replacement of software components by new versions or alternatives that offer different interfaces for interaction. For work on analysing and resolving software dependencies we refer to [41] [42] [43].

The preservation of external services that are not operated by the host institution is a challenge. While external services, such as data, storage and computation, offer several advantages for the operation of the process, the limited access to their implementation creates a lot of problems for the long term preservation. The availability of external services cannot be ensured as the operation service can be stopped by the provider, replaced by another service, or the provider can go completely out of business at any time. Preservation approaches for external services can be the migration to alternative services, the emulation of the service or the arrangement of an escrow agreement. The different strategies are discussed in detail below.

Preservation strategies and supporting strategies
Different strategies are used to maintain the significant properties of the process over time. Strategies can be combined for different components of the process.

- **Metadata**
  In order to maintain the usability, interpretability, accessibility and understandability of the process, additional metadata of its components are required. Understandability involves providing sufficient information so the component can be interpreted and understood in the future. Usability refers to the information that is required to translate the bit stream into a form that can be used by human users, or processed by computers. The OAIS [3] refers to understandability as Representation Information that provides additional meaning to a digital object. The TIMBUS context model and especially domain-specific models define many metadata for the process and its component. A lot of work has been done on preservation metadata and metadata standards in PLANETS[14] and CASPAR[15]. PREMIS provides a standard for preservation metadata [44]. Domain-specific research on metadata has been done by KEEP[16], ARCOMEM[17] and PrestoPrime[18] supporting emulators, social web and audio-visual content.

- **Migration**
  Migration can be seen as the copying or conversion of digital objects from one technology to another. The focus of migration is on the digital object rather than its environment. It is used for software as well as for hardware, e.g. migration of storage from one storage media to another media. The migration from one file format to another format implies a modification of the data. It needs to be guaranteed that no required information will be lost during migration and the significant properties of the object keeps are kept intact. Migration is a widely adopted strategy for documents stored in obsolete and out-dated formats that are used by human users.
  Besides the traditional migration approaches of format and storage media, the migration to alternative services or components can be a viable approach for business processes. Business Information systems are often optimised for performance and workload, characteristics that may not necessarily be relevant for archiving and future use. Examples are RAID storages, work balanced web services, etc. The use of single storage media or single web services can ease the preservation of the compo-

[14] http://www.planets-project.eu/

[15] http://www.casparpreserves.eu/

[16] http://www.keepproject.eu/

[17] http://www.arcomem.eu/

[18] http://www.prestoprime.org/

nents by maintaining the same functionality. In terms of licences, alternative resources can be suitable strategy to overcome legal conflicts. Examples are the use of open source software that allows for later modification of the application and access to the source code, instead of proprietary software. Important is that the alternative implements the same significant properties as the original service.
Another aspect of migration can rise from the use of external services. As the availability of external services cannot be assured, a potential strategy is to transfer such external system into the own system (in-housing). The strategy requires access to the implementation of the services and data of the service as well as the licences and right to operate the service. Examples are cloud storage that is operated by a third party. An equivalent service that is easier to preserve can be set up in-house.

- **Emulation**
  An emulator software mimics the behaviour and functionality of components, hardware or software. Emulation is a widely adopted strategy to preserve older computer platforms (e.g. video game console systems). It provides a layer between the host system and the originally used software. For hardware emulation the functionality of original hardware environment is provided by the emulation software. It translates between the new host system and the original software. Emulation can be used as long term preservation strategy for VMs (virtual machines) if the required hardware that is directly used by the VM becomes obsolete, or for specific hardware components that cannot be preserved for the future.

- **Virtualisation**
  Virtualisation (most common hardware virtualisation) has become a common business practice for server management. Virtualisation software provides a separation layer between the application services and the underlying hardware resources. It allows for consolidating server instances by using multiple virtualised servers that might share the same physical environment. By hiding the underlying hardware environment, the virtual machines have become portable and can be easy relocated on other hardware environments. Virtualisation has become widely adopted in IT with numerous solution providers for hardware and software. Examples for virtualisation software are Oracle VirtualBox[19] and VMWare[20].

  The separation from actual hardware provides an abstraction layer of the physical environment, such as network, storage and display. It increases the robustness of the VM against changes of the underlying hardware. The virtualisation is a practical approach to capture complex systems to maintain the dependencies within the VM. A VM uses some of the host system hardware directly. A part of the commands of the virtualised machine is executed directly on the hardware, e.g. CPU instructions. Other commands are executed by the VM software. The long term use of virtualisation is lim-

[19] https://www.virtualbox.org/
[20] http://www.vmware.com/

ited as it works only if the used physical environment (e.g. CPU) is compatible to the VM software. As soon as the hardware gets obsolete, hardware emulation is required to run the VMs. Different virtualisation approaches can be applied to existing business processes that are not implemented in a VM yet.

- Clone: a virtual image of an existing system that corresponds to a 1:1 mapping between the physical original system and the created VM. Small modifications are done for driver support of the VM layer. Tool support for such a conversion is provided for most operating systems. The clone contains all software components that are on the physical machine. In many cases a machine is not a dedicated host for the software of a single business process, but hosts various software services, which would be all captured in the VM. The additional software that is not required by the process can cause problems in the future. For example security problems by unauthorised access to the system via external services that are no longer maintained or concurrency problems arising from the locking of resources. Unused software services should thus be uninstalled from the captured VM for preservation.
- Build: this approach sets up a new VM with all required components for the business process. The required software components, data and configuration of the business process are extracted and identified from the source system. A new VM is set up by installing the software components with the collected configuration and the stored data. The build approach can be useful for systems that host different services that are not required for the business process to be preserved. The result of this approach is a dedicated VM that contains only the required components for the business process.

- **Re-build of SW Systems**

  Re-build stores the required software system in an archive. The idea is to identify the installed software packages, configuration and data of the source system. The installation sources of the software packages including the configuration and data are stored for archiving. At redeployment time, the system is rebuilt from these archived sources. This approach is similar to the building of virtualised instance, but instead of building a ready virtual machine image, the sources of the software services are archived.

  The re-build of a system can be automated with tool support for example for Linux[21] [22]. The identification of installed software as well as the instantiation of a new installation can be automated. In WP6 of the TIMBUS project, we are developing tool support for Linux systems to provide automated re-build.

- **Mock-up of SW Services**

  A potential solution for software services provided by third parties (e.g. Web Services) can be a mock-up of the services in the form of a simulation of the original service. The basic principle is to in-

[21] http://wiki.debian.org/VMBuilder/

[22] https://launchpad.net/vmbuilder/

tercept and record messages from the original system, which the simulation can then use to respond to request that have been captured previously. The approach is limited, as it can only be used for deterministic services (i.e. services for which the request and response pair always match, and which themselves are not dependent on any external state), and the mock-up can only respond to messages that have been recorded in the original system. For simple external services and for the preservation of particular instances of a process the mock up can provide a suitable solution if no other possibilities are given. An analysis of mock-up strategies for web services, and recommendations to make web services more resilient in general, can be found in [45].

- **Software Escrow**
  Business processes are often executed by using proprietary and customised software application and services. The software is developed and adapted for the individual needs of the customer by an external software vendor. The software is in many cases delivered as closed source to the customer that means the source code remains at the vendor and only the binaries of the software are delivered to the customer. From the preservation perspective, this scenario limits the potential preservation strategies for the software, as the software cannot be adapted to changes in the execution environment in the future. This implies that the execution environment itself needs to be maintained for the future, which can lead to considerable efforts. One option to improve the range of preservation strategies for proprietary software is Software Escrow. In a Software Escrow agreement, the source code of the software is deposited at a third party and released to the customer in case of predefined events (e.g. when the vendor goes out of business, or does not want to further maintain the software). A detailed description of Software Escrow, including the legal aspects as well a framework supporting the validation of the deposit material, is described in Chapter 4. An escrow agreement allows that the software can be adapted to new requirements in the future.
  The escrow principles can also be applied for external services. When a service provider deposits the implementation of the services at an escrow agent, the user of the service can get access to the implementation once the service gets unavailable. This could enable the user to run the service at his site if the provider stops offering the service.

Security aspects

Business processes have implemented a number of security features for example to avoid unauthorised access to sensitive data to prevent fraud, manipulation or data theft. Information security features do not provide any functional per se–security is regarded as a non-functional property, but is not required for the execution of a business process. Nevertheless, in many cases it is a required feature to protect the data by implementing security arrangements such as file encryption, complex user authentication and authorisation, or firewalls. Naturally, many security features are in conflict with the goals of Digital Preservation. The former seeks to limit access, while the latter pursues the goal of enabling easy access. Security adds complexity to systems, wherever it is implemented, and thus increases the effort required to preserve a system. Successful preservation provides two options: either remove security features wherever possible, or make security features future proof. Both options are challenging tasks and require well-considered methods.

Removing security features completely is only an option if a business process can be executed completely within a controlled environment, i.e. in virtualised machines that runs on a secured and hermetically sealed host system. In many cases this solution will not be possible, either if legal regulations or policies require security implementations or if there are too many dependencies on distributed systems that are beyond the control of the process owner. For many businesses it is essential to avoid data leaks and security breaches at all times, especially with regards to reputation and public perception. The main goal for the second option, preserving security features of business processes, is thus to keep them secure after redeployment as well. Hence authentication mechanisms such as passwords and user credentials and the software need to be preserved in a reliable way. As users (i.e. employees) might change in the future, their user roles, capabilities, competencies, job descriptions and other data needs to be preserved along the technical infrastructure as well. This includes authorisation models, software and other required systems. If it is not explicitly stated within the authorisation or authentication descriptions already, the roles must be mapped to organisational units and competencies as well. It is necessary to be able to derive a precise definition of who is allowed to access which data and which infrastructure. The establishment of security mechanisms in the redeployment environment is part of the redeployment procedure.

Preserving the security of a business process always bears risk. Common examples are passwords that have been lost during the course of time. If an encryption method is still secure enough not to be broken within reasonable time and effort, the loss of passwords can be a serious problem that makes the recovery of the preserved encrypted artefacts impossible. The same is true for secure authentication systems that require special software to be available in all involved systems (e.g. proprietary solutions). Data leakage, manipulation and fraud, viruses and hacker attacks will also in the future threaten redeployed business processes. Hence a well-balanced level of security is essential for the preservation of business processes and their successful redeployment. Research on the security aspects of business processes for Digital Preservation is done in Task 4.7 of the TIMBUS project and will be presented in the upcoming deliverable *D4.7 Validation of DP'ed Business Processes & Verification of Redeployed Business Processes*.

Procedures

The result of this process step is a Process Preservation Plan which defines four different procedures: acquisition, preservation, redeployment and verification & validation procedure. A procedure defines trigger events, condition and responsibilities for a set of activities.

- **Acquisition procedure**
  In order to preserve the process in the archive, the components and the process need to be captured. The acquired data need to be consistent, complete and in a suitable format. Depending on the process scenario as defined in Section 3.3.3, different levels of information are captured from the source environment. The preservation strategies employed also define the required data to acquire. Examples of the data are the software components and configuration set-

tings data. The data need to be captured in a consistent state of the process. This means all data needs to be in a valid state, e.g. transactions are finished, and data is stored persistently. The redeployment of the data should lead into a defined and valid state of the process again, which can then be executed. The capturing can be for example a snapshot of a database or the virtualisation of a machine. The data needs to be captured in a suitable format that can be further processed at a later stage for preservation, archival storage and redeployment. The format depends on the preservation and redeployment strategy for the component.

- **Preservation procedure**
  The preservation procedure defines all actions applied to the business process that migrate the process into a format ready for archival storage and ensures the long term availability of the components. The preservation strategies are defined in the preservation procedure. Actions to treat components that are at risk of becoming obsolete or of not being available in the future (e.g. third party services) are also defined here. Quality-ensuring measurements for verifying the results of preservation actions need to be defined. Preservation metadata is assigned to the archived business process data.

- **Redeployment procedure**
  The redeployment procedure defines how the business process can be taken from the archive and how it can be executed again in a new preservation environment. It defines the steps for the redeployment including technical, organisational as well as legal aspects. The technical procedure describes the required services that need to be provided by the redeployment environment, the setup for the software services and the provision of the required data. The organisational procedure defines, for example roles that need to be assigned to actors and credentials of the redeployed services that need to be set (e.g. access rights to privileged data). The redeployment needs to be checked for legal correctness according to current regulations. This applies, for instance, to the redeployed data that are particularly protected by law (e.g. personal data, medical data), contracts with third parties (e.g. external web services) and usage terms for software.

- **Verification and Validation procedure**
  In order to ensure that the process is redeployed correctly, a verification and validation (V&V) procedure is required. It defines measurement points to check that the redeployed process shows the same significant properties as defined for the original process. The V&V check should cover all significant properties defined as part of the preservation requirements.
  Validation and verification data are collected in the source environment for this check. A framework for validation and verification of archived business process is currently under development and will be presented in the upcoming deliverable *D4.7 Validation of DP'ed Business Processes & Verification of Exhumed Business Processes*. The V&V plan specifies the validation and verification procedure for the preserved and redeployed business process.

Identify external dependencies

The preservation plan defines the external dependencies of the preserved business process. A preservation solution has different external dependencies on components that are not part of the archived process. Examples are technological dependencies such as the required computer architecture for redeployment, used file formats, operating systems, software and interfaces in the process. Organisational and legal dependencies are contracts, policies and regulations. The external dependences are either directly used in the preserved process or indirectly affect the process, e.g. operating system to run an emulator, viewer applications for file formats or CPU architecture to run software. In order to keep the preserved business process accessible and useable for the designated community, changes of the external dependencies need to be monitored. Monitoring is part of the risk management (described in Section 3.4.2.1). The monitoring process for archived business processes is described in Section 3.4.5. If the preserved business process depends on technology that is no longer available or at risk of becoming obsolete (such as a CPU, operating system, or network infrastructure), preservation actions need to be taken. Then the assessment of DP approaches process is triggered by the monitoring. The context model instance can be used to identify external dependencies by identifying components that are not included in the preserved business process. The identified interfaces between components of the preservation solution and the outside world in the context model helps to define the monitoring criteria for the archived process.

The definition of a preservation plan is a highly complex task, as different aspects need to be considered for a holistic preservation solution. The process to identify and create a plan strongly depends on the settings surrounding a business process. In this section we described the procedures of a preservation plan and key aspects that need to be considered for preservation. As a result, this step should produce plans that acquire, preserve, archive, and redeploy the business process. The plans define technical as well as organisational steps to acquire the required data from the source environment, apply preservation actions and prepare the process for archival storage. Procedures for later redeployment in new environment are defined as well as measurements to validate und verify the redeployed process behaviour.

3.4.2.4.3 Evaluate plan

The preservation plans defined in the previous step need to be evaluated against the preservation requirements, to assess whether the proposed procedures are complete and correct.
For a full evaluation of the plan all actions defined therein have to be executed and evaluated. Due to the complexity and required effort of the plan not all aspects can usually be tested to full extent. For this reason, samples and parts of the process are used for the evaluation. The subset for the evaluation needs to reflect the variance of technical components of the process, be it file formats or different software components that are involved in the process. In order to evaluate the correctness of the preservation process and of the redeployment process, the preservation requirements are analysed and measurement points for the requirements are defined. Comprehensive requirements such as conformance to standards, guidelines or laws can require the consulting of domain experts for the evaluation. In the next step, experiments are designed to test the requirements. All experiments need to be repeatable and documented. For testing the redeployment procedure, empty test systems or virtual machines can provide a redeployment environment.

In the evaluation step, the experiments are executed and the measurements are taken. An important aspect of the evaluation is cost assessment for the preservation process. Approaches for the cost calculation of preservation with respect to business processes are presented in [24].

3.4.2.4.4 Analyse results

In the next step the results of the evaluation are analysed. Hence the measurements taken in the redeployment environment are analysed and checked against the requirements. The results of different plans are compared with each other to identify the advantages and disadvantages of the available approaches. Risk measurements are also identified in this task. This is achieved by the risk management of the business process and the effects of the application of preservation plan on the identified risks are analysed.
The evaluation data which is gathered in this step serves as preparation for the decision making. Important evaluation criteria are the fulfilment of the preservation requirements and the costs of the plans.

3.4.2.4.5 Decision: Preservation plan complete

The decision determines whether the proposed preservation solutions fulfil the preservation requirements or if the preservation attempt failed. In the latter case, a refinement process for the re-acquisition of the context and a re-planning of the strategies can be triggered (as shown in Figure 11). The same is true if components of the process are missing or more detailed aspects for preservation are needed. In this case again the acquisition of the context is retriggered. New approaches can be tested by re-execution of *Define Preservation Plan*. Different preservation tools or refinement of procedures can improve the plans. When the preservation plan meets the preservation requirements the next step is the development of a preservation action plan as defined in Section 3.4.2.3.

The outcome of *Develop Preservation Plan* process is one or more plans for the preservation of the business process that complies with the preservation requirements of the setting.

Select components	**Input**	CMI, Preservation Requirements
	Output	CMI with selected component
	TIMBUS Architecture	Preservation Expert Suite: Preservation Alternative Assessment
	Stakeholder involved	R,A: Preservation Manager
	Description	The CMI describes the current implementation of the business process. In this step components of the business process are selected that need to be preserved in order to maintain the significant properties of the process. The selection can be supported by reasoning techniques that identify dependencies of components that need to be preserved. Other components of the implementation can be excluded for preservation as they are only

		needed for current execution but are not relevant for the future, e.g. reporting to other services, load balancing, and backup functionality (i.e. non-functional requirements).
Define Preservation plan	**Input**	Preservation requirements, RAR, CMI with selected components, Preservation Knowledge Base
	Output	Defined preservation plan for the business process
	TIMBUS Architecture	Preservation Expert Suite: Preservation Alternative Assessment
	Stakeholder involved	R,A: Preservation Manager
	Description	In this step preservation approaches are defined including procedures to acquire the process data from the source system, preservation strategies to maintain the significant properties over time (e.g. emulation or migration for out-dated components and components that are risk of becoming obsolete), migration of the process into an archival storage format, access procedures for redeployment of the process in the future and a procedure for the validation and verification of the redeployed process.
Evaluate plan	**Input**	CMI, defined preservation plans, source environment
	Output	Evaluation of the preservation approaches against the preservation requirements
	TIMBUS Architecture	Preservation Expert Suite: Preservation alternative Assessment, Preservation Planner, Preservation Log Gap Detector DP Engine: Preservation Executer, Execution Monitor
	Stakeholder involved	R,A: Preservation Manager
	Description	The aim of the process is to evaluate the preservation approach against the preservation requirements. Experiments are designed to evaluate different aspects of the preservation process. The results of the experiments are evaluated.
Analyse results	**Input**	Results of the evaluation step
	Output	Evaluated preservation alternative
	TIMBUS	Preservation Expert Suite: Validation and Feedback

	Architecture	
	Stakeholder involved	R,A: Preservation Manager
	Description	The results of different approaches are compared against each other. Advantages and disadvantages of the approaches are identified. The analysis prepares the collected data of the evaluation for decision making.
Decision: Plan fulfils the requirements ?	**Input**	Evaluated preservation plans
	Output	Decision whether the preservation approaches fulfil the requirements or if components are missing or requirements are not fulfilled
	TIMBUS Architecture	Preservation Expert Suite: Validation and Feedback
	Stakeholder involved	R,A: Preservation Manager
	Description	The decision implements an iterative refinement for preservation approaches. The refinement can include a re-execution of the context acquisition or a re-definition of the preservation approaches. If the preservation alternative fulfils the preservation requirements the *Define Preservation Action Plan* process is triggered.

3.4.3 Preserve

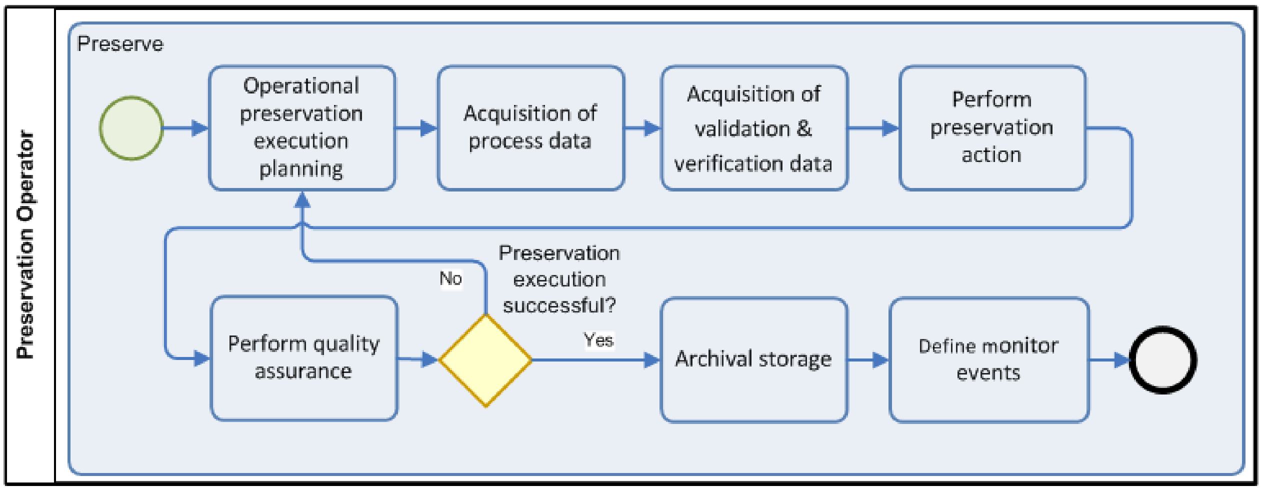

Preservation Operator
Preserve
Operational preservation execution planning
Acquisition of process data
Acquisition of validation & verification data
Perform preservation action
No
Preservation execution successful?
Yes
Perform quality assurance
Archival storage
Define monitor events

Figure 12: Preserve

The preservation phase is triggered by the risk management process to execute the planned actions for the preservation of the business process. The process steps are shown in Figure 12. The acquisition and preservation procedures of the preservation plan are applied to the business process. The software and data of the business process are captured from the source environment. Preservation actions are executed and the process is prepared for archival storage. Quality assurance mechanisms check the correct execution of the preservation phase.

3.4.3.1 Operational planning of preservation execution

In the first step of this phase, the preservation plan is initiated on the business process. Triggers and preconditions for the data acquisition and preservation procedure are defined for the current stetting. The responsibilities for each task are assigned to persons in the organisation. Quality Assurance (QA) mechanisms are defined and planned for each process step. The QA checks the output and input of each task against requirements and standards. It helps to monitor the execution of the preservation ensuring completeness and correct capturing and preservation of the business process.

3.4.3.2 Acquisition of process data

In this step all required data for the preservation of the business process are acquired from the source environment. The level of captured data depends on the process scenario as defined in Section 3.3.3. Software components, configuration data and other relevant data are acquired. The required data are specified by the preservation requirements and the planned preservation strategies. For the acquisition of data it is important to capture a valid state of the process for a correct re-execution in the future. Software components such as databases and workflow engines support the export of snapshots of the system state. Moreover-so-called “Physical-to-Virtual” (P2V) tools (e.g. VMware Converter[23]) support the creation of snapshots of physical machines.

[23] http://www.vmware.com/products/converter/overview.html

3.4.3.3 Acquisition of validation and verification data

In order to check the correct redeployment of the process in the future, validation and verification data need to be acquired from the source system as reference points. Examples for validation and verification data are log files of software components or defined behaviour tests consisting of input and expected output samples. The validation and verification plan created in the planning phase defines measurement points of the process and expected values. The expected values are captured form the source environment in this step. Logging tools can help to acquire the test data, as they can record the message exchange and the information flow between software components. If the components implement a deterministic behaviour, the data can be used to validate and verify the redeployment, because if the same data is processed in the new environment, the results should be identical. For non-deterministic software components, the test cases need to be defined manually.

3.4.3.4 Perform preservation action

After the data was captured from the source system the required preservation actions are applied. In many cases, several preservation actions have to be combined in order to achieve the desired results. These actions can include the execution of tools (e.g. migration tools) or the setup of a specific rendering environment (e.g. emulators). As the preservation actions often include modifications and changes to the data, the correct execution of all preservation actions need to be monitored. Undetected errors that occur during the execution of a preservation action can lead to a loss of information. A quality check of the result of the preservation actions is done in the next step.

After the execution of the preservation actions the process data are prepared for archival storage. It includes the assignment of archival metadata and the migration into a format that can be stored in the archive. The metadata help to organise, describe, index, structure, discover, find and manage the digital information. For the business process the description of the relations and dependencies of the process components is very important. The archival storage also includes the redeployment procedure.

3.4.3.5 Perform quality assurance

The completeness of the correct data, correctness of the preformed actions and the successful migration to an archival storage format need to be validated and verified before the process data is finally stored in the archive. The QA needs to ensure that the significant properties of the process are intact and that all required information and components for the redeployment are present. The quality check can be done on different levels. The output of single preservation actions can easily be verified (e.g. for format migration, the content of the resulting output file can be checked). For a complete validation of the preservation of the complete business process, the redeployment needs to be executed for testing purposes. An empty system (e.g. in the form of virtual machines) can be used as a test environment. The redeployment process is executed as described in Section 3.4.4. The validation and verification procedure can be used to check the significant properties of the redeployed process.

A negative result of the quality assurance leads to a re-start of the preservation phase, with refinements for the execution (as shown in Figure 12). If the preservation requirements are fulfilled, the preserved business process can be stored in the archive.

3.4.3.6 Archival storage

In this step the business process is stored in the long term archive. The correctness of the preservation has been validated and verified. The process can now be bundled and stored. The selected preservation strategies and supporting strategies as discussed in Section 3.4.2.4 define the stored data in the archive, e.g. VM images or software repositories for redeployment. A notification is submitted to the risk management about the progress to update the risk registry.

3.4.3.7 Define monitor events

Once a business process has been stored in the archive, the work is not done yet. In order to guarantee the long term capabilities of the archive, the preserved business process in the archive needs constant monitoring. This monitoring helps to ensure that the preserved business process is prepared for the demands of a changing environment. Two aspects need to be considered for the archived process: that the process is compliant with the organisation’s preservation objectives and the external dependencies of the preserved process are available. In this step monitor events of the preserved business process are defined that need to be collected and analysed. The monitoring of the events is part of the risk management. The general risk monitoring and review process is described in [24]. The specific monitoring process of archived business processes and the acquisition of relevant data for the monitoring of the business is described in Section 3.4.5 that feeds into the monitoring process described in [24].

External dependencies need to be actively monitored to ensure that the process can still be redeployed. Such dependencies are software and hardware requirements, as well as organisational settings or legal environments that are required to redeploy the process. The external dependencies are identified as part of the *Develop Preservation Plan* process as described in Section 3.4.2.4. In this step measurements and thresholds for the external dependencies are defined. They are submitted to the risk management for active monitoring.

Review cycles for the redeployment procedure and for the preservation and redeployment technology are defined and submitted to the risk management. The results of this process are fed into the monitoring and review process of the risk management described in Section 7.4.5 in [24]. The following table shows the specification of the activities in the preservation process.

Operational preservation execution planning	**Input**	Selected preservation strategy, PPP
	Output	Execution plan for preservation of the business process
	TIMBUS Architecture	Preservation Expert Suite: Preservation Planner
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Operational Manager, Technology Operator
	Description	In this step schedule is defined for the execution of preservation and responsible persons are assigned.
Acquisition of	**Input**	Source environment, PPP

process data	**Output**	Captured software, data and configuration from live system (e.g. current snapshot of data bases)
	TIMBUS Architecture	Agent Module: Metadata Capturer
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Operational Manager, Technology Operator
	Description	Capturing of required data from the source environment. Defined synchronisation points to capture a valid state of business process.
Acquisition of validation and verification data	**Input**	Source System, PPP
	Output	Validation and Verification data for the preserved business process
	TIMBUS Architecture	Agent Module: Metadata Capturer
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In order to validate the correctness of the preserved business process at exhumation stage, validation data are required for the validation procedure. The capturing of validation data is defined in the preservation plan.
Perform preservation action	**Input**	Captured data and PPP
	Output	Preserved business process (updated CMI, new data, software, configuration, documentation of performed preservation action)
	TIMBUS Architecture	DP Engine: Preservation Executer, Execution Monitor Preservation Expert Suite: Preservation Planner
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	Execution of preservation action including emulation, migration, virtualisation, etc. Assignment of additional metadata for preservation
Perform quality	**Input**	Preserved business process
	Output	Evaluation of QA

assurance	**TIMBUS Architecture**	Preservation Expert Suite: Preservation Log Gap Detector, Validation and Feedback
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In order to verify the captured business process and the executed preservation action the preserved business process needs to be tested against the preservation requirements. For that purpose the business process is redeployed on an empty test system to validate its behaviour.
Decision: Preservation successful	**Input**	Results of QA
	Output	Yes/No decision
	TIMBUS Architecture	Support for this decision is given by the Preservation Expert Suite.
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	Feedback loop for refinement of the preservation actions. If the preservation results do not meet the requirements, the preservation process can be restarted for refinement of the actions.
Archival storage	**Input**	Preserved BP
	Output	AIP (Archival information packages)
	TIMBUS Architecture	Preservation Repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	Migration of business process in storage ready format. Storage in archive repository.
Define monitor events	**Input**	PPP and CMI
	Output	Defined monitor parameters for re-evaluation of preserved business process
	TIMBUS Architecture	iERM: Risk Monitor

	Stakeholder involved	A: Preservation Manager R: Preservation Operator I: Process Owner, Risk Manager
	Description	Events are defined which require a re-evaluation of the preservation plan. Possible triggers for the events are e.g. periodically scheduled reviews, or changes in external dependencies. These changes could by of technological (external services, hardware, software) or organisational nature (target community, requirements, law).

3.4.4 Redeployment

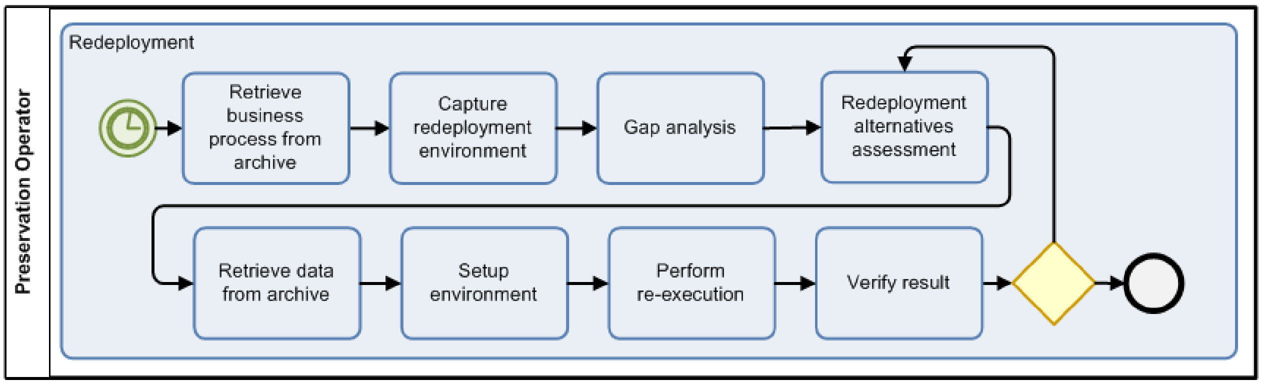

Preservation Operator
Redeployment
Retrieve business process from archive
Capture redeployment environment
Gap analysis
Redeployment alternatives assessment
Retrieve data from archive
Setup environment
Perform re-execution
Verify result

Figure 13: Redeployment

The redeployment phase defines the reactivation of a preserved business process in a new environment. Beside the technical redeployment of the process, organisational and legal aspects of the process need to be reactivated and adjusted to the organisational structures and legal conditions. The redeployed business process is then verified against the validation measurements of the original process. The process steps are shown in Figure 13.

3.4.4.1 Retrieve business process from archive

In the first step, the process description of the archived process is retrieved from the archive. The description includes the redeployment procedure and the inventory list of the archival package including artefacts and the relation with other components. The redeployment procedure describes the required infrastructure for the redeployment including technical as well as organisational aspects.

3.4.4.2 Capture redeployment environment

In this step, the characteristics of the redeployment environment are captured. This includes the suitability and compatibility of the new execution environment with the archived process. The available technical components, organisational and legal aspects that are relevant for the redeployment of the process are identified. The context model can be used to describe the context of the redeployment environment. Additional tool support can be used to acquire the technical aspects of given environment.

3.4.4.3 Gap analysis

The gap analysis compares the required environment for the redeployment of the business process and the actual redeployment environment. It identifies matches of components as well as elements missing for the redeployment procedure. In the next step the identified gaps are addressed.

3.4.4.4 Assessment of redeployment alternatives

For the redeployment, the technical infrastructure needs to be adjusted and prepared providing hardware, interfaces and software services for the process. The redeployment procedure is adjusted and extended for the given environment. Gaps that were identified in the previous step need to be resolved. Different approaches can be used to overcome the gaps, e.g. the use of different tools to emulate components, or migration of data formats. Information security features are identified that can be used for the redeployed process to avoiding unauthorised access to sensitive data and services. Security mechanisms that were implemented in the archived business process might have to be replaced by current techniques to be conforming to current standards, as the original (and thus potentially outdated) security feature implementations bear the risk of vulnerabilities. Redeployment strategies are tested and evaluated from technical as well as financial point of view. Beside the technical redeployment, organisational and legal discrepancies need to be resolved. The outcome of this step is a set of actions that prepares the current environment for the redeployment of the business process as defined in the redeployment plan.

3.4.4.5 Retrieve data from archive

In this step, all data (e.g. software components, data) of the business process are retrieved from the archive.

3.4.4.6 Setup environment

In this step, the environment is prepared for the redeployment. The required software is installed in the target environment, the configurations are set and the required data is imported. Tools and components for validation and verification are installed as defined in the Verification & Validation Plan. Original user roles and access rights have to be adjusted in order to comply with the current environment. In order to protect sensitive data and services of the redeployed process, new or updated security mechanisms have to be installed.

3.4.4.7 Perform re-execution

In this step the redeployed process is executed in the new environment. In order to validate the correct redeployment of the process the test procedures of the Verification & Validation Plan are executed. During this process all significant properties of redeployed business process are gathered.

3.4.4.8 Verify results

Finally, the measurements taken in the previous step are verified. The Verification & Validation Plan defines acceptable ranges for the measurements for the evaluation criteria. As shown in Figure 13, if the redeployed process fails to fulfil certain criteria, the *Assessment of Redeployment Alternatives* (as described in Section 3.4.4.4) can be re-initiated to adjust the redeployment (e.g. use of other tools, hardware, etc.).When the results of the verification are positive, the redeployed business process can be used productively. The following table provides a specification of the above-discussed process steps.

Retrieve business process from archive	**Input**	Archive
	Output	Retrieved business process description
	TIMBUS Architecture	Preservation Repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Risk Manager
	Description	In the first step of the redeployment the description of the preserved business process is retrieved from the archive.
Capture redeployment environment	**Input**	Redeployment environment
	Output	CMI of the redeployment environment
	TIMBUS Architecture	Acquisition Module: Properties Extractor
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Legal expert, regulator
	Description	Collects all information about the redeployment environment, e.g. IT-infrastructure, organisational and legal aspects.
Gap analysis	**Input**	Retrieved business process from the archive and CMI of the redeployment environment
	Output	Gap report
	TIMBUS Architecture	Preservation Expert Suite: Preservation Log Gap Detector
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	The description of the business process contains the infrastructure that it requires for redeployment. The extracted model of the redeployment environment describes the available infrastructure. The Property Gap Detector identifies matches of the components and missing artefacts for the redeployment. The gap analysis is not limited technical aspects organisational as well as legal and contractual aspects need to be analysed.

Redeployment alternatives assessment	**Input**	Gap report
	Output	Redeployment plan
	TIMBUS Architecture	Redeployment Planner, Verification and feedback component
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	This step is responsible for assessing and selecting the best redeployment strategy for the setting. The redeployment procedure defines all the required actions needed for redeploying the business process in the redeployment environment. Based on the gap analysis the plan needs to be adjusted for the current environment. The result is a list of actions, describing how to setup of the environment and guiding the installation of additional software services (e.g. emulators) in order to redeploy the process. The redeployment plan also addresses organisational aspects, for example the roles used in the preserved business process that need to be set up, as well as the assignment of roles with their specified required on competences and privileges.
Retrieve required data from archive	**Input**	Redeployment plan
	Output	Retrieved business process data including software, configuration, and working data (such as database tables with business date)
	TIMBUS Architecture	Preservation repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	After the redeployment planning the data, software and configuration of the business process are retrieved from the archive.
Setup environment	**Input**	Redeployment plan
	Output	Redeployment environment
	TIMBUS Architecture	DP Engine: Redeployment Executor

	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In this step the environment is instantiated for the redeployment, according to the redeployment plan. The required software is installed, the configurations are set and the data are imported. Tools and components for validation and verification are installed as defined in the Verification & Validation Plan. Examples are the deployment of software, databases, virtualised machines, and the recovery of data.
Perform re-execution	**Input**	Instantiated environment
	Output	Redeployed business process and monitoring log
	TIMBUS Architecture	Redeployment executor, execution monitor
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In this step the redeployed business process is executed. In order to verify the correct redeployment of the process, the test procedure defined in the Verification & Validation Plan is executed and measurements are taken.
Verify result	**Input**	Monitoring measurements from the re-execution step
	Output	Validation result of the redeployed business process
	TIMBUS Architecture	Preservation Log Gap Detector, Verification and feedback component
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In the last step the execution of the business process is validated. Data captured from the original process are compared against the data extracted from the redeployed process. Deviations in the execution are detected and analysed.
Decision about valid redeployment	**Input**	Validation of the redeployed BP
	Output	Yes/No
	TIMBUS	Verification and feedback

	Architecture	
	Stakeholder involved	A: Preservation Manager R: Preservation Operator I: Process Owner
	Description	If the redeployed business process does not fulfil the Verification & Validation requirements, identified inconsistencies are reported back to Redeployment Alternatives Assessment step for the adjustment of the redeployment process (shown in Figure 13). If the validation result of the redeployment is positive, the process owner is informed that the business process can be used productively.

3.4.5 Monitoring and Review

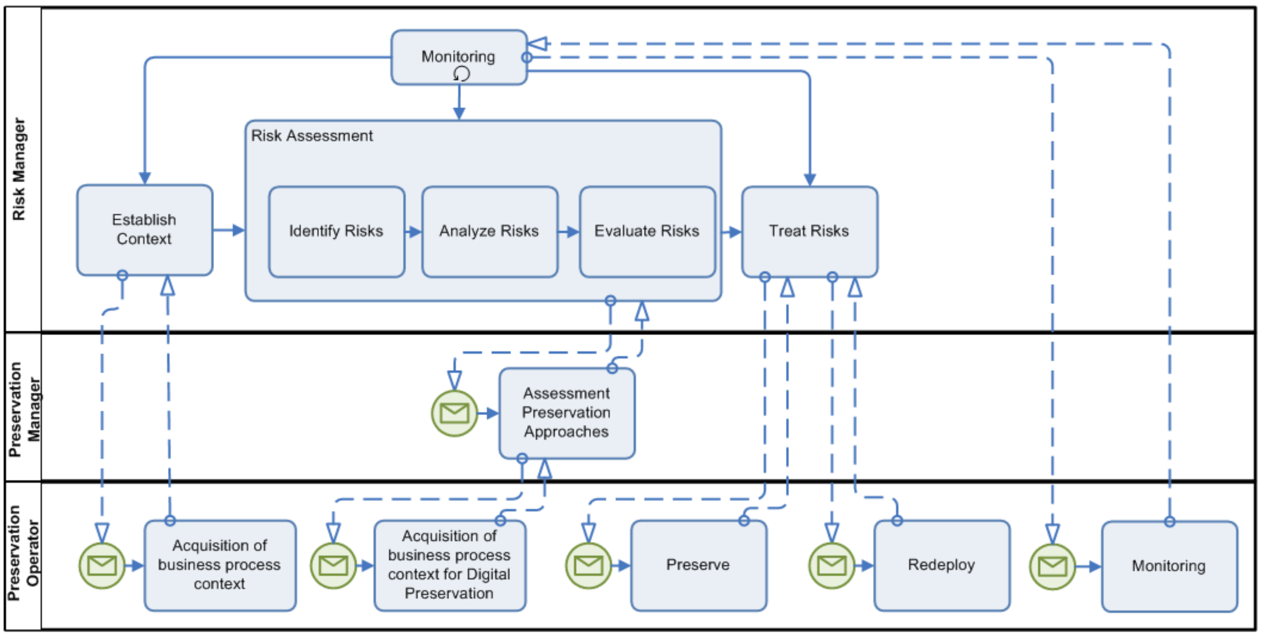


Figure 14: TIMBUS Process including Monitoring

Monitoring and reviewing ensures that the archived business process is adaptable to the requirements of a changing environment over time. Two types of monitoring are relevant for the preservation of business processes: that the process is compliant with the legislation and organisation's preservation objectives, and that the dependencies and requirement of the preserved process are valid. The monitoring from the risk management perspective is described in Section 7.4.5 of [24]. The monitoring process in this document describes the data gathering for the risk monitoring and part of the decision taking that overlaps with the monitoring process described in [24]. The gathered data are fed into the monitoring process of the risk management as shown in Figure 14. The responsibilities of roles for monitoring are defined in Table 6.

Table 6: RACI for Monitoring of Preserved BP

	Monitoring of Preserved BP
Process Owner	
Process Operator	
Technology Manager	
Technology Operator	
Preservation Manager	A
Preservation Operator	R
Executive Manager	
Operational Manager	
Risk Manager	I
Legal Expert	
Auditor	
Regulator	

We identified two types of monitoring of the preserved business process. The first type is the monitoring of the context and requirements of the source environment as described in Section 3.4.5.1. The archived business process needs to be checked against changes of preservation's requirements in the institution.

The second type, described in Section 3.4.5.2, monitors the requirements of the preserved process for a redeployment environment and the implementation of the archived business process. The archived business process requires specific infrastructure to be redeployed, thus this needs to be monitored to ensure the availability of the process.

3.4.5.1 Monitoring of business process context in source environment

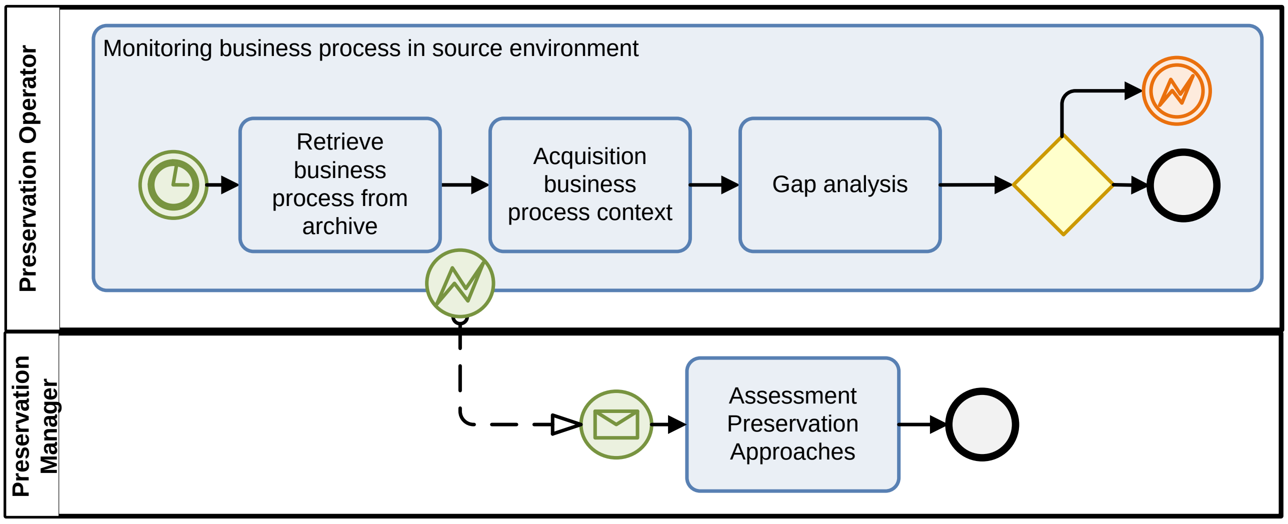

Preservation Operator
Monitoring business process in source environment
Retrieve business process from archive
Acquisition business process context
Gap analysis
Preservation Manager
Assessment Preservation Approaches

Figure 15: Monitoring of business process context in source environment

The context of a process can change over time. The context specifies amongst others the motivation, requirements, goals and obligation for the preservation. The context used to plan and implement the preservation solution for the business process needs to be monitored against changes in the preservation requirements. The preserved business processes needs to be verified against changed or new requirements on a regular basis. In case of non-conformance with the requirements, adaptations and modification of the archived business process needs to be performed. The process to monitor the requirements in the source environment is shown in Figure 15. Changes in the context have different causes, e.g. legal obligations or new re-use scenarios for the business process. Also, in case that the process is still running, new implementation and improvement of the process in the source environment can require updates of the archived business process.

The process shown in Figure 15 starts with the retrieval of the description of the archived process from the archive. The description contains the documentation of the preserved process and artefacts created during the planning phase such as preservation requirements, context model.

In the second step, the current context from the source environment is captured including the current requirements, obligations and motivation for the preservation of the process. If the archived business process is still running in the environment the process is captured again and can be compared with the archived process.

In the last step, a gap analysis is performed comparing the context of archived process and the current context in the source environment. Identified discrepancies are evaluated. If the preserved process does not meet the current requirements a revision of the preservation solution needs to be triggered (as show in Figure 15).

Retrieve business process from archive	**Input**	Process identifier
	Output	Retrieved business process model
	TIMBUS Architecture	Preservation Repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Risk Manager
	Description	In the first step of the monitoring of business process in the source environment the description of the preserved business process is retrieved from the archive.
Acquisition business process context	**Input**	Source environment
	Output	CMI of the business process in source environment
	TIMBUS	DP Agent Module, DP Acquisition Module, Legal Life-Cycle Module, Preservation Exert Suite

	Architecture	
	Stakeholder involved	R: Preservation Operator A: Preservation Manager C: all other stakeholder I: Risk Manager
	Description	The current context of the preserved business process is captured and documented including current preservation requirements, obligations. If the archived process is still running in the source environment the process itself is captured as well.
Gap analysis	**Input**	Retrieved business process description from the archive and CMI of the source environment
	Output	Gap report
	TIMBUS Architecture	Preservation Expert Suite: Preservation Gap Detector
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	The context of the preserved process is compared to the current context. Discrepancies are identified and assessed. If the archived process does not meet the current requirements the *Assessment of Preservation Approaches* is triggered as show in Figure 15 to adjust the preservation solution.

3.4.5.2 Monitoring of preserved business process

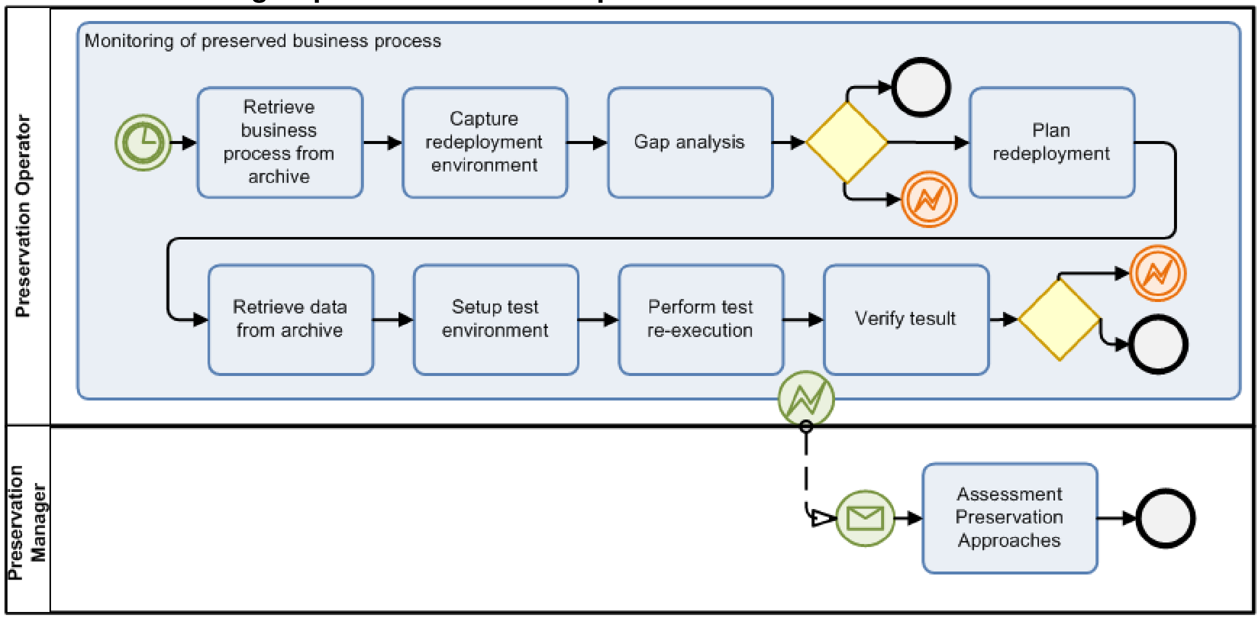

Preservation Operator
Monitoring of preserved business process
Retrieve business process from archive
Capture redeployment environment
Gap analysis
Plan redeployment
Retrieve data from archive
Setup test environment
Perform test re-execution
Verify tesult
Preservation Manager
Assessment Preservation Approaches

Figure 16: Monitoring of preserved business process

Actively monitoring the dependencies and requirements of a preserved process is required to ensure the possibility for redeploying the process on current infrastructure. The dependencies include infrastructure and components that are required for redeployment, including for example, specific hardware, software or operating systems. Besides the monitoring of the dependencies, the dry-run execution of redeployment procedure on a regular basis is recommended. Frequent tests of the redeployment procedure help to identify practical problems and potential discrepancies of the preserved process.

The used technology for preservation and redeployment should be reviewed from time to time. New versions of the used tools and technology can improve the preservation solution of the business process.

The process for monitoring and review of preserved business process is shown in Figure 16. The process steps are similar to the redeployment process described in Section 3.4.4. For the review and test of the redeployment, the process is not re-instantiated to full extent. Parts of the process and sample data are selected for the test. The process starts with the retrieval of the description of the archived business process. The characteristics and available components of the redeployment environment are captured and documented in the context model. A gap analysis of the required environment for the archived process and the current available environment identifies deviations. Examples for gaps are technical environments that were required to run archived software, or required skills of personal to run the process, which have now changed or are no longer available. The findings of the gap analysis are assessed. In case of significant deviations the preservation of the process needs to be adjusted. Thus, the planning phase is triggered as shown in Figure 16. The procedure can also be used to review the technologies used in the preserved archive. Update of technology used and tools of the preserved business process can improve the compatibility to current infrastructure and prevent technological obsolescence. The first three steps of the monitoring process as shown in Figure 16 should be done on a regular basis to review the external dependencies and the technology used by the preserved business process.

By doing a test-wise redeployment of the business process practical problems and discrepancies of the process and the redeployment procedure can be detected. The results of the monitoring and review process are reported to the risk management as shown in Figure 14. The table below summarizes the process steps need to monitor the preserved process.

Retrieve business process from archive	**Input**	Process identifier
	Output	Retrieved business process model
	TIMBUS Architecture	Preservation Repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator C: Risk Manager
	Description	The description of the preserved business process is retrieved from the archive.
Capture redeployment environment	**Input**	Redeployment environment
	Output	CMI of the redeployment environment
	TIMBUS Architecture	Agent Module
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	Collects all information about the current redeployment environment.
Gap analysis	**Input**	Retrieved business process from the archive and CMI of the redeployment environment
	Output	Gap report
	TIMBUS Architecture	Preservation Expert Suite: Preservation Log Gap Detector
	Stakeholder involved	A: Preservation Manager R: Preservation Operator

	Description	The description of the process specifies the infrastructure required to redeploy the process. The extracted model of the redeployment environment describes the available infrastructure. The Property Gap Detector identifies matches of the components and missing artefacts for the redeployment. If substantial discrepancies are identified the *Assessment Preservation Approach* process is triggered to update the archived BP. Within the gap analysis the used technologies and tools of the archived process can be reviewed. Updates and use of new technologies can counter technological obsolescence of the archived process implementation. The first three steps of the process can be used for regular review of the external dependencies and used technologies. The results of the analysis are reported to the risk management. The complete monitoring process describes the test wise redeployment of the process.
Plan redeployment	**Input**	Gap report
	Output	Redeployment plan
	TIMBUS Architecture	Preservation Expert Suite: Redeployment Planner, Verification and Feedback
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	A full review of the archived process includes the redeployment in a test environment. The redeployment planner is responsible to assess and select the best redeployment strategy for the setting. The redeployment plan describes the match between the preserved business process and the current environments. It generates a list of action that describes the setup of the environment and the instantiation of the process.
Retrieve required data from archive	**Input**	Redeployment plan
	Output	Retrieved business process data
	TIMBUS Architecture	Preservation repository
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	The process data, software and configuration are retrieved from the archive.

Setup test environment	**Input**	Retrieved business process and redeployment plan
	Output	Instantiated test environment
	TIMBUS Architecture	DP Engine: Redeployment executor
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In this step the environment is instantiated according to the redeployment plan. The required software is installed, the configurations are set and the data imported. Tools and components for validation and verification are installed as defined in the Validation & Verification Plan. Examples are deployment of software, databases, virtualised machines, and installation of data.
Perform test re-execution	**Input**	Prepared test environment
	Output	Redeployed process and monitoring log
	TIMBUS Architecture	Redeployment executor, execution monitor
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In this step the redeployed process is test-wise executed. In order to verify the correct redeployment of the process, the test procedure defined in the Verification & Validation Plan is executed and measurements are taken.
Verify result	**Input**	Monitoring log of original and redeployed process
	Output	Validation of the redeployed business process
	TIMBUS Architecture	Preservation Expert Suite: Preservation Log Gap Detector, Verification and feedback
	Stakeholder involved	A: Preservation Manager R: Preservation Operator
	Description	In a last step the execution of the business process is validated. Data captured from the original process are compared against the data extracted from the redeployed process. Deviations in the execution are detected and

		analysed.
Decision about valid test redeployment	**Input**	Results of the verification of the redeployed business process
	Output	Yes/No (list of significant discrepancies in case of no)
	TIMBUS Architecture	Preservation Expert Suite: Verification and Feedback
	Stakeholder involved	A: Preservation Manager R: Preservation Operator I: Process Owner
	Description	Identified inconsistencies and problems are reported to the Assessment Preservation Approach process to update the archived business process for the current redeployment environment (as shown in Figure 16).

3.5 Architectural view of the TIMBUS process

This section presents the architectural view of the TIMBUS processes, specifying the implementation and support for the processes by the TIMBUS architecture. The TIMBUS architecture, presented in [6], consists of six modules providing services to support the preservation process. The design of the architecture was driven by the requirements of the use cases. ArchiMate [8] is used to present the architectural view of the TIMBUS process. It allows modelling the different layers of enterprise architecture including business, application and technology. The concept of viewpoints [8] can be applied to the models in order to highlight different aspects and concerns of the model (e.g. different level of detail or components). A short introduction into the concepts of ArchiMate is given in [22]. Figure 17 shows the layered overview of the TIMBUS process including business and application layer of the enterprise architecture. It shows the business services that are provided and used by the three actors (Risk Manager, Preservation Manager and Preservation Operator). A detailed overview about stakeholders' involvement and responsibilities of the TIMBUS process is presented in Section 3.4.1. The five business services are realised by the five core business processes of TIMBUS (*Risk Management, Acquisition of business process context, Assessment of Preservation Approaches, Preserve and Redeploy*). Figure 17 further shows the use of the six TIMBUS application services (defined in the TIMBUS architecture) by the business processes. A detailed view of the architectural support for the processes is shown in the following sections.

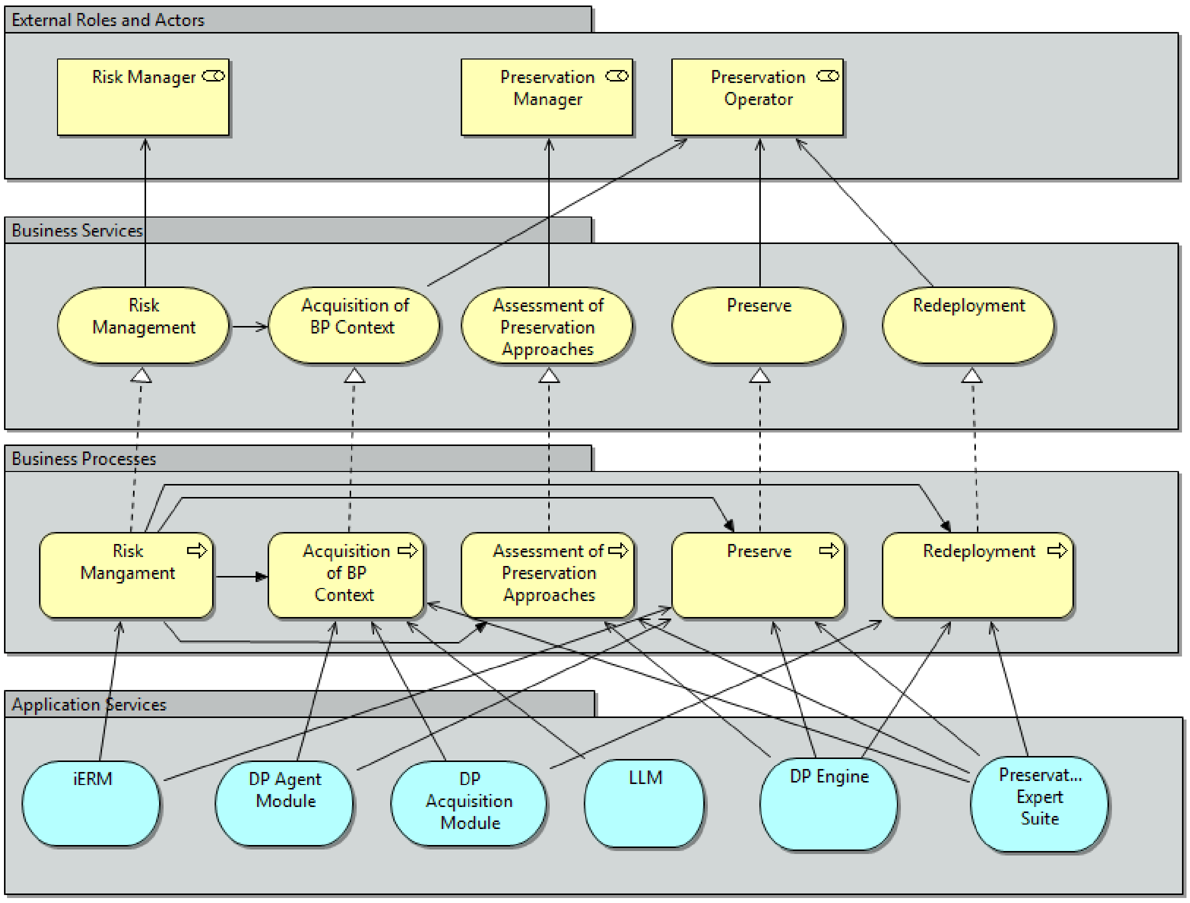


Figure 17: Architecutral Layered View of the TIMBUS Process

3.5.1 Risk Management

The *Risk Management* process is described in Section 3.4.2.1 and in more detail in [24]. Figure 18 shows the process steps of the *Risk Management* process. The supporting components of the Intelligent Enterprise Risk Management Module (iERM) are shown in Figure 18.

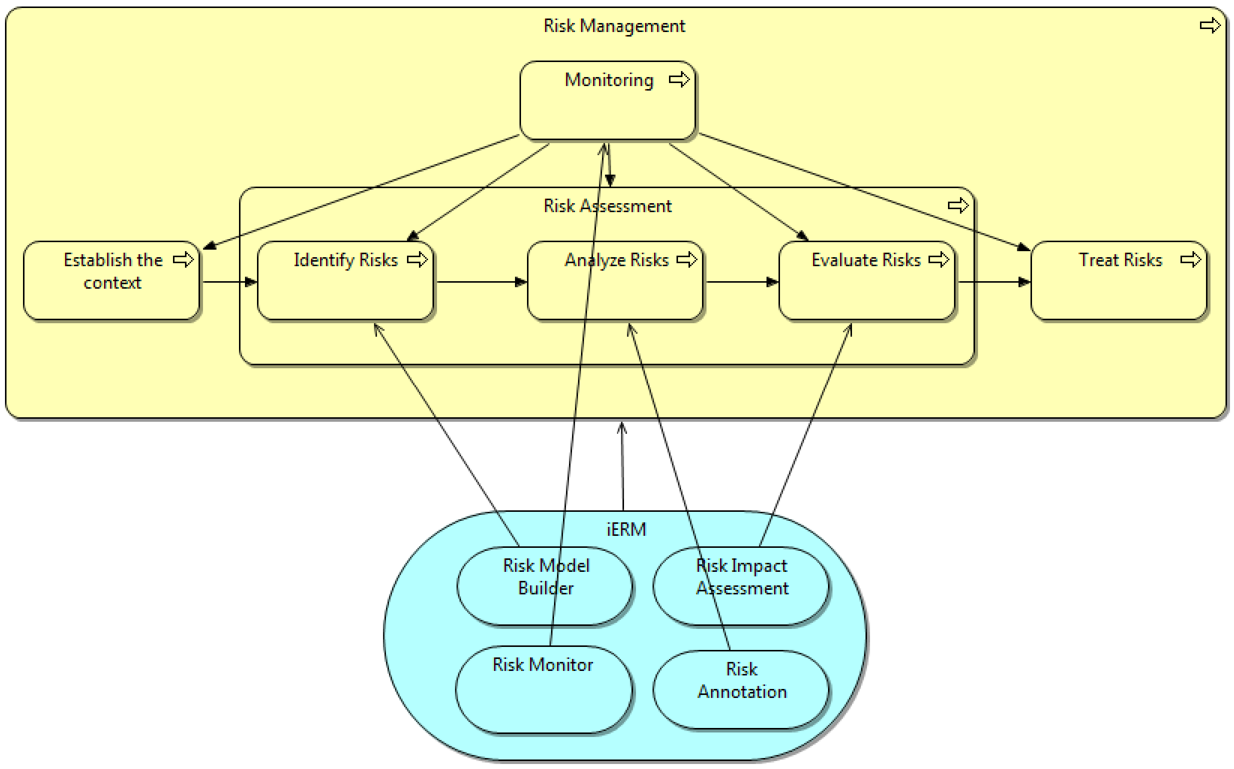


Figure 18: Risk Management - Application Usage View

3.5.2 Acquisition of business process context

The *Acquisition of the business process context* defined in Section 3.4.2.2 uses the services of four architectural modules (DB Agent, Acquisition, Legal Life-Cycle Module and Preservation Expert Suite). Figure 19 shows the use of the process steps.

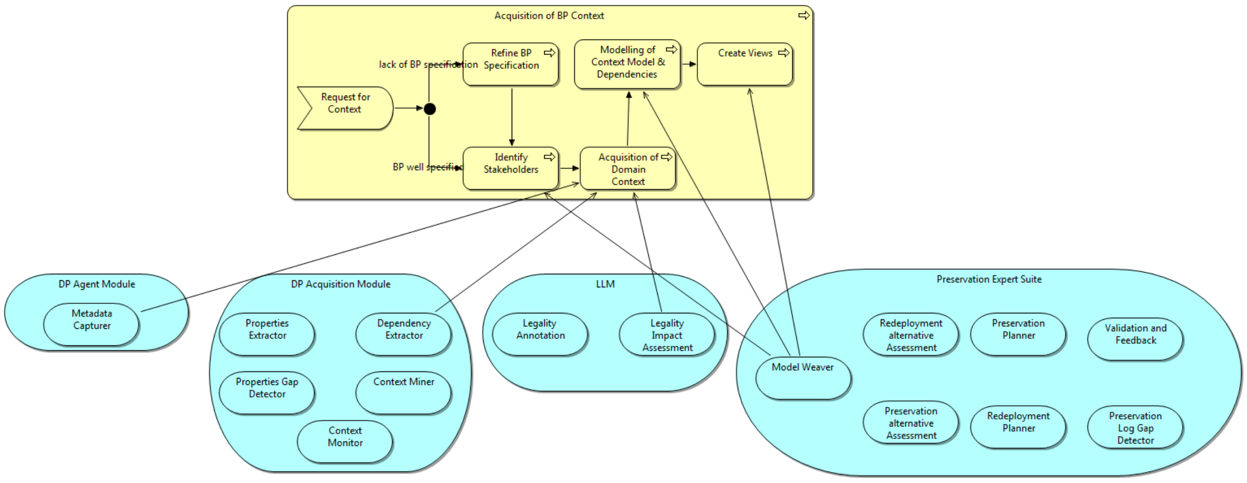


Figure 19: Acquisition of business process Context Application Usage View

3.5.3 Assessment of Preservation Approaches

The *Assessment of Preservation Approaches* process is mainly supported by the Preservation Expert Suite as shown in Figure 20. The DP Engine is required to test the approach for the evaluation of the preservation plan as specified in Section 3.4.2.4.3.

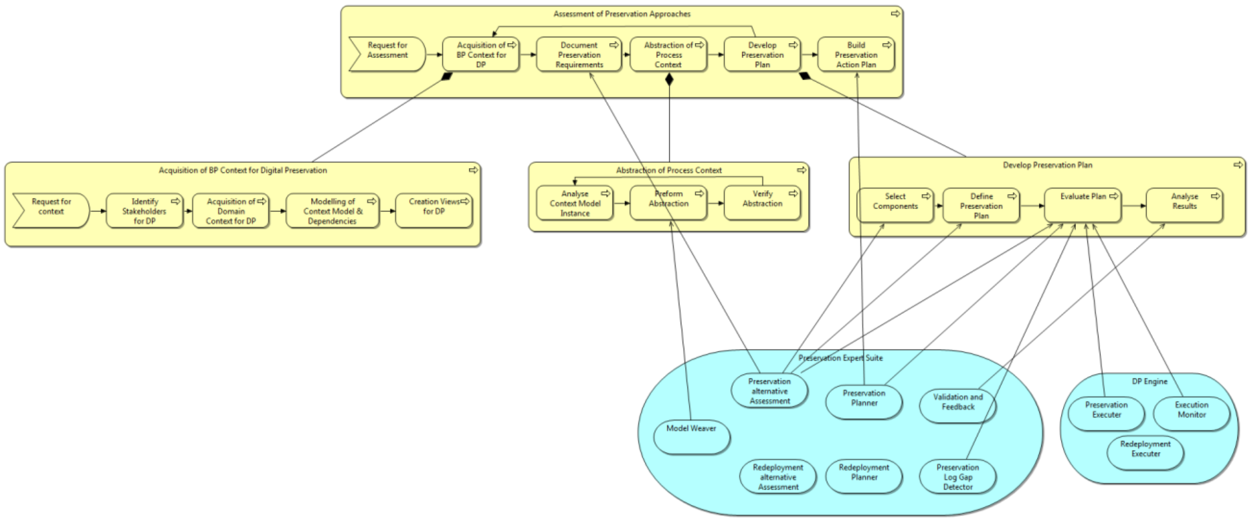

Assessment of Preservation Approaches
Request for Assessment
Acquisition of BP Context for DP
Document Preservation Requirements
Abstraction of Process Context
Develop Preservation Plan
Build Preservation Action Plan
Acquisition of BP Context for Digital Preservation
Request for context
Identify Stakeholders for DP
Acquisition of Domain Context for DP
Modelling of Context Model & Dependencies
Creation Views for DP
Abstraction of Process Context
Analyse Context Model Instance
Preform Abstraction
Verify Abstraction
Develop Preservation Plan
Select Components
Define Preservation Plan
Evaluate Plan
Analyse Results
Preservation Expert Suite
Preservation alternative Assessment
Preservation Planner
Validation and Feedback
Model Weaver
Redeployment alternative Assessment
Redeployment Planner
Preservation Log Gap Detector
DP Engine
Preservation Executer
Execution Monitor
Redeployment Executer

Figure 20: Assessment of Preservation Approches - Approaches Application Usage View

3.5.4 Preservation

Services of the DP Agent, DP Engine, Preservation Expert Suite and IERM are used by the process steps of the preserve phase. The service of the Preservation Repository is used for archival storage as shown in Figure 21.

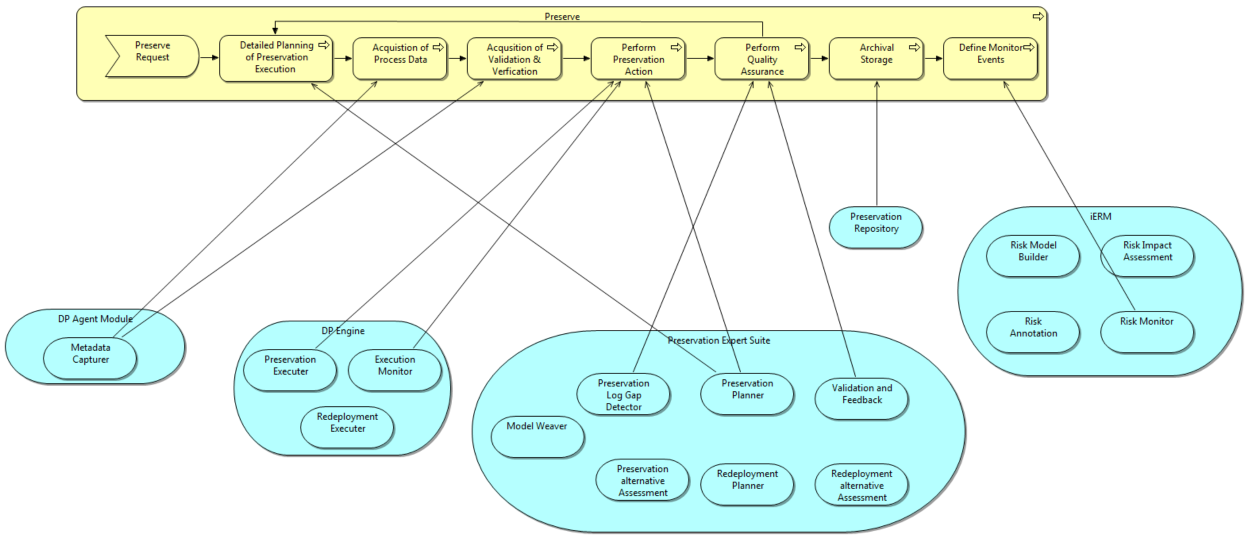


Figure 21: Preserve - Application Usage View

3.5.5 Redeployment

The services of the TIMBUS architecture used by the redeployment process are shown in Figure 22. Key services are the Redeployment Planner and the Redeployment Executor. The verification of the redeployment results is supported by the Validation and Feedback component, the Preservation Log Gap Detector and the Execution Monitoring service. The monitoring and review process of TIMBUS presented in Section 3.4.5 uses the same architectural services as the redeployment process.

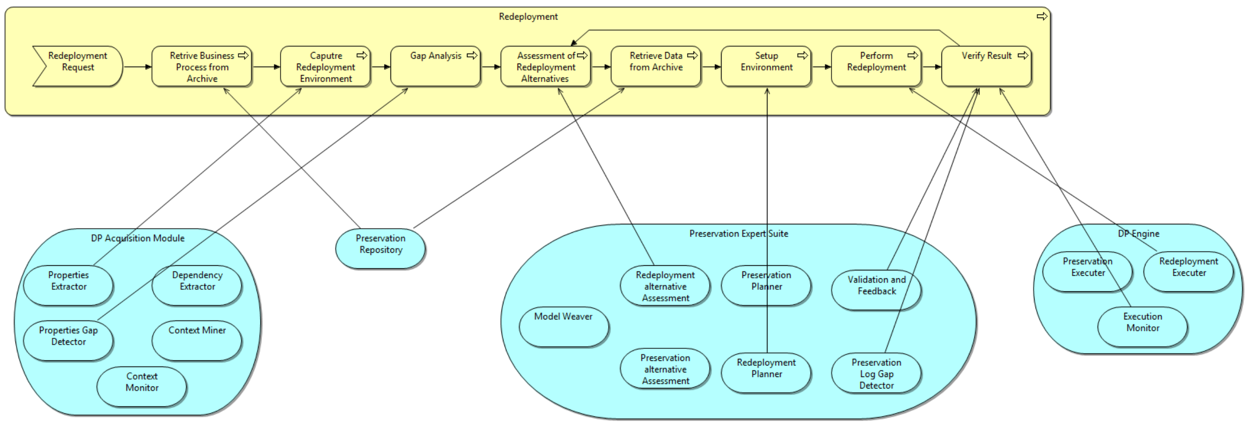


Figure 22: Redeployment - Application Usage View

3.6 Use Case

The TIMBUS process to digitally preserve a business process described in Section 3.4 provides a framework that can be adopted and adjusted to the needs of specific settings. The framework was designed considering the requirements of the TIMBUS use cases (cf. [1] and [2]). The TIMBUS process provides a flexible and extensible framework that can be instantiated for all kinds of business processes. In this section, examples for the implementations of the process framework for specific process steps are presented using the TIMBUS use cases. Section 3.6.1 outlines the TIMBUS process for the civil engineering use case of WP8 and Section 3.6.2 for the eHealth use case of WP9.

3.6.1 WP8 Industrial Project: Civil Engineering Infrastructure

The civil engineering use case deals with the monitoring process of large civil engineering structures such as dams. LNEC[24] is required by the Portuguese national legislation to monitor the structural safety of the structures. The business process in this use case covers the data gathering and the analysis for monitoring of the building structure by using the GestBarragens system. A detailed description of the use cases is given in [1]. The data analysis consists of multiple processes. For the TIMBUS use case seven core processes are selected and shown in Figure 23.

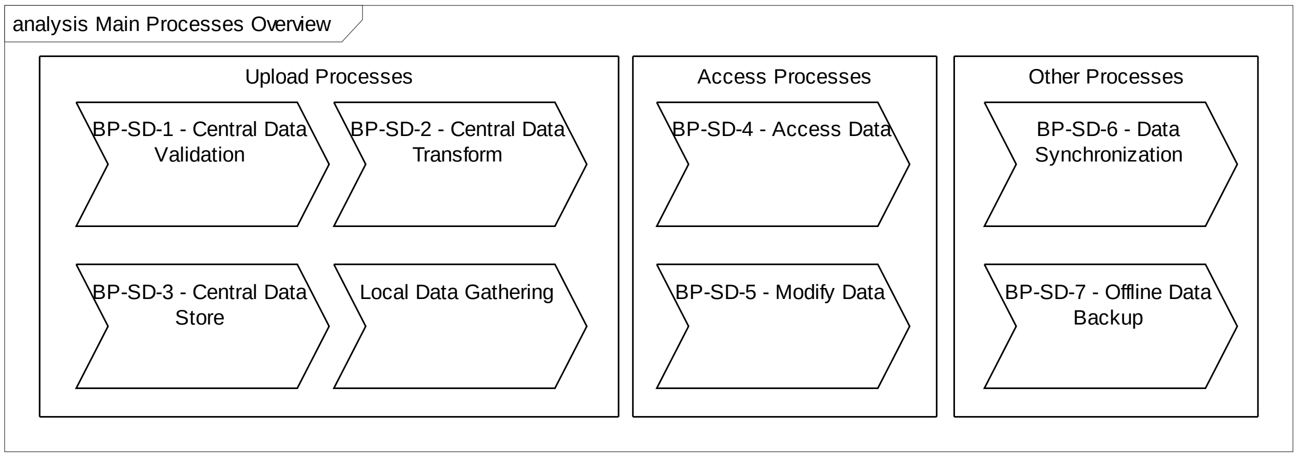

analysis Main Processes Overview
Upload Processes
BP-SD-1 - Central Data Validation
BP-SD-2 - Central Data Transform
BP-SD-3 - Central Data Store
Local Data Gathering
Access Processes
BP-SD-4 - Access Data
BP-SD-5 - Modify Data
Other Processes
BP-SD-6 - Data Synchronization
BP-SD-7 - Offline Data Backup

Figure 23: GestBarragens main processes overview

In order to run the TIMBUS processes, the stakeholders as defined in Section 3.3.2 need to be assigned to actors at LNEC. The organisational structure of LNEC is shown in Figure 24 and described in [1]. Table 7 presents the assignment of the stakeholders of the TIMBUS framework to the stakeholders at LNEC, which is visualised in Figure 25. The Functional Department for this use case is Concrete Dam Department. The mapping is not a 1:1 match. Some stakeholders have one or more roles in the TIMBUS process. The Process Operator for BP-SD-1, BP-SD-2, BP-SD-3 and BP-SD-5 is the Concrete Dam Department technician while for

[24] http://www.lnec.pt/

BP-SD-6 and BP-SD-7 it is the IT Technician. BP-SD-4 is operated by the Concrete Dam Department technician and the structure owner. The matching of the TIMBUS stakeholders is important to have all relevant stakeholders involved.

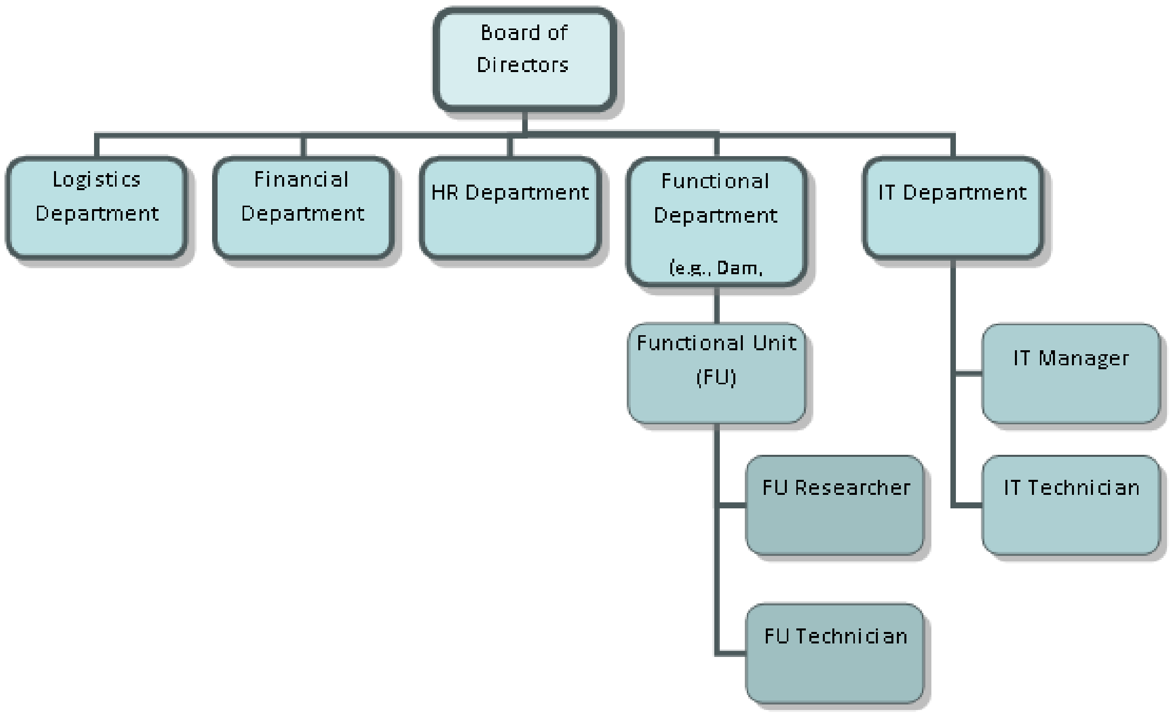


Figure 24: LNEC Organization Diagram

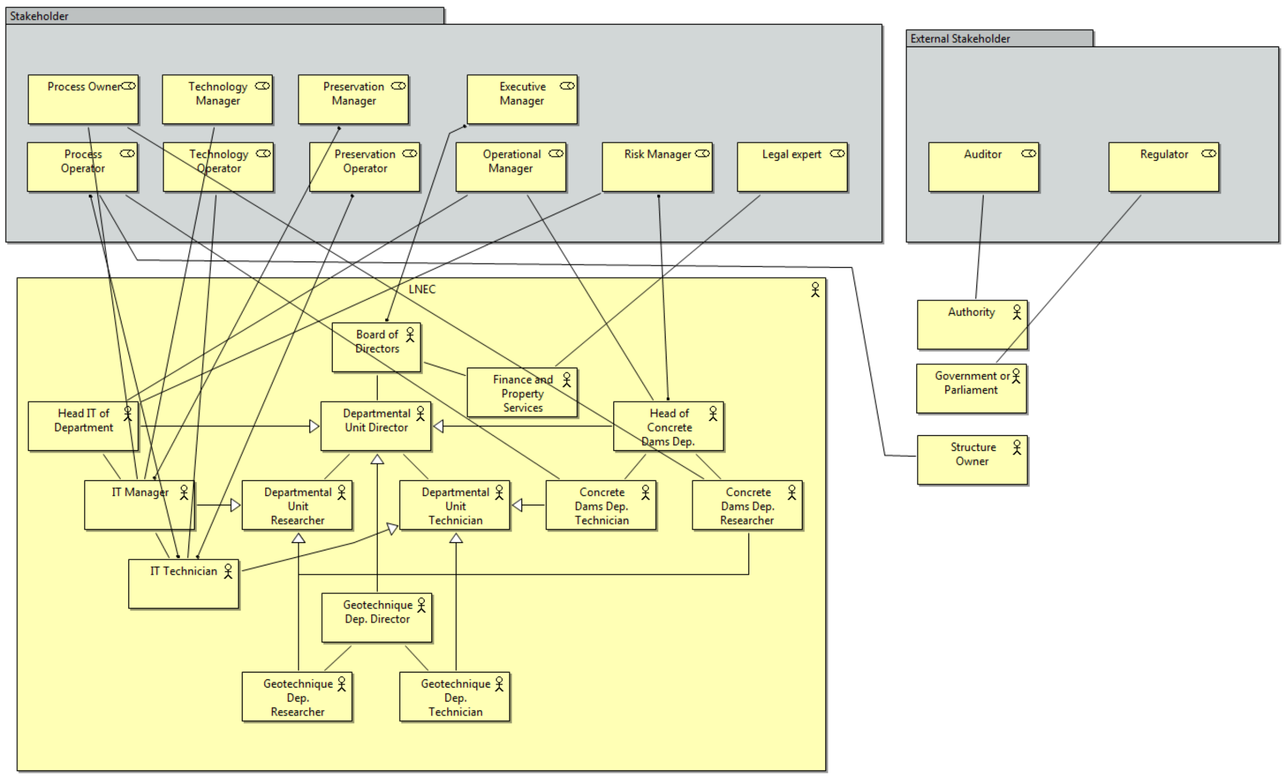


Figure 25: LNEC stakeholders

Table 7: Stakholder assignment

TIMBUS Framework	LNEC
Process Owner	Concrete Dam Department Researcher, IT Manager
Process Operator	Concrete Dam Department technician Structure Owner, IT Technician
Technology Manager	IT Manager
Technology Operator	IT Technician
Preservation Manager	IT Manager
Preservation Operator	IT Technician
Executive Manager	Board of Directors
Operational Manager	Head of the Concrete Dams Department, Head of the IT Department
Risk Manager	Head of the Concrete Dams Department, Head of the IT Department
Legal Expert	Finance and Property Services
Auditor (external)	Authority
Regulator (external)	Government or Parliament

As the TIMBUS process is driven from Risk Management, a detailed analysis of the risk associated with business processes is performed in a first step. In order to analyse the risks of the process, the context information of the process needs to be gathered (see Section 3.4.2.1). Figure 26 shows the context acquisition process for this use case.

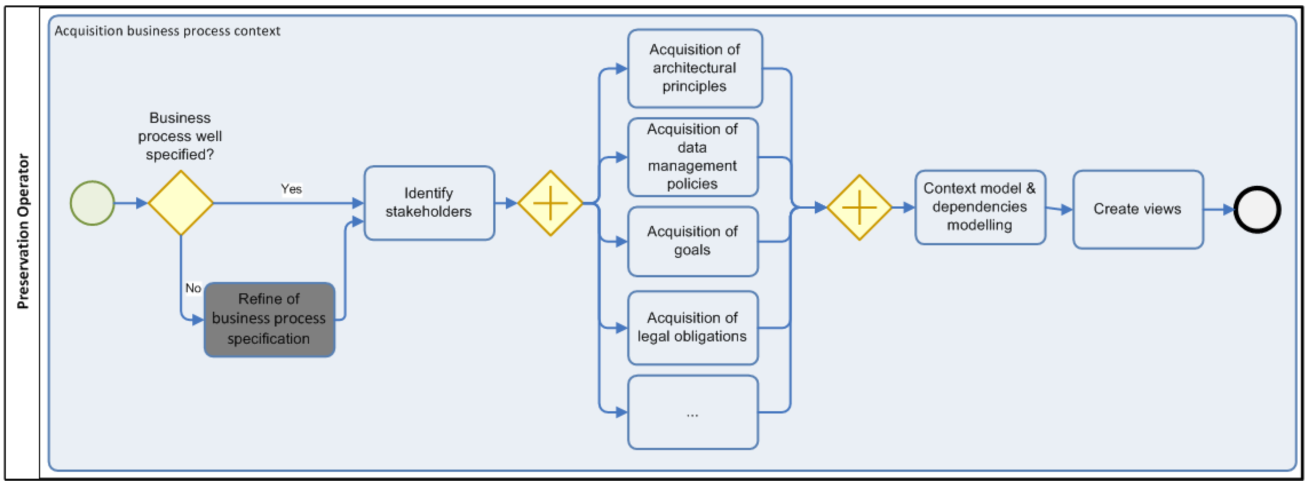

Preservation Operator
Acquisition business process context
Business process well specified?
Yes
No
Identify stakeholders
Refine of business process specification
Acquisition of architectural principles
Acquisition of data management policies
Acquisition of goals
Acquisition of legal obligations
...
Context model & dependencies modelling
Create views

Figure 26: LNEC Acquisition of business process context

Decision: business process well specified?	**Input**	Process descriptions and specification
	Output	Process is well specified. Specification can be found in [1] Chapter 5.8, BP-SD1 - BP-SD7 and Appendix of [1]
	Stakeholder involved	R: IT Manager A: IT Technician
	Description	In the first step the process specification and descriptions are checked for completeness. In this use case the process are well specified and documented. The process description is summarized in section 5.8 and in the appendix of [1]. The overview diagram of the six processes is shown in Figure 23. No further refinement is required and the process to identify the stakeholders is triggered.
Identify stakeholders	**Input**	Documentation of BP
	Output	LNEC stakeholder description [1]
	Stakeholder involved	R: IT Manager A: IT Technician C: Concrete Dam Department Researcher
	Description	The stakeholders involved and their preservation concerns are documented. For the first iteration of the context acquisition the risk management is

		focused on the identification of associated risks of the business processes. The stakeholder can be seen in Figure 25. The concerns of the directors and the board of directors are documented.
Modelling of context model & dependencies	**Input**	Source environment
	Output	Document [1] (Chapter 5 and 7)
	Stakeholder involved	R: IT Manager A: IT Technician C: all other stakeholders (e.g. Concrete Dam Department Researcher, Structure Owner)
	Description	In order to identify and analyse risks of the processes, obligations, guidelines and principles that apply to the processes are of interest. Document **[1]** describes the architecture principles, data management policies, goals and legal obligations of the processes. The fundamental concepts of the implementation are captured as well.
Create views	**Input**	Output of the previous process
	Output	LNEC process Views (as described in document [1], Chapter 5 and 7)
	Stakeholder involved	R: IT Manager A: IT Technician
	Description	The requirements and obligation for the processes are summarized in tables in [1]

The risk management process is described in detail in [27]. It presents the process of identifying, analysing and evaluating risks. Many identified risk with a high risk level were driven by obligations that need to be fulfilled by LNEC (e.g. LR01 – Non-compliance with General Legal Obligations, LRP01 – Non-compliance with Data Protection Regulations, LRP03 – Non-compliance with Laws and Regulations due to Law Changes). Other risks deal with reputation and integrity of information at LNEC (SR01 – Reputation Loss/Loss of Trust and OR07 – Loss of Integrity of Information, OR11). Digital preservation is a possible solution to mitigate some of these risks. The *Assessment of Preservation Approaches* for the LNEC use cases follows the TIMBUS framework as described in Section 3.4.

Acquisition business process context for Digital Preservation	**Input**	CMI, business goals, preservation policies
	Output	enhanced CMI
	Stakeholder involved	RA: IT Manager C: all other stakeholder
	Description	The architectural view of the process is shown in Figure 27. It shows the components and relationships across the different layers of the process. It shows an aggregated view of the processes as well of the technical implementation of the supporting information system. More detailed views of the process are shown in the Appendix of [1]. The TIMBUS context model instance of the LNEC use case with domain-specific model is developed as part of Task 4.2 + 4.4 and WP8. A first outlook on the domain-specific model for the sensors involved in the process is given in [22].

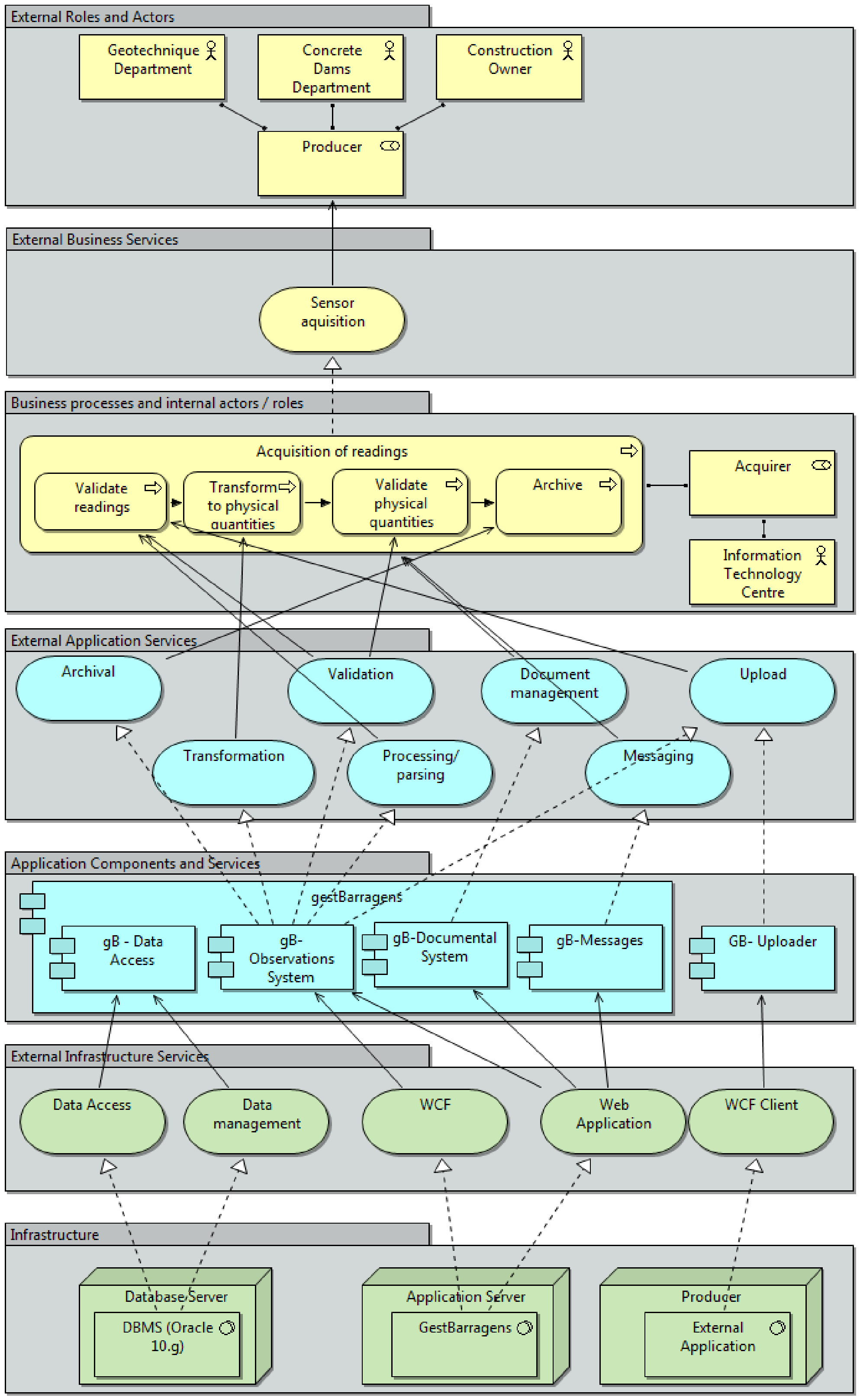


Figure 27: Architectural, Layered View of the LNEC process

Document preservation requirements	**Input**	CMI, business goals, preservation policies
	Output	List of preservation requirements for the business process ([1] Section 7.5)
	Stakeholder involved	R,A: IT Manager C: all other stakeholders
	Description	Document [1] presents the process to identify the preservation requirements for a potential preservation solution. Driven by goals (shown in Figure 28), reuse scenarios, obligations, policies and principles the specific requirements of the process are identified. They are categorized into functional and non-functional requirements.

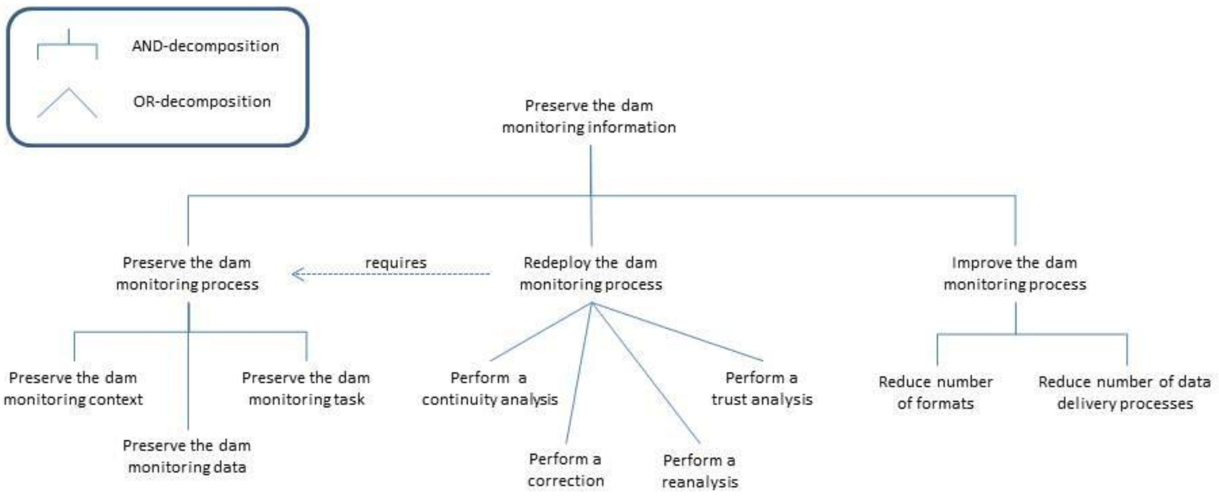

AND-decomposition
OR-decomposition
Preserve the dam monitoring information
Preserve the dam monitoring process
requires
Redeploy the dam monitoring process
Improve the dam monitoring process
Preserve the dam monitoring context
Preserve the dam monitoring task
Preserve the dam monitoring data
Perform a continuity analysis
Perform a trust analysis
Perform a correction
Perform a reanalysis
Reduce number of formats
Reduce number of data delivery processes

Figure 28: Goals identified for preservaing the dam monitoring [1]

Abstraction of process context	**Input**	CMI, preservation requirements
	Output	Finalized CMI
	Stakeholder involved	RA: IT Manager
	Description	The final CMI of the LNEC use case is under development. The current version does not need further abstraction as the model was designed especially for preservation.
Develop	**Input**	CMI, preservation requirements

Preservation Plan	**Output**	List of evaluated preservation plans
	Stakeholder involved	RA: IT Manager
	Description	The identification of the preservation strategies for the LNEC processes is currently carried out in the TIMBUS project. The work of TIMBUS, in particular WP8 and tool development in WP 6 will address the development of a preservation solution for these processes. Research was done to identify potential strategies for the process and the underlying information systems. For the sensors involved in the process, simulation can be a potential strategy, similar to the mock-up of the web services approach described in Section 3.4.2.4.2. The LNEC GestBarragens information system implements a number of components for managing technical documents and analysis tools for physical and mathematical models. A virtualisation of the complete system helps to capture the required components and its dependencies. For the capturing of the system's data sources (e.g. the Oracle DBMS that supports the LNEC processes), different approaches are currently being analysed.

This section provided an initial description of the framework instantiation of the planning phase for the LNEC use case. More examples of the instantiation are shown in [1] and [27]. The development of a preservation plan is currently done within the project. The preservation and redeployment phase of this use case is planned as described in this document without modification or alternations.

3.6.2 WP9 eHealth

The eHealth use case describes a business process for searching and discovering adverse drug events (ADE). A detailed description of the use case is provided in [46]. Adverse drug events describe any injury caused by medication (whether drugs were used at normal dosage and/or due to overdose). The business process provides a complete solution for discovery and search of such ADE rules. The search functionality is used by doctors or pharmacists in the drug prescription process via on-line services. In case of incorrect recommendations of the systems, the prescribed drugs can cause serious complications to the patient health. Lawsuits are potential consequences of incorrect rules. In order to investigate on incorrect ADE rules the discovery process needs to be reconstructed and rerun in the future. The redeployed process therefore needs to be executed on the same hardware/software stack and use the same input data for the discovery process.

The process is highly distributed across three companies as shown in Figure 29. The three companies are DrugFusion, DataMole and SemanTech. Two external services are used by the process Central Medical Repository for Drug Prescriptions (CMRDP) and Pharmaceutical Company (PC).

The key driver for the preservation of the process is DataMole. They are providing a discovery service for new ADE rules by using the unique discovery algorithm. The algorithm is under constant changes due to the introduction of new features, bugs fixes or aligning with new drug taking guidelines. With the preservation of the process DataMole wants to document and prove that their discovery algorithm worked correctly for all rules that were identified. The motivation for the preservation is to prevent potential lawsuits against the company for incorrect rules provided to DrugFusion. A critical task in this use case is the identification of the relevant functionality and components in the distributed environment that need to be preserved. The use case is in an early stage of the preservation process. The TIMBUS framework can be applied on the use case without adoption of the process flow. A refinement of the process specification was triggered and the final documentation is provided in [46]. Figure 30 shows an overview of processes involved in the eHealth use case. The next step is the identification of the stakeholders and the creation of a context model. TIMBUS deliverable D9.3 [46] presents very detailed models about the implementation of the information system involved in the business process. It shows the software components and the underlying infrastructure of the three companies. The communication between the three companies is mainly implemented with Web services. Figure 31 shows the hardware and infrastructure technologies of the business process. For the context model of the use case two domain-specific ontologies (DSO) are currently being developed: software DSO and licence DSO. The software DSO specifies the software implementation including packages, dependencies and communication between the packages. The licence DSO provides detailed information about the software licences involved in the business process. The two domain specific ontologies allow for a detailed model of the business process context and further support the preservation of the process. The detailed information about the software implementation can be used to identify potential preservation strategies for the system involved in the process. The DSOs are present in deliverable *D4.3 Dependency Models Iter. 2* [22].

TIMBUS deliverable D9.3 [46] outlines a first set of risks, identified based on the business process documentation and the current state of the context model. The process framework is not just a strict step by step process. It rather suggests phases of actions with defined output. Before a certain phase can be finished and the next phase can be started all actions defined in the current phase have to be executed and all required information has to be gathered and documented. Tasks in the same process phase can be started at the same time. Processes build on the results of previous steps, but an initial start of preceding process steps can provide useful feedback to previous steps. The eHealth use case shows a good example for this feedback loop that is not explicitly modelled in the process framework (the feedback loop is for the evaluation of the preservation plan as shown Section 3.4.2.4). A first risk analysis before the final version of the context model has been completed. A number of risks were identified, mainly concerning non-compliance with legal obligations, licences and contracts of the process. The identified risks can raise requirements for the context model. In the case of the eHealth process, more detailed information about the legal aspects of the context need to be gathered and documented in the context model. Moreover, [46] provides a set of preservation requirements for the eHealth process. The current version mainly addresses technical requirements for a potential preservation solution.

The work on the use case will be continued in WP9 following the process framework presented in this document. A first version of the context is finished and will be further extended by DSO specifying the domain specific aspects in more detail. The first set of identified risks shows that a preservation solution can be a vital strategy that helps mitigate many compliance risks regarding law and contracts. Virtualisation of systems can be used to capture and preserve the services provided by the information systems. A key aspect of the eHealth use case are the distributed services used that are operated by three different companies. The use of external services raises a challenge for the long term preservation. Escrow agreements as described in Section 4 can provide a potential solution mitigating the risks imposed by external services.

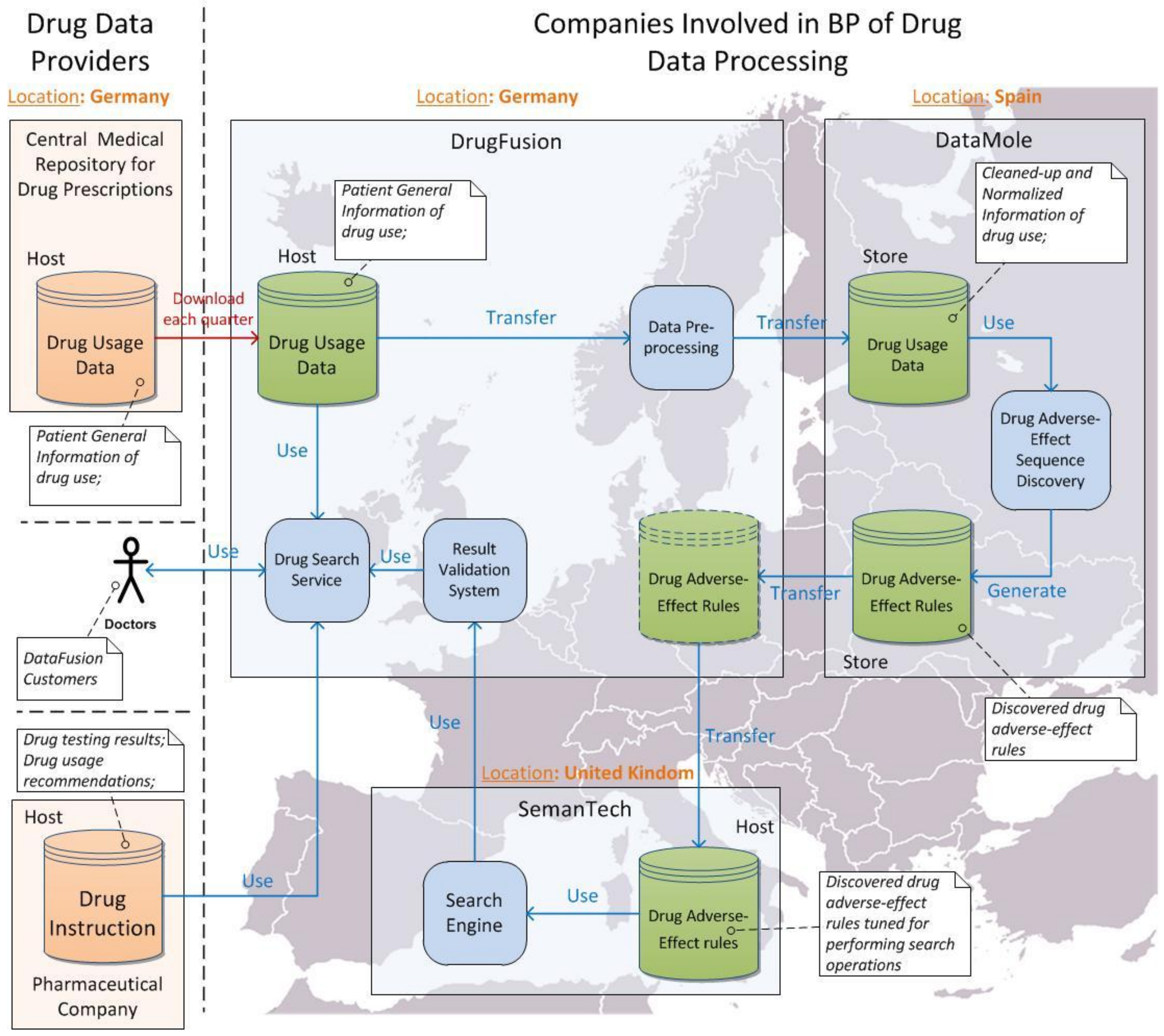

Drug Data Providers
Location: Germany
Central Medical Repository for Drug Prescriptions
Host
Drug Usage Data
Patient General Information of drug use;
Download each quarter
Doctors
DataFusion Customers
Drug testing results; Drug usage recommendations;
Host
Drug Instruction
Pharmaceutical Company
Companies Involved in BP of Drug Data Processing
Location: Germany
DrugFusion
Patient General Information of drug use;
Host
Drug Usage Data
Transfer
Data Pre-processing
Use
Drug Search Service
Use
Result Validation System
Drug Adverse-Effect Rules
Location: Spain
DataMole
Cleaned-up and Normalized Information of drug use;
Store
Drug Usage Data
Use
Drug Adverse-Effect Sequence Discovery
Generate
Drug Adverse-Effect Rules
Store
Transfer
Discovered drug adverse-effect rules
Location: United Kindom
SemanTech
Host
Search Engine
Use
Drug Adverse-Effect rules
Discovered drug adverse-effect rules tuned for performing search operations

Figure 29: High level diagram of the e-Health business process [46]

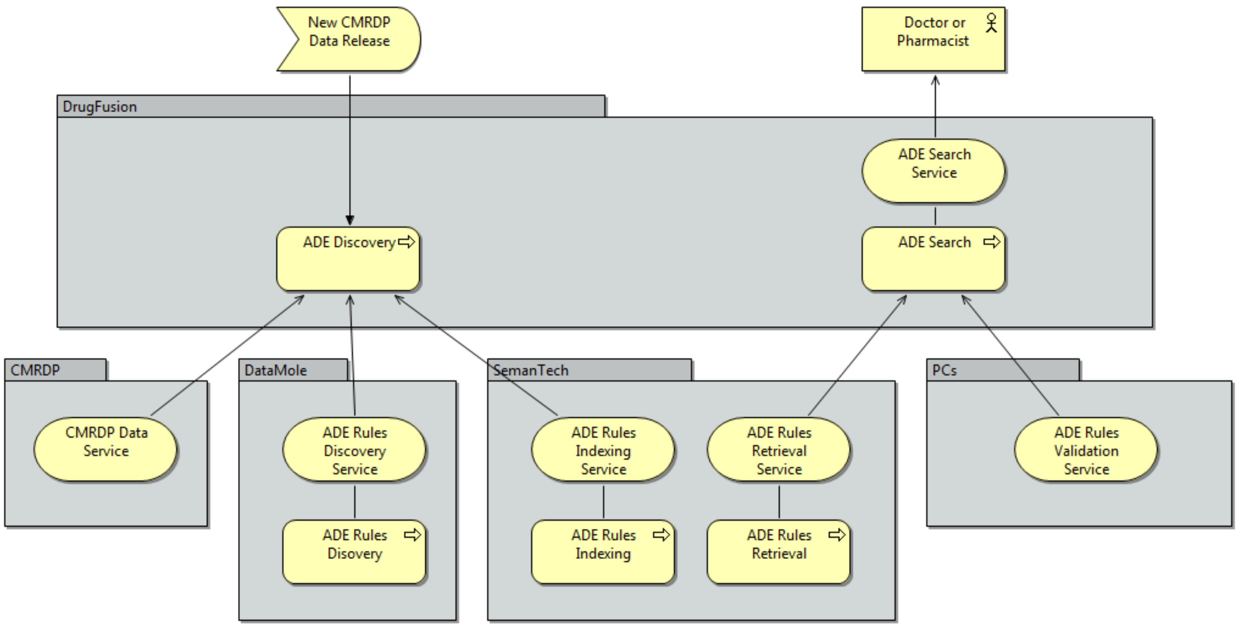


Figure 30: eHealth Business Layer: General Overview

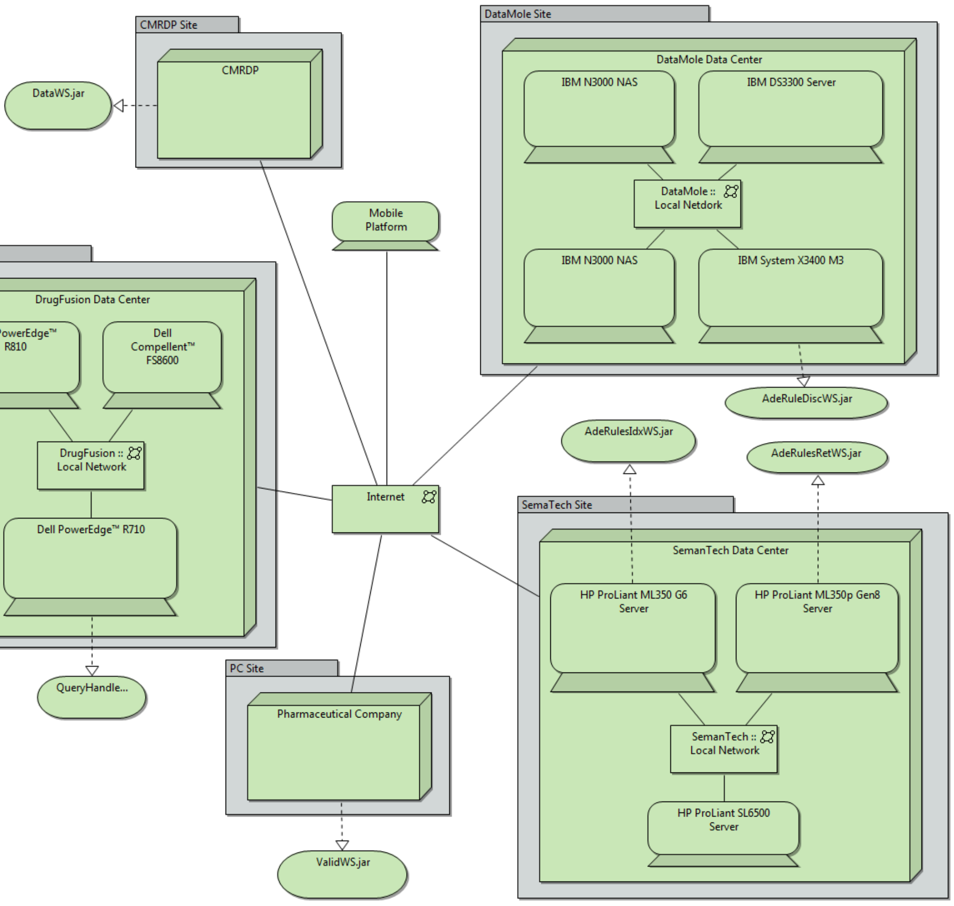


Figure 31: eHealth Technology Layer: General Overview

3.7 Summary

The TIMBUS process presented in this work provides a flexible framework for preserving business processes. Our Digital Preservation approach for business processes is driven by a risk management perspective. The TIMBUS framework is divided into three phases: plan, preserve and redeploy. The planning phase includes the acquisition of the process context, describing all artefacts that are relevant for the process including technical, organisational and legal aspects. A risk assessment identifies and evaluates risks associated with process in the organisation. A generic enterprise risk management process is used based on the ISO 31000 [4]. TIMBUS focuses on Digital Preservation as risk mitigation strategy for identified risks. TIMBUS allows assessing different preservation strategies during the planning phase. In order to develop a preservation strategy, the context of a business process needs to be acquired. TIMBUS developed a context model that provides a basic structure that can be extended with domain-specific knowledge. The model allows capturing the context of the process including technical, organisational and legal aspects across different layers such as business, application and infrastructure layer. It further enables to model the relationships and dependencies of the components. The holistic model of the process can be used to identify components and concepts of the process that need to be preserved over time to meet the preservation requirements. A combination of preservation strategies can be used to preserve the significant properties of the process.

We discussed different approaches to preserve the processes and the underlying information systems. Many well established approaches such as migration or emulation serve as examples. The use of external services is problematic from a preservation perspective, as the availability of these services cannot be guaranteed. TIMBUS developed mitigation strategies for this threat. Mock-up services and escrow contracts with third party providers can provide preservation solution for external services. Software escrow is discussed in more detail in Section 4.

TIMBUS allows comparing different preservation strategies. Hence, cost-benefit considerations support the selection of appropriate risk mitigation strategies. The preservation phase specifies the processes to execute the selected preservation strategy. The preservation process requires the acquisition of the process data from the source environment, the execution of required preservation actions, and the migration of the process into an archival format. TIMBUS allows validating the success of the preservation process. In order to assess the later redeployment, validation data is captured from the source environment. Accompanying quality checks ensure the correct execution of the preservation.

The last phase of the TIMBUS process is redeployment. It defines the reactivation of a preserved business process in a new environment. Beside the technical redeployment of the process, organisational and legal aspects of the process need to be reactivated and adjusted to the changed organisational structures and legal conditions. The redeployed business process is then verified against performance measurements of the original process.

The here presented TIMBUS framework specifies the fundamental steps and actions that need to be taken for the long term preservation and redeployment of a business process. In order to support a wide range of business processes one requirement to the framework was to provide flexibility for implementing the process steps in a domain-specific scenario. We took this requirement carefully into consideration when designing the TIMBUS process framework and provide the needed flexibility. The process defines the required action steps including input, outputs and responsibilities. It guides the process by providing explanations, examples and references to related work. The documentation of the framework focuses on the specific aspects of process preservation with less focus on data centric views of preservation that have been addressed by a number of other projects [47].

The design of the framework was driven by the TIMBUS use cases. The TIMBUS process can be adapted to the individual needs of a setting. The various implementations and characteristics of business processes need to be considered for preservation. The framework should not be seen as a rigid and closed step by step instruction. TIMBUS is rather a specification of required actions and defines what kind of information is required for secure preservation and successful redeployment. It provides a set of process steps that need to be executed during a specific phase, but leaves enough space for the requirements of the original business process. It guides users through the execution of the preservation task by providing recommendations and examples of supporting tools and methods, but the actual implementation of the process and the used methods depends on the specific setting. The framework defines information and procedures that need to be captured, and created for specific phases of the process for further processing. The documentation of the process highlights specific aspects that need to be considered, such as security features or dependencies of components. This ensures that the focus is set on critical tasks. The TIMBUS framework also considers organisational aspects and responsibilities that allow monitoring and controlling of the execution.

The preservation of complex business processes requires manual work and sound decision taking. The input of domain experts is required who provide their knowledge for certain steps such as the creation of context models or preservation plan development. This increases the effort required for the preservation activities. Within the TIMBUS project, WP6 identifies and develops tools to support and automate parts of the TIMBUS processes. In this section we further highlighted how specific components of the TIMBUS architecture support the designed processes for business process preservation. The implementation of the architecture is addressed in WP6 of TIMBUS.

4 Holistic Software Escrow

Like other industries, the IT sector shares the risk of having dependencies to external third parties due to services being increasingly outsourced. In most cases a mitigation of this risk is accomplished by changing the relationship to the external partner by its acquisition and integration into the own organisation. A study by Boston Consulting Group and UBS [48] indicates that nearly one in five of the companies surveyed intends to undertake at least one acquisition. At least 18% of the respondents stated "Access intellectual property and R&D" as main driver for mergers and acquisitions (M&A). So these M&A activities bypass the risks introduced by outsourcing by changing the relationship to the third party. One approach to mitigate the outsourcing risks without having to integrate the partner into the own organisation is Software Escrow, which places a trustable third party between the IT partner and his customer [49].

4.1 Introduction

An acquisition of other companies as mentioned above is sometimes unreasonable or not possible. Thus the only way for companies to acquire the missing knowledge is to outsource tasks to another company. These Business-to-Business transactions also involve the commerce of intangible goods like software. In these cases, the consumer of the contract needs software developed for a specific task. For a certain amount of money he gets the desired program, but due to high costs often only in the form of a license without the possibility to further maintain and develop the software. Hence it is possible for the developer to license his product to other companies as well. For commercial software it is common to only hand over the object code, not the source code including the most valuable trade secrets of the developer. In case the developer goes bankrupt or refuses to support and maintain his product, the consumer cannot get hold of the source code for further development. A mitigation of this risk is to make Software Escrow agreements, ensuring the access to the source code under specific circumstances.

To be able to use the code in the future, all components of the software have to be preserved. The Digital Preservation part in such a software scenario is therefore of utmost importance. It has to be guaranteed that each component of the software, which also includes external services like a Web service, is still available even after it gets handed to the consumer. This includes the deposit of all artefacts that are necessary to develop the source code. However, deciding on what is necessary and when an artefact fulfils quality requirements that make it useful for further development, are crucial decisions when agreeing on a Software Escrow contract. The evaluation of the escrowed material regarding completeness and quality has to guarantee that maintenance and enhancement of the program will be possible for another developer in the future without unforeseeable investment from the consumer. Today, commonly used verification methods (cf. Section 4.3.2.2) examine the deposited material rather superficially. However, without a proper evaluation of the artefacts, a successful redeployment of the software will be troublesome or even impossible, e.g. if the source code is incomplete and documentation is missing. Therefore a manual quality assessment done by the escrow agent will be necessary to guarantee that the software fulfils the desired quality level. A thorough examination can take long however and may not be feasible due to high costs. A cost-benefit trade-off can be achieved by supporting the process with automated artefact analysis. Source code evaluation is already in the focus of common Continuous-Integration tools. However these do not include analyses in respect to Software Escrow, which extends specific Digital Preservation concerns (e.g. calls to an outside Web service) or maintainability issues (e.g. the quality of the comments). Thus we propose a Technical Software Escrow Framework built on the idea of a Continuous-Integration tool. It has to be able to support a manual review with automatic checks of the deposited software, including source code, documentation, and all other artefacts necessary for further developing and maintaining the software. The design of our framework allows dealing with various kinds of artefacts. With its plugin structure it provides different types of measurement for each artefact, e.g. for checking the readability of comments analysing the used libraries for code hidden there or their license. If additional checks are necessary the design allows for extending the framework for these measurements. To support a reviewer examining the deposit material at the software escrow agent, it highlights artefacts that did not pass a test or exceeded a pre-defined threshold.

To give a comprehensive overview, we analysed not only technical aspects of Software Escrow, but included a broad legal point of view as well. For providing a better understanding of the procedure we divided it in three phases (plan, execution, redeployment). The first phase contains mainly legal aspects on what has to be considered and agreed on before the actual escrow takes place. The second phase includes the procedure for the deposit and how to verify the material. Phase three describes the events that precede the release of the software and the release proceedings from a legal point of view.

The structure of the sections is as follows. First, we will present general concepts of Software Escrow (Section 4.1.1) and give a short overview of its phases (Section 4.1.2) before each of them will be discussed in a separate section. Afterwards we will look at what aspects to consider, from a technical (Section 4.1.3) and legal (Section 4.1.4) point of view. The three phases of Software Escrow will be described in the subsequent sections in more detail. Section 4.2 contains the planning part, including how to select an escrow agent (Section 4.2.2), what to consider if contracts already exist (Section 4.2.3) which materials to select for the deposit (Section 4.2.4), and the rights of the materials (Section 4.2.5). The second phase, the execution part, is explained in Section 4.3. It contains information on how the deposit is executed (Section 4.3.2) and explains the verification with the Technical Software Escrow Framework (Section 4.3.3). This section describes the design of a Software Escrow framework (Section 4.3.3.1) and the components needed for a holistic evaluation with focus on maintainability (Section 4.3.3.2). The output of the framework is described as well (Section 4.3.3.3). The third phase, redeployment of the escrowed software in Section 4.4, describes trigger events for a release and the process that follows them (Section 4.4.2) as well as what information has to be exchanged between the parties (Section 4.4.3). The related work Section 4.5 contains the background information of the Technical Framework, including different plugin groups (Sections 4.5.1 to 4.5.4). The description of a prototype implementation for the Technical Framework follows in Section 4.6 with details on the CI tool it was built on (Section 4.6.1) and different experiments carried out (Section 4.6.2) in order to evaluate the framework. In Section 4.7 we conclude the Holistic Software Escrow and give an outlook.

4.1.1 Software Escrow

The term Software Escrow agreement originated in the Anglo-American field of law and refers to contractual agreements, where within the framework of this contract the source code of a computer program and the relevant materials, which will be described later on, will be deposited at a neutral third party and in case of the occurrence of one of the contractually established trigger events, the deposit materials have to be handed over to the consumer.

There can be three escrow parties:

- the **consumer**, who wants to use the software and safe his investments in it at the same time
- the **software developer**, who makes the compiled object code available to the consumer and hands over the sources and all other necessary objects to the escrow agent
- the **escrow agent**, who has to verify that the deposited material meets the requirements as contracted, e.g. that all objects are available and accessible, respectively readable. For this three different verification grades are commonly used, including a standard and full verification and a bespoken verification that can focus on special considerations. More on this topic can be found in [50], in its Section 6.5.3.2 – Verification of Deposit Objects – and in [51].

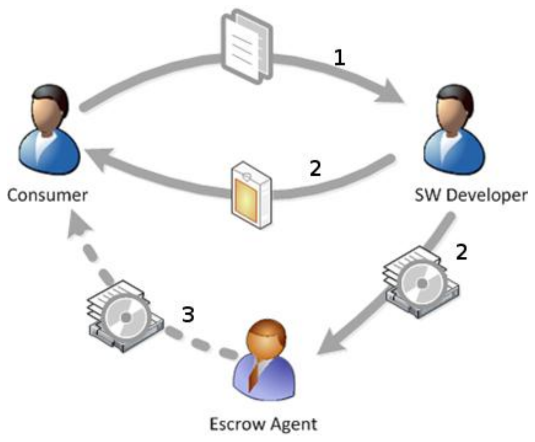


Figure 32: Relationship between escrow parties

Figure 32 shows an illustration of the relationship between the parties. The numbers denote the course of events:

1. The consumer gives his requirements for a program to the software developer.
2. Subsequently the developer engineers a suitable program, whose binary and executable files are delivered to the consumer, whereas the source code, the documentation and everything important for the development and maintenance of the program are committed to the escrow agent.
3. If an event from the escrow agreement gets triggered (e.g. the developer files for bankruptcy), the escrow agent hands out the committed software artefacts from step 2 to the consumer.

The Software Escrow agreement can contain different trigger events, which will be defined below. The escrow agreement helps to solve the conflicting interests of the software developer and the software user. The developer's algorithm is kept secret while the customer's investment is protected by allowing him to get access to the source code and the deposit material under certain circumstances.
The Software Escrow agreement can be a two- or tri-party contract. The contracting parties of the "classical" two-partied contract are the software developer and the escrow agent for the purpose of depositing the source code and the relevant materials at the escrow agent. The escrow agent is authorised to hand over the deposit materials to the software user, who must be named to the escrow agent, in case one of the trigger events occurs.
Basically there are only two contracts, the Software Escrow agreement between the software developer and the escrow agent, and the software licence between the software developer and the software user.[25] The Software Escrow agreement is, in this case, a real contract in favour of a third party (the software) user.
Another contract model for Software Escrow agreements is the chain of contracts, where the software developer signs a contract with the software user, who gains the ownership of the source code and the relevant materials, and who is obliged to use an escrow agent for the execution of the contract. This means that the software developer is obliged to hand the software artefacts directly over to the escrow agent. The software user has the obligation to sign a Software Escrow agreement with the escrow agent. This agreement regulates the occasions in which the escrow agent is obliged to release the deposited materials to the software user. However, between the software developer and the escrow agent there exists no contractual relationship; the developer is only obliged by his contract with the software user to hand over the material directly to the agent, who carries out the deposit of the source code and the relevant material.
It is also possible to establish a Software Escrow agreement in favour of various software users. The Software Escrow agreement can be signed in favour of a determined or an undetermined number of software users. Depending on the specific contracts it can be a two- or tri-party contract.[26]
As mentioned above, the Software Escrow agreement can also be signed as a tri-party contract. In this case, the contracting parties are the software developer, the software user and a neutral third party called the escrow agent. In the earlier years of the use of escrow agreements, the source code was often deposited at a notary or lawyer. Nowadays, there is a tendency towards using specialized companies as escrow agents, which have the knowledge to fulfil the relevant contractual obligations to assure the quality of the deposit material. The escrow agent has the obligation to safeguard the source code and the deposit materials and to verify them. So it is essential that the Software Escrow agreement contains a list of materials that have to be deposited by the software developer, specifies what kind of verification must be made, and what the trigger events are that oblige the escrow agent to hand over the deposit materials to the software user.
Since this contractual constellation is the most common in practice[27], the explanations of this work are based on the tri-party Software Escrow agreement.

[25] *Auer-Reinsdorff/Kast*, Software Escrow, in *Auer-Reinsdorff/Conrad* (ed), Beck'sches Mandatshandbuch IT-Recht (2011) § 10 Rz 35 ff.

[26] A detailed description of Software Escrow agreements signed in favour of several users is contained in TIMBUS Project, D4.4: Digital Preservation & Legal Issues – Overview [50].

4.1.2 Phases

The typical life cycle of Software Escrows can – like the general TIMBUS three phase approach described above in Section 3.3.1 – in turn be broken down into three phases, i.e. escrow planning, escrow execution and escrow redeployment. A functioning Software Escrow agreement must be aware of all these phases and provide a contractual solution for all the potential problems threatening their unobstructed execution. The first phase in the abovementioned life cycle is the *escrow planning* phase. Software escrows should be seen in the broader context of risk management and business continuity planning activities respectively [52]. In the on-going IT outsourcing trend they constitute a mitigation measure minimizing the risk exposure of enterprises. Escrow agreements are able to support the re-execution of business processes with minimum time and minimum expenses. Thus the *escrow planning* phase starts by identifying and analysing the critical business processes that are depending on the software put into escrow. A cost-benefit analysis can help locate crucial parts of the software for which escrow is needed. These parts will be evaluated for their quality in order to determine if they are suitable for further development in the future. The phase finishes by drafting an escrow agreement based on the findings of the previous steps. In the *escrow execution* phase the verification with the Technical Software Framework and the deposit of the materials selected for Software Escrow takes place. This part also comprises suggestions on the optimised contractual solutions for the processes executed therein, e.g. the design of a suitable information procedure regarding the deposit of source code at the escrow agent. The same considerations also apply to the third and last Software Escrow phase, *escrow redeployment*. The quintessential task of this phase is the quick release of the escrowed materials in case a triggering event occurs.

In practice, all of the decisions on the drafting of the Software Escrow agreement, i.e. regarding the concrete contractual design of the three escrow phases, are made in the escrow planning phase. However, for a better understanding of the chapter on "Holistic Software Escrow" and in order to improve its readability, the procedures that, in combination, form the execution phase and the redeployment phase respectively, are described in detail in the corresponding Sections "Phase II – Execution" (4.3) and "Phase III – Redeployment" (4.4), not in the Section on the escrow planning phase (4.2).

4.1.3 Technical considerations

When analysing software that has to be put into escrow, an evaluation has to consider different points of view. General aspects are derived from the knowledge on archiving digital data, digital preservation. Other considerations deal with the quality in software development, including programming mistakes or bugs. We will also take a look at the possibility of code obfuscation or code hiding in the following Sections.

[27] *Auer-Reinsdorff/Kast* in *Auer-Reinsdorff/Conrad,* § 10 Rz 36; e.g. NCC Group, http://www.nccgroup.com/en/our-services/software-escrow-verification/software-escrow/types-of-agreements/ (13.3.2013); SES, http://www.s-e-suk.co.uk/software-escrow/escrow-agreement-options/ (13.3.2013); TÜV Süd, http://www.tuev-sued.de/industrie-konsumprodukte/dienstleistungen/hinterlegungsservice-escrow/tuev-sued-software-escrow/escrow-agent-lizenzgeber-lizenznehmer (13.3.2013).

4.1.3.1 General aspects

Software has specific risks that have to be considered for Software Escrow. Direct risks for Software Escrow include:

- Reading errors on the medium – Due to improper archival of the medium or defects in the files the entire data or part of it cannot be read.
- Incomplete source code – Without the complete source code a successful compilation will be impossible.
- Configuration/instructions not available – Without a proper configuration or instruction the compilation and setup might not be possible.
- Insufficient documentation – Without a good documentation maintenance and development of software are difficult to impossible.
- Absent licenses – The licenses for libraries or the build environment are needed to make further development of the software.
- Objects are out of date – To be of value, escrowed software always has to be updated to the same version that the consumer uses.

To face these problems, completeness and quality of the escrowed artefacts has to be guaranteed. Completeness begins at defining all important objects of escrowed software. The source code alone is not enough for a future maintainability or development. Program description, compiler, or documentation are essential objects for preservation as well. A more exhaustive list of artefacts to preserve can be found in Section 4.2.4. When completeness is guaranteed, the quality of each object has to be evaluated.

Quality includes that data is kept up to date and that its maintainability is taken into account. Other than common material put into escrow, software development is a continuous process, constantly changing the

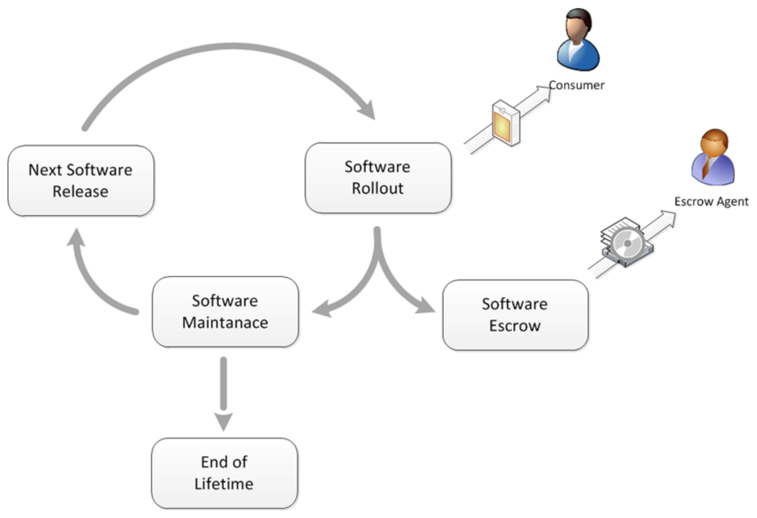


Figure 33: Software release cycle with escrow

deposited material due to bug fixes or updates. Thus it is important to state in the escrow agreement a regular update of the escrowed artefacts, simultaneously done with the delivery of the binary program update to the consumer. The responsibility of making this update lies on the side of the developer, who has to include this step in his development process. The graphic in Figure 33 depicts this process.

Each object has to be of sufficient quality, e.g. the program source code has to follow standards to be understandable by another developer. To verify this quality, an escrow audit framework will be built. With this, a semi-automated or automated artefact review will be possible, thus providing support to a manual review at the escrow agent. The benefit for the consumer is a large coverage of tested artefacts and their quality assurance. The developer, on the other hand, does not have to hand over his code to the customer directly. Tests are done by an independent third party.

4.1.3.2 Source code considerations for escrow

Regular software maintenance is already a non-trivial issue. But when it comes to Software Escrow, which includes the knowledge transfer for future maintenance activities conducted by a potential third party, preparing a container with software artefacts for this very event becomes a complex task. Therefore the deposited artefacts have to be understandable, especially the source code which represents the core artefact.

When looking at poor quality source code from the view of a Software Escrow customer two different views come up. On one hand, one has to deal with unintentional bugs, which occur frequently during development, and on the other hand, shortcomings created on purpose have to be considered. For the first kind, according to [53], where the authors dealt with a large number of bugs during their research for their bug tracking tool, these can be amongst others:

- Mistakes: Even the best programmer is not immune against simple mistakes like implementing a null pointer exception in Java by using the && operator instead of ||, as the authors of [53] have found many times.
- Language latent bugs: Bugs that do not affect the correct execution of a program until a particular program behaviour occurs, because the compiler cannot check them (e.g. requirements for a serializable class in Java).
- Threads: Concurrency bugs are common in Java because programmers tend to use the method without properly regarding the constraints that come with threading.

Unintentional poor quality code has been a research topic for a long time, thus dealing with these kinds of bugs includes common approaches of established tools. Experimental approaches for analysing the quality of projects with respect to Software Escrow can be found in Section 4.6.2. Dealing with intentional poor quality documentation will be content of the readability experiments, which can be found in Section 4.6.2.1.2.

Dealing with purposeful shortcomings is more difficult. It implies that a developer does not want to hand over his code to the Software Escrow agent for various reasons. He might want the customer to be dependent on him, and thus make the source code useless for a third party. Concealing source code without affecting functionality will be the topic of the following Sections. Another reason for the developer is, that giving away the source code goes along with keeping a certain degree of quality, including a clear structure, good documentation, or best coding practices. This requires more effort on the programmer side.

4.1.3.3 Code Obfuscation

Methods to obfuscate code emerged out of the necessity to hinder others from understanding reverse-engineered software. Reasons can be found in hiding vulnerabilities or simply preventing the theft of intellectual property. Obfuscation is common practice in software engineering domains where the physical access to the software is mandatory and encryption of the code is not an option. Material put into escrow includes the source code already, so we have a different scenario here. According to [54] the goal of code obfuscation is to make reverse engineering a costly and time consuming effort. If obfuscated source code is escrowed, these costs have to be invested by the consumer after the redeployment. To avoid this, the reviewing process of the software, before putting it into escrow, has to take into account different code obfuscation techniques mentioned in [54]:

- Layout Obfuscation – E.g. alteration of identifier names or comments
- Data Obfuscation – Changing data structures to make understanding difficult, e.g. replacing a simple increment variable with a more complex expression.
- Control Obfuscation – E.g. to inline procedures or insert unnecessary dead code
- Preventative Transformations – Exploiting the weaknesses of specific decompilers to stop them from operating

Practical approaches like in [55] distinguish between disassembly and a decompilation part and focus on the former one with the goal of entirely thwarting it.

In [56] problems came up when doing experiments and evaluations with non-obfuscated and obfuscated code. The decompilation of the obfuscated sources could only be done with one decompiler, namely Dava[28], and the constraint of disabling most of the obfuscations because otherwise the decompilation was thwarted. Thus obfuscators can be seen as quite effective, hindering decompilers to reverse-engineer software and producing highly complex code if the decompilation is successful.

4.1.3.4 Hide source code

An approach to hide only parts of a source code may be hiding source code in binary form. Two methods will be discussed in more detail, using the example of Java: hiding source code in libraries and running binaries from inside the code, which is closely related with running system specific programs.

METHOD 1 – HIDE IN LIBRARIES

The first method, hiding source code in libraries, deals with program logic in a binary file. For instance, a programmer who wants to hide his Java code from the escrow could add compiled *.class* files to a commonly

[28] http://www.sable.mcgill.ca/dava/index.html

used library like *Apache Commons*[29]. By importing the library into a project, the hidden class and its functionality could be used easily.

ADDING FILES

In the first case we assume that only files were added, the other files of the library were not altered in any way. If the developer used a publicly downloadable library that is still accessible during reviewing of the source code, then the submitted library can easily be tested by comparing the file lists of both libraries. A simple count of the number of files could also quickly help to find inconsistencies between the archives. Both approaches entail a manual review that has to verify that differences are due to purposeful obscuration and are not caused by e.g. using different compilers.

An alternative could be binary comparison. This counteraction involves comparing the library of an escrowed project to the original library on a binary/byte level. Attention has to be paid to different problems arising with such a comparison. If the same library that was used in the project can still be accessed, then a binary comparison should be easy. The situation is different when the library can be accessed in source code form, like it is the case with Apache Commons, and got compiled before adding it to the project. Two different problems arise with this: First, a binary comparison might not help figuring out the changes on the library because there can easily be differences due to different compilers producing different binaries. Second, if the source code of a library is accessible then the developer can easily smuggle his own algorithms into the code without showing discrepancies in the list of class files.

ALTERING FILES

This directly leads to a second case, different to the example in which the files were merely added: altering files in the library. As mentioned before, one precondition for this is free access to the source code of the library. If the library is only available in a binary form, altering files is less likely because the library has to be decompiled and the cost to hide functionality increases. One possibility to check for changes, under the assumption that a more or less simple binary comparison is unsuccessful, would be comparing file size. This could be either on the level of the library to compare the size of a jar file or could be done for each class file in the library. Tests showed that the size of binary files stays the same when using the compilers of Java 6 and 7, so only an alteration of the underlying code might change the file size[30].

Another problem arises if the library used is not publicly available like open source libraries. In this case there might not be a ground truth to compare the escrowed library to. From a technical point of view there are only two solutions in this case: requesting the library from the distributor to compare it, or let him do the comparison himself. However, these cases belong in the legal domain predominantly for which reason they will not be discussed in further detail here.

METHOD 2 – EXECUTE BINARIES

The second method includes binaries that are executed directly from the source code. This includes, for instance, running shell scripts or system inherent programs like copy or make-directory from inside the

[29] http://commons.apache.org/

[30] Of course, replacing instead of adding/deleting source code could be used to not change the file size.

code[31]. These cases have to be checked for bad coding practice, e.g. if only Unix commands are used for a program also running on Windows machines. For a program put in escrow these code parts have to be adjusted or it has to be checked if the executed scripts are within the escrowed material and do not contain any further external references. If other binaries or web services are called from within the code, the escrow agent will have to make sure that they do not contain any hidden source code as well.

4.1.4 Legal considerations

4.1.4.1 Copyright and author's rights on software

With the Computer Program Directive[32] computer programs are copyright protected as literary works and this protection has to be implemented in the national law of the European Member States. The scope of the copyright protection is quite broad and applies to the preparatory design material of the program as well as to "the expression in any form of the computer program". Hence, the program is protected as source code and as object code. However, the underlying principles and ideas of the software are not copyright protected.

The computer program gains protection as the author's own intellectual creation, as determined in Article 1 (3) Computer Program Directive. Recital 8 of the Directive explicitly clarifies that no other criteria should be decisive to determine the protection of the computer program. Therefore it is not necessary that the author proves the fulfilment of any other qualitative or quantitative criteria to gain protection for his computer program. The European Parliament's proposal to adopt a reference to the "author's own creative intellectual effort" was not incorporated in the final version of the Directive. So the understanding of originality in the Directive is significantly lower than in Continental European Law and more or less equals the concept of British and Irish copyright.

Despite this rather low level of originality, computer programs that are absolutely banal and commonplace among the software developing industry are excluded from the Directive's protection as they fail the requirement of being an "intellectual creation".[33] Although according to the Directive there are no requirements such as a certain quality or complexity, programs that are not the result of any creative activity at all and do not require neither skill, nor labour, nor investment must be regarded as commonplace and therefore being outside of the Directive's scope of protection.[34]

Pursuant to Article 2 (1) of the Directive the author of a computer program normally is the natural person or group of natural persons that has created the program. Besides, national legislation can also permit the

[31] An example for Java in Windows would be "Runtime.getRuntime().exec(cmd /C copy file folder)" with which the file *file* gets copied to the folder *folder*.

[32] Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs, OJ L 111/16-22, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:111:0016:0022:EN:PDF (17.10.2012).

[33] *Dreier* in *Dreier/Schulze*, Urheberrechtsgesetz^3 (2008) § 69a UrhG Rz 27.

[34] *Blocher/Walter* in *Walter/Lewinski* (ed), European Copyright Law (2010) 5.1.16.

copyright ownership of legal persons. This is the case, for example, in British copyright law[35], whereas in some Continental countries like Germany only natural persons can be authors of copyright protected works[36].

In Member States such as France, whose legislations recognise collective works, the person who is considered its creator shall be deemed to be its author. A definition of "collective works" is not given in the Directive and thus also left to the national legislator.[37]

Article 2 (2) states that the exclusive rights regarding a computer program created by a group of natural persons jointly also shall be owned jointly. However, the understanding of what is regarded a work of joint authorship varies among the different national legislations. Under British and German law it refers only to uniform works, which merge the contributions of the single authors inseparably and hence can be exploited just as a whole. On the contrary, in countries like France and Belgium already the mere combination of independent works for joint exploitation meet the legal requirements of collective works.[38]

Paragraph 3 of Article 2 contains an important provision for works made by employees: If an employee creates a computer program in execution of his duties or following the instructions given by his employer, the employer – and not the employee himself – by law automatically becomes owner of all *economic* rights in this program. This exception from the basic rule, that the author of a creation always is considered holder of the corresponding copyright (with all its economic benefits), is justified in the face of the regular payment the employee already receives for his labour.[39]

The provision can be modified by contract between the parties though. It does not affect moral rights of the author, such as claiming paternity of the program, and leaves the application of a special moral rights regime to the national legislation of the Member States. In Germany, for example, the author has a certain core of moral rights in his creation which cannot be conferred to others – whereas the British copyright is completely transferable.[40] These and other differences between the Member States' national legislations regarding the moral rights of the creator also explain the use of the broader term "author's rights" instead of "copyright" in the Directive.

Depending on national legislation, the term "employee" can be understood in a broad sense and also include government officials and functionaries.[41] However, the exceptional rule of Article 2 (3) cannot be applied to programs created by software houses or freelance software developers on commission and on the ground of a contract between independent parties; it is limited to programs of employees created on the basis of

[35] *Blocher/Walter* in *Walter/Lewinski*, European Copyright Law, 5.2.13.

[36] *Thum* in *Wandtke/Bullinger*, Urheberrecht^3 (2009) § 7 Rz 8.

[37] *Blocher/Walter* in *Walter/Lewinski*, European Copyright Law 5.2.14.

[38] *Blocher/Walter* in *Walter/Lewinski*, European Copyright Law 5.2.20.

[39] BGH, MMR 2002, 101 = GRUR 2002, 149 = NZA-RR 2002, 202 – Wetterführungspläne II; see Hoeren, Skriptum Internet-Recht 10/2012, 55, http://www.uni-muenster.de/Jura.itm/hoeren/materialien/Skript/Skript_Internetrecht_Oktober_2012.pdf (17.10.2012).

[40] *Wiebe* in *Spindler/Schuster* (ed), Recht der elektronischen Medien^2 (2011) § 69 b UrhG Rz 2.

[41] *Blocher/Walter* in *Walter/Lewinski*, European Copyright Law 5.2.27.

contracts of employment. Besides, the provision only applies to works created by the employee during his working hours and within his contractual duties. Therefore, it cannot be applied to computer programs developed before the beginning or after the expiration of the employment agreement.[42]

But within the area of employment contracts it is the provision of Article 2(3) which allows the employer to exclusively exploit the economic value of the computer programs developed by his employees and confer licenses to others.

4.1.4.2 Software licenses

Often "software licences" – a term originating from Anglo-American law – are referred to as "license contracts" but at least in the continental legal systems of Germany and Austria this type of contract does not exist. Therefore software licences are to be assigned to one of the different types of contracts known to the Private Law of these countries, like sale contracts (*Kaufverträge*), rental/hire contracts (*Mietverträge*), contracts for personal service (*Dienstverträge*), contracts for work and services (*Werkverträge*), mixed contracts (*typengemischte Verträge*) etc., depending on the distinctive features of the agreement in question. The classification of the agreement cannot be based on the name of the contract chosen by its parties but on its content. It can become very relevant to interpret the extent of the stipulations and specify the exact obligations and duties of the parties.

If pre-existing standard software including the right of permanent usage is sold to a purchaser for a singular payment, this agreement normally would be considered a sale contract.[43] But if a software house creates completely new software designed for individual requirements of a customer, this most likely would be based upon a contract for work and services.[44]

However, on a business level none of these two clear situations are very common. Today more and more software houses construct their products by combining already existing parts stored in their software library and adapt the result to the individual needs of the customer.[45] In this case, the applicable contract law depends on the effort necessary for the customisation of the software. As a general rule, an agreement can only be considered a contract for work and services, if its preponderant part actually concerns services (of developing, creating, adjusting, customising the software) and not just sale and delivery of the product.[46] Modifying and customising a standard program, in such a way that it is of no use to anyone else but the customer himself, would be an example.[47] The principal feature of a contract for work and services is the duty to *produce* – against payment – certain *results* defined in the agreement, for example an installed software program performing a set of tasks specified in the contract between software house and client. So the simple sale of standard software without any obligation to successfully adapt the computer program to

[42] *Wiebe* in *Spindler/Schuster,*Elektronische Medien § 69 b UrhG Rz 3.

[43] *Redeker*, IT-Recht^5 (2012) Rz 523.

[44] *Marly*, Praxishandbuch Softwarerecht^5 (2009) Rz 635.

[45] *Conrad/Schneider*, Erstellung von Software, in *Auer-Reinsdorff/Conrad*, § 8 Rz 1.

[46] *Marly*, Softwarerecht Rz 633.

[47] See *Hoeren*, IT-Recht 10/2012, 102, http://www.uni-muenster.de/Jura.itm/hoeren/materialien/Skript/Skript_Internetrecht_Oktober_2012.pdf (21.11.2012).

the particular needs of the buyer does not match the characteristics of this type of contract and belongs to the scope of application of sale contract law.

Apart from the obvious advantage of a contract for work and services regarding the obligation of the software house to not only provide the software but to complete successfully the customising tasks established in the agreement, this type of contract has some additional benefits in comparison with sale contract law in case of defects, e.g. more extensive rights of the customer and a considerably longer warranty period.[48]

But even if the focus of the license agreement is on the providing of software and not on its creation and customising, it is not necessarily a sale contract. Depending on the peculiarities of the agreement at hand, a classification as a rental/hire contract often may be more adequate. This kind of contract usually applies when one party receives a payment in exchange for leaving an object temporary in hands of the other party so the latter is able to make use of it. The essential characteristic for distinguishing these agreements from sale contracts is the permission to use the software only for a limited period of time; another indication may be a regular payment for the use of the program.[49]

There is also the possibility of developing software under a contract for personal service which does not include the obligation to actually finish successfully the making of a program or parts of it but only to be working on it. This results in a very limited liability of the software developer in the event of failure to complete the project he is assigned to. However, usually this type of contract only applies, if the developing party has a low level of self-responsibility and independence, like an employee developing software for his employer.[50]

Summing up, the non-existence of “license contracts”, as such in the Private Law of countries like Germany and Austria, makes it necessary to analyse thoroughly the content of software agreements in terms of their contractual nature to clarify the extent of possibly unclear stipulations.

4.1.4.3 Software maintenance

Besides software license contracts that enable the software user to apply the computer program in its enterprise, software maintenance contracts and Software Escrow agreements can be established to ensure on-going operability and protect the investment in the software. So far, no standard type of software maintenance contract has yet evolved.[51] Such agreements may therefore comprise a wide variety of types of support for the software solution implemented in an enterprise. The regular content of a software maintenance contract can be arranged into three main categories:

1. keeping the software operable and usable through bug-fixing but also through the supply/implementation of patches,

[48] *Redeker*, IT-Recht Rz 297.

[49] *Redeker*, IT-Recht Rz 529, 596

[50] *Redeker*, IT-Recht Rz 497 f.

[51] *Wolner*, Computerrecht, in *Hausmaninger/Petsche/Vartian* (ed), Wiener Vertragshandbuch II^2 (2012) 349.

2. the supply/implementation of updates,[52]
3. application support and consultancy on the functions of the software.

Furthermore, the circumscription of these tasks will then – in contemporary software contract frameworks – regularly be complemented by the appointment of certain service levels regarding the fulfilment of the maintenance operations. So-called service level agreements (SLAs), especially govern response-/resolve times and the quality and scope of the maintenance services to be undertaken pursuant to the software maintenance contract.[53]
The company that developed the software will regularly be the obvious choice for the abovementioned tasks. It is however also possible that a third party is contracted to perform the maintenance of the software, provided that it has access to the source code.[54] From a legal point of view, the classification of a software maintenance contract depends on the definition of the tasks to be fulfilled by the software maintenance provider and on their concrete formulation in the contract. If the maintenance provider obliges himself to deliver certain contractually determined results regarding its tasks, the underlying agreement most likely is a contract to produce a work.[55] If such results cannot be identified in the agreement, usually the provisions on service contracts apply.[56] The latter contractual solution will, of course, be the obvious choice for the maintenance provider as in that case, he will not necessarily have to fix issues like bugs on a regular basis, but only provide his best efforts to do so. Moreover, the decision whether to rent or buy software in the first place will impact the necessity of concluding a software maintenance contract. In case of the software being rented,[57] the rental fee could potentially also cover some maintenance services. There is, however, only an obligation to maintain the software to the degree that it complies with the specifications laid down in the software license contract. It thus depends on the concrete formulation of the license contract whether it also covers maintenance services.[58] Therefore, it is advisable to at least include a section regarding software

[52] An update may be e.g. the configuration of the software to (re-)establish compliance after legislative changes affecting the enterprise of the software user. Furthermore, software maintenance contracts may also comprise the obligation to supply and implement upgrades, i.e. such modifications that – in contrast to updates – add further functionality to the software, cf. *Meents*, IT-Recht, in *Walz* (ed), BeckecktOT Formularbuch Zivil-, Wirtschafts- und Unternehmensrecht Deutsch-Englisch^2 (2010) 1016 et seq.; *Wolner* in *Hausmaninger/Petsche/Vartian* 350.

[53] *Bussche/Schelinski*, IT-Vertragsgestaltung, in *Leupold/Glossner* (ed), MsneragsgestaltungL_CITATION {"cit^2 (2011) Rz 375. For more details on SLAs, TIMBUS Deliverable 4.4. Digital Preservation & Legal Issues – Overview 129 et seq.

[54] *Bussche/Schelinski* in *Leupold/Glossner,* Rz 370.

[55] So-called *Werkvertrag*, §§ 1165 ff ABGB; §§ 631 ff BGB. The definitions of the contract types used in this text follow the terminology used in the English translation of the German Civil Code (BGB) accessible via the legal information system of the Federal Republic of Germany operated by the Federal Ministry of Justice, gesetze-im-internet.de/englisch_bgb/index.html (27.02.2013).

[56] So-called *Dienstvertrag*, §§ 1153 ff ABGB; §§ 611 ff BGB.

[57] The abovementioned considerations for rented software also apply to ASP (application service provider) contracts.

[58] *Iro* in *Koziol/Bydlinski/Bollenberger*, Kurzkommentar zum ABGB^3 (2010) mmentarRz 3.

maintenance in the software license contract to ensure that updates will be performed in case of legislative changes affecting the enterprise of the software user.[59]

Some software maintenance contracts will moreover contain provisions on the regular release cycles regarding the software licensed to the user. There is a strong motivation for software development companies to align their efforts in keeping the software implemented in a specific enterprise operable through software maintenance services and releasing new versions of the same software, e.g. through simple patches, updates, or elaborate upgrades introducing new functionalities to the software. Software users might however not always share those interests and might be reluctant to buy/rent the latest upgrade and implement it in their enterprise, e.g. out of the fear of data loss or additional costs for training of their employees. Also in this case, the on-going maintenance of the old software version still running on the enterprises hardware is jeopardized. In the light of the rapid technological development in this area, it will not be in the interest of the software development company to service several versions of the same software at all times. The duration of maintenance contracts for older software will thus regularly be limited[60] and the software user will correspondingly be in need of another solution to ensure the continuing maintenance of his software. To this end he will be in need of having access to the software's source code. In the case of proprietary software the most widely disseminated contractual model to ensure orderly access to the source code is a Software Escrow agreement. Besides that reality in the software maintenance business, all of the abovementioned types of contracts – depending on the concrete formulation of the maintenance tasks – are to be classified as continuing obligations. As a consequence, a special legal doctrine will regularly be applicable in the case of software maintenance contracts. Pursuant to this non-contractually abolishable legal doctrine, continuing obligations may be terminated for good cause at any time and also irrespective of whether the underlying contract is a fixed-term contract or a contract for an indefinite period of time.[61] Another effect arising from the nature of software maintenance contracts as being continuing obligations is the insecurity of their on-going existence in case of the insolvency of the maintenance provider since the liquidator then has the choice whether to terminate or meet the maintenance contract.[62] The insecurities regarding the continuing availability of maintenance services because of the various points made in this section adds emphasis to the argument made above, namely the need to ensure access to the software's source code through the contractual means of an escrow agreement.

4.1.4.4 Motivation for circumspect contract drafting – the Telecom NZ story

The recent case of *Telecom New Zealand* v *Aldous*[63] serves as a good example for the importance of circumspect contract drafting when planning to engage in a Software Escrow agreement. Telecom New

[59] *Wolf*, Taktische Vorschläge zur Gestaltung von Softwareverträgen (1. Teil), EDVuR, 6 (19).

[60] Eg *Wolner* in *Hausmaninger/Petsche/Vartian* 342.

[61] *Bollenberger* in *Koziol/Bydlinski/Bollenberger*, ABGB §(2010Rz 7.

[62] § 21 atIO.

[63] The following case description follows the recent litigation between *Telecom New Zealand* v *Aldous*,
High Court of New Zealand 26.06.2012, CIV-2012-404-3513 [2012] NZHC 1501, http://www.burgess.co.nz/law/wp-content/uploads/2012/08/Aldous.pdf (27.02.2013).

Zealand, a telecom service provider hereinafter referred to as Telecom NZ, had entered into a software license agreement with Aldous, a software development company. As the particular software has been essential to the operation and the decommissioning of Telecom NZ's (old) CDMA[64] network, a tri-party escrow agreement was concluded between Aldous, Telecom NZ and an escrow agent. Pursuant to the agreement's terms Aldous was obliged to deliver the source code and adjacent documentation to the escrow agent. The deposit procedure foresaw that in case of an update of the software, Aldous would have to deposit these materials within a period of ten days of each update and the escrow agent would then be under an obligation to notify Telecom NZ upon receipt of the materials. Furthermore, the escrow agreement enabled Telecom NZ to demand the release of said materials under certain circumstances, among them i.e. Aldous' insolvency.

The latter scenario became reality. Aldous went into receivership in mid-2012. Since the beginning of the software license contract Aldous had been further developing the software, implementing the corresponding updates in Telecom NZ's enterprise and putting them into escrow. Pursuant to an estimation of Telecom NZ, the cost of such updates amounted to approximately NZD 7.000.000 excluding license and other fees in the period from 2008 to 2012 alone.[65] However, during its financial struggles, Aldous' had carried on updating Telecom NZ's software and implementing these updates, but had ceased to deposit the corresponding source code and adjacent documentation with the escrow agent. From this moment on, the escrow agent had logically stopped informing Telecom NZ of the deposit of new updated code versions, but Telecom NZ did not notice the absence of such messages. After the occurrence of the triggering event of Aldous' insolvency in 2012, the escrowed source code that was released to Telecom NZ proved to be only an old version not matching the updated new software version implemented in Telecom NZ's enterprise.

Due to a flaw in the deposit procedure, namely an imperfection in the contractual design of the information-/notification procedure, paired with Telecom NZ's inattentiveness regarding the monitoring of the software versions implemented in its enterprise, the on-going operability of its CDMA network is now jeopardized. As the receivers appointed to Aldous will be eager to make maximum profit of the new source code as an asset to be exploited for the benefit of the Aldous' creditors, Telecom NZ will probably have to pay a huge sum to get into possession of that source code. A more detailed design of the notification procedure could have saved Telecom NZ the extra financial expenses for the new source code version. More detail on contractual solutions preventing software users of such inequitable situations like e.g. more comprehensive notification obligations or their reinforcement through penalty clauses is given below in the Section Redeployment 4.4.

4.2 Phase I – Planning

4.2.1 Overview

The Software Escrow planning phase must already be initiated at the stage of the software licensing contract negotiations as it is vital for the Software Escrow's on-going existence that its rudimentary provisions are

[64] I.e. Code Division Multiple Access.

[65] *Telecom New Zealand v Aldous*, Rz 7.

already to be found in the software licensing agreement. The planning phase includes all of the preparatory measures that have to be taken before the Software Escrow can be executed. As certain costs are associated with the setting-up of a Software Escrow, the first step should be a risk analysis establishing whether the software contemplated for escrow is crucial for the execution of the business process it is supporting and even more importantly, whether that business process itself is crucial for the enterprise executing it. Once the viability of the Software Escrow is established, the right escrow agent for the job has to be found. The next step is the drafting of a suitable escrow agreement. Such a contract has to be aware of the three phases of Software Escrows, offer solutions for the potential problems which can occur during the lifecycle of a Software Escrow and particularly deal with the IPRs (Intellectual Property Rights) protecting the developer's software.

In that context, it shall once again be emphasized that – although the contract drafting is carried out in the planning phase – for the purpose of this text, the contractual solutions for the execution and the redeployment phase are dealt with in the section corresponding to that phase.

4.2.2 Selection of the escrow agent

It is a cost-effective possibility to deposit the material relevant for the escrow in a sealed envelope at the user's place because no fee for the deposit must be paid. But this option bears different risks for the software developer as well as for the user.[66] In the past, the relevant material was deposited in a sealed envelope in a safe at a bank or at a lawyer or notary. Either the software developer or the user entered into a tenancy agreement with the bank. This form of an escrow agreement had disadvantages. By using a sealed envelope, it was not possible to verify the content of the material and to determine whether the material meets the necessary requirements. The deposited material could be useless at worst. Furthermore, if the agreement was established between the user and the bank, the software developer had no influence concerning the access of the user to the safe. Further specifications were necessary to satisfy the interests of the software developer.[67] Lawyers and notaries were deemed to be qualified because they were able to deal with the relevant legal questions. Nevertheless, banks, lawyers and notaries lacked the professional skills to test and verify the deposited material. Consequently, to be sure that the deposited material was not useless, another company had to be hired ensuring that the technical requirements were met. Furthermore, by deposing the material at a lawyer or notary, the impartiality had to be guaranteed.[68] These considerations have led to the deposit of the material at an escrow agent[69]. A specialized escrow agent has the know-how to ensure the safety and quality of the deposited material. Model contracts to ensure confidentiality between the contracting parties and guidelines for the deposit of updates and the verification process of the

[66] Please see TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 140 f for detailed information.

[67] TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 141; *Lewis/Moor*, Ensuring IP Protection through Escrow, ESL 2003, 8 (8).

[68] TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 141 f; *Kochinke/Fraune*, Hinterlegung von Quellcodes im US-amerikanischen Recht, CR 1992, 7 (11); *Lewis/Moor*, ESL 2003, 8 (8).

[69] Escrow agent in this context means a company specialized in this field.

material are used.[70] Furthermore, the escrow agreement contains all the necessary obligations of the parties and further specifications. Consequently, the different interests of all parties can be served (please see below for detailed information concerning the escrow agreement and the verification process).[71]

4.2.2.1 Requirements regarding the escrow agent's IT infrastructure

Another aspect of choosing the right escrow agent is largely dependent on the type of IT equipment he has at his disposal.[72] Ideally, the escrow agent's infrastructure has to correspond to the technological platform that is necessary to view, compile and run the software to be put into escrow. It would certainly be beneficial for the software user to choose an escrow agent with an in-house verification lab to ensure the independence and impartiality of the verification process. Moreover, such a constellation could also prove useful for the software development company, as the source code and the know-how embodied in it are only exposed to one party, the escrow agent. It can be foreseen in the escrow agreement that only certain qualified personnel of the escrow agent is to be entrusted with the tasks regarding a particular deposit. Furthermore, the escrow agent can be obliged to contractually swear his employees to secrecy about the information gained in course of their work.[73] However, if the parties cannot agree upon an in-house execution of the verification process, the escrow agreement should consider the question of to what extent the escrow agent will be in need of using the software developer's or a third-party's infrastructure to perform the tests on the deposit materials.

4.2.2.2 Data security requirements – liability of the escrow agent for improper storage

The escrow agent can be held liable for damages suffered by the licenser or licensee in accordance with the relevant provisions of law, if he acted intentionally or negligently.[74] The escrow agreement should contain terms of liability. The liability of the escrow agent can be limited to a maximum amount. Subject to the mandatory legal liability, the escrow agreement can also contain a clause establishing liability of the escrow agent only for damage that is typical for escrow work, foreseeable for the escrow agent and if a cardinal obligation has been negligently violated.[75] The agreement must be in accordance with standards established with regard to unfair contract terms. Liability of the escrow agent should be excluded for circumstances resulting from refusal of the contracting parties to fulfil an obligation or from defective fulfilment. An insurance should be taken out.[76]

[70] *Prenafeta Rodriguez*, Sobre el Contrato de Escrow: Naturaleza jurídica y algunos problemas, www.noticias.juridicas.com/articulos/20-Derecho%20Informatico/200203-2755122021023780.html (06.12.2012).

[71] TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 142 ff.

[72] *CEN*, ESCROWGUIDE - Source Code Escrow - Guidelines for Acquirers, Developers, Escrow Agents and Quality Assessors, CWA 13620-4:1999 E, Part 4: A guide to providing a reliable escrow service 17.1.

[73] *Wolner* in *Hausmaninger/Petsche/Vartian* 358 f.

[74] *Wolner* in *Hausmaninger/Petsche/Vartian* 365; *Imhof* in *Weitnauer* (ed), Beck'sches Formularhandbuch IT-Recht^2 (2009) 120.

[75] *NCC Group Escrow Germany,* www.nccgroup.com/media/107015/sl_gmbh__ger__june_2012.pdf (29.11.2012) 11.1.

[76] TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 149 (150).

The deposited material must be stored in a safe and secure environment.[77] The escrow agent is obliged to protect the escrowed material from damage and theft. Furthermore, he must constantly control the temperature and humidity of the storeroom.[78] It must be ensured that the atmospheric environment is adequate to maintain the quality of the deposited material.[79] The material has to be treated as confidential.[80] As it contains the know-how of the software product, the software developer's interests concerning his intellectual property rights must be protected. Furthermore, confidentiality is crucial for the user as well, e.g. if the software was individually created for the user's needs and has a key function in his business. The escrow agent has to oblige persons working for his company and persons assigned to carry out verification procedures to maintain confidentiality.[81] Measures need to be taken to prevent access by unauthorized persons.[82]

The group of people who has access to the material should be limited to a certain number. The licenser should be informed about the group of people who has access to the deposited material. The escrow agent is obliged to identify a person before granting access. He should note the time, duration and purpose of the access and if information was removed or added.[83]

The escrow agent shall not be liable for any delay or failure in performing a contractual obligation if the reason for the delay or failure was beyond the escrow agent's reasonable control, e.g. act of God, war, prohibitions by any governments, or other legal authorities affecting the agreement.[84]

4.2.2.3 Information-/notification obligations

The escrow agent has to notify the contracting parties about the damage or destruction of the material. Additionally, the software developer and the user have to be informed if such an event is likely to occur. Furthermore, the escrow agent has to inform the parties about activities of third parties which are likely to affect the deposited material, e.g. problems with external companies that are responsible for the maintenance or security of the storage environment.[85]

4.2.3 Alignment of the contracts pertaining to the software contracting framework

For the purpose of the establishment of a holistic Software Escrow solution, it has to be ensured that already the software licensing contract and also a possible maintenance contract are aware of the escrow agreement.[86] The latter agreements will regularly have to be qualified as continuous obligations, which

[77] *NCC Group Escrow Germany,* www.nccgroup.com/media/107015/sl_gmbh__ger__june_2012.pdf (22.11.2012) 4.1.1.

[78] *Sheffield/Leeven*, Software-Quellcodehinterlegung, CR 1995, 306 (307).

[79] TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 149 (151).

[80] *Bömer, „*Hinterlegung" von Software, NJW 1998, 3321 (3323).

[81] TIMBUS Deliverable 4.4: Digital Preservation & Legal Issues – Overview, 149 (150 f).

[82] *NCC Group Escrow Germany,* www.nccgroup.com/media/107015/sl_gmbh__ger__june_2012.pdf (22.11.2012) 4.1.1.

[83] *Wolner* in *Hausmaninger/Petsche/Vartian* 358 f.

[84] *NCC Group Escrow Germany,* www.nccgroup.com/media/107015/sl_gmbh__ger__june_2012.pdf (29.11.2012) 14.8.

[85] *Soler Matutes/Piattini Velthuis*, Manual de Gestión y Contratación Informática (2006) 894.

[86] *Kast/Schneider/Siegel*, Software Escrow, K&R 2006, 446 (449); *Redeker*, IT-Recht Rz 592.

might be terminated for reasons stemming from the escrow agent's sphere that are not contractually manageable, e.g. the agent's insolvency. It is thus vital for safeguarding the goal of ensuring the on-going maintainability of the software, to have a lasting contractual mean – in that case, provisions on Software Escrow in the software licensing contract – at disposal, regardless of what the current legal state of the Software Escrow agreement might be. So, the software licensing agreement ideally already has to contain a rudimentary – but nonetheless legally enforceable – preliminary Software Escrow agreement. The latter contractual measure especially has to entail a solution for the case of shortcomings of the first escrow agent or the termination of the escrow agreement by the latter agent for any reason. The software user should be given a position that allows him to legally enforce the conclusion of an escrow agreement with another agent. A concrete timeframe has to be set for the execution of all of the escrow agreements following the conclusion of the software licensing contract.[87] Another important point that needs to be addressed in the software licensing contract is the concrete scope of the IPRs – regularly copyright – to the source code and adjacent materials that the software user will be given in case one of the events triggering the release of the deposited materials is realized.

4.2.4 Selection of the material to be deposited

Having certain scenarios in mind, one would assume that the concrete selection of materials contemplated for an escrow deposit could vary depending on the amount of knowledge on the software existent in the software user's enterprise. In case of major involvement of the software user in the production process of a software solution handcrafted for the specific needs of this user for instance, knowledge on the concrete functionality and design of the software would be readily available in the form of the know-how of the user's employees involved in that process. This situation may however alter over time, as these employees could leave the user's enterprise and thus their know-how on the software development process would be lost. Therefore, for the purposes of the holistic Software Escrow solution contemplated in this chapter, the list of software development project artefacts is an absolute one. It consists of all the materials necessary to enable any software developer (without prior knowledge of the concrete software development project) with average coding skills to carry out maintenance services on the user's software in a reasonable timeframe.

Of course, the list contemplated for the purposes of this chapter is a very high level one in the sense that it may apply to any software, irrespective of the concrete programming language and development environment used. This list should be included in the software license contract that should in turn already contain the rough outline of the Software Escrow agreement to be concluded. In course of the Software Escrow agreement drafting phase, the items listed in the high level enumeration in the abovementioned software license contract should be broken down into more specific categories and specify the concrete software development artefacts that need to be deposited in order to ensure the maintainability of the deposited software upon redeployment.

[87] *Karger*, Software-Hinterlegungsverträge, in *Kilian/Heussen* (ed), Computerrechts-Handbuch[29] (2011) Rz 86.

The form in which the deposit of the information represented in the single deposit object enumerated in the Software Escrow agreement's listing is carried out will largely depend on the concrete software. If, for example, the software uses only standard compilers then the sole information on the type and version number might suffice, whereas if more specific compilers are used, the deposit of the compiler software might be warranted.

From a technical point of view there are different groups and artefacts that belong in an escrow deposit. All of the deposited artefacts have to be checked and evaluated, which is a time-consuming and therefore costly task. Thus, besides choosing the most essential parts of software, it is important to decide on a selection of artefacts that have to be deposited. A comprehensive summary of reasonable artefacts can be found in [49]. A shorter list, based upon the objects mentioned there and focusing on maintainability, leaving aside functionality, follows:

- Interfaces – Export interfaces can be checked for their implementation. This, however, overlaps more with a functionality test than the maintainability we are focusing on in this work.
- Intellectual property – Especially licenses for e.g. libraries used in the software or compilers are interesting for maintainability.
- Documentation - [49] discernes system documentation and end user documentation. The latter one only contains different functionality descriptions which are not in our focus. However, regarding artefacts concerned with system documentation, system architecture, and a general documentation of the software at hand, subsuming other artefacts like a maintenance guide or component descriptions, will be of interest.
- Test environment – All artefacts important for testing the software have to be included, like test infrastructure, test cases, test data, test script, or test reports.
- Design environment – Especially the design models are important for maintainability.
- Build environment – Compilers, runtime environments, configurations for the build like compile scripts or third party libraries are of interest.
- Applications – The most fundamental part of a software, the source code, as well as data sources, like a data base, or other binaries are important for maintainability.

4.2.5 Rights to the source code and adjacent materials

To fully perform his tasks during the execution phase, the escrow agent needs the right to demand regularly the newest version of the source code from the software producer and verify the deposited materials to assure the quality of the program.[88] Especially for long-term agreements the parties may also find it useful to entitle the escrow agent foresightfully to migrate the deposited materials from a then out-of-date storage device to a more current medium.

[88] Cf. *Schneide*r, Hinterlegung von Software, Escrow, in *Schneider/von Westphalen* (ed), Software-Erstellungsverträge (2006) Chap V Rz 81.

The software developer usually has no right to demand back the deposited original version of the source code from the escrow agent during the execution phase. However, this does not exclude the possibility to establish contractually an additional right to a copy of the deposited materials in case the software producer suffers a data loss and has no sufficient backup himself.

The consequences of the release of the escrowed materials regarding the rights to the source code have to be stated in the escrow agreement. Normally software licenses do not provide more than a right of use and software support to the client; this contractually established principle is not affected by the release of the escrow materials. Thus, the original right holder usually remains owner of all commercial rights of exploitation to the code and is protected in this position by certain obligations of the software client (see below). The obligation of the escrow agent to maintain confidentiality regarding the source code continues after the release of the materials.[89]

It is of utmost importance to clarify the client's rights regarding the software and its source code from the moment of the escrow release on. To protect the investment of the customer in the program and therefore ensure his possibility to continue its use without difficulties, he needs more than just the simple right of use. In German copyright law § 69d (1) UrhG[90] provides the software user with some basic rights regarding using, bug-fixing and making backup copies of the software; however, these limited regulations are normally insufficient to ensure the continuous usability of the software without any help of the producer and original right holder.[91]

The scope of the rights transferred together with the source code can depend on the actual trigger event for the release of the source code. In the case of insolvency of the software provider an extensive transfer of rights to the customer seems more adequate than in the trigger event of failed bug-fixing by the software provider.[92] In the latter situation the transfer of rights could be limited to the right to use the released source code exclusively to fix the program. But especially for triggering events such as the insolvency of the software provider the escrow agreement should contain unambiguous regulations that entitle the client to fix bugs, adapt the program to altered demands, develop it further, and perform the necessary maintenance tasks. In addition, the client should receive the right to copy new versions of the program for his own use.[93]

It is not necessary to concede more rights to the customer than the abovementioned, though. The purpose of an escrow agreement is ensuring the continuous usability of the software in case the producer is unwilling or unable to provide the required service. Therefore the software rights of the customer after the release of the source code do not need to include rights of any further distribution of the program or any commercial use that exceeds the limitations of the original software contract.[94] Besides, the rights of the user should be

[89] Cf. draft of an escrow agreement in *Jaburek*, Handbuch der EDV-Verträge^3 (2003) Vol 2, 296 f.

[90] Gesetz über Urheberrecht und verwandte Schutzrechte, 09.09.1965, BGBl. I, 1273.

[91] Cf. *Auer-Reinsdorff/Kast* in *Auer-Reinsdorff/Conrad* § 10 Rz 83.

[92] Cf. *Auer-Reinsdorff/Kast* in *Auer-Reinsdorff/Conrad* § 10 Rz 76.

[93] *Marly*, Praxishandbuch Softwarerecht^5 (2009) Rz 1782.

[94] *Marly*, Softwarerecht Rz 1782.

adapted to the needs of the individual case – depending on the software license at hand – especially regarding temporal and territorial limitations.[95]

If not provided otherwise by the contract, the source code itself and related materials usually have to remain strictly confidential in respect of third parties. The customer has to protect this confidentiality, e.g. by revealing these details only to employees who actually need them to fulfil their tasks and are also bound to secrecy as far as legally possible after the ending of their contract of employment. If it becomes necessary to disclose the source code to an external service provider that performs software maintenance tasks, the escrow agreement should oblige the customer to commit such providers to absolute non-disclosure.[96]

In case the customer does not comply with his obligations regarding confidentiality or other limitations of his rights to the software and its source code, his misbehaviour may result in compensation claims of the original right holder. To fully perform his tasks during the execution phase, the escrow agent needs the right to demand regularly the newest version of the source code from the software producer and verify the deposited materials to assure the capability of the program.[97] Especially for long-term agreements the parties may also find it useful to entitle the escrow agent for the future to migrate the deposited materials from a then out-of-date storage device to a more current medium.

4.3 Phase II – Execution

4.3.1 Overview

The execution phase will regularly stretch over a long time span. The execution of the escrow agreement entails the initial deposit process regarding the software development artefacts, the safe storage of said materials and the on-going repetition of the latter steps in case new software versions are implemented in the enterprise of the user. Each deposit process in turn consists of the transfer of the software development artefacts to the escrow agent, the verification of these artefacts and also possible corrections or even enhancements of the artefacts in case they do not pass the quality requirements set by the parties in the escrow agreement. An optimum execution phase serves two main purposes. It ensures that there is always the latest version of the source code plus adjacent materials of the software implemented in the enterprise of the user in storage at the escrow agent. Above that, it safeguards the quality of these deposited software development artefacts to such a degree that any software developer with average coding skills will be able to carry out maintenance services on the user's software in a reasonable timeframe.

4.3.2 Deposit procedure

The software development artefacts that have been selected for deposition at the facilities of the escrow agent (hereinafter referred to as the *deposit materials*) will have to be submitted to the escrow agent by the software development company. Those materials will then have to be verified by the escrow agent. In case

[95] Cf. *Auer-Reinsdorff/Kast* in *Auer-Reinsdorff/Conrad* § 10 Rz 87.

[96] Cf. draft in *Jaburek*, EDV-Verträge Vol 2, 296.

[97] Cf. *Schneide*r in *Schneider/von Westphalen*, Chap V Rz 81.

the deposit meets the thresholds set for the deposit materials, i.e. the materials pass verification, they will be submitted to safe storage. If, however, the verification indicates that the materials submitted are deficient in the sense that on-going maintenance of the software implemented in the enterprise of the software user cannot be ensured with these materials, then the software development company will have to resolve the problems on the basis of the verification report. This procedure will have to be repeated for any deposit until the criteria for a successful deposit are met. Only then may the submitted deposit materials be sent to safe storage. In the following sections, the verification procedures existent in the Software Escrow market and one crucial point, namely the definition of the thresholds for a successful deposit, will be outlined.

Furthermore, in course of the lifecycle of an escrow agreement, the whole deposit process will regularly have to be repeated several times. It has to be carried out for the initial as well as for any subsequent deposit. The latter deposits might be triggered by bug fixes or new versions of the application implemented by the development company in the software user’s enterprise. To ensure that such bug-fixes, updates or upgrades correlate with the deposit materials, and thus, the latest software version implemented in the software user’s enterprise may be created on the basis of the deposit materials, a carefully designed network of information and notification obligations must be put in place. The procedures necessary for that will be dealt with in Section 4.3.2.4.

4.3.2.1 Submission deadlines for materials

Besides a well-designed network of information-/notification obligations and the definition of the thresholds for a successful deposit, from a legal point of view there is another necessary contractual precaution to be taken in order to make an escrow agreement work in practice. As it is of the utmost importance to ensure that the software development artefacts are in fact submitted (and resubmitted in case the verification report suggests to do so) to the escrow agent for verification, the Software Escrow agreement must foresee a deadline for the initial and any subsequent updated software deposit. In that regard, some model contract clauses suggest durations of seven days[98] or two weeks[99] following the completion of the escrow agreement for the initial submission of the materials to be deposited. According to the latter model contracts, the obligation to submit updated materials for deposit is either created upon the implementation of the new software version in the software user’s enterprise and/or at the latest once a year upon a certain date.[100] Depending on the negotiation between the software user and developer, it might moreover be advisable to introduce penalties for the developer’s failing to meet these deadlines. As an example, the user might deduct a portion of the software’s licence fee as stipulated damages in case the software development company does not comply with its obligation to deposit its (updated) software development artefacts.

[98] *Jaburek*, EDV-Verträge Vol 2, 290.

[99] *Wolner* in *Hausmaninger/Petsche/Vartian* 358.

[100] Cf. *Jaburek*, EDV-Verträge Vol 2, 291; *Wolner* in *Hausmaninger/Petsche/Vartian* 358.

4.3.2.2 Existing verification procedures

There is – up until now and to the authors' best knowledge – only one reference document for Software Escrow verification processes, the CEN's[101] ESCROWGUIDE. It lists three types of different verification levels (standard, full and bespoke) describing the thoroughness of the verification methods applied to the deposit materials.[102] Besides the latter definitions, Software Escrows, depending on the degree of verification procedures in place, are being referred to as *active* and *passive* escrows.[103] The main difference between these two terms is that in case of the passive escrow, the deposit materials are not subjected to any technical verification. The active escrow, on the contrary, entails a technical verification and moreover provides an administrative process making sure that the deposit materials are frequently updated.

For the purposes of this section, however, the terminology and definitions provided by the CEN's ESCROWGUIDE shall be used. The *standard verification* ensures that the deposit materials submitted to the escrow agent are accessible, i.e. neither compressed nor encrypted, but readable and virus-free.[104] Although it also entails a rough check of a random source code sample, the main focus of the standard verification lies on ensuring the accessibility of the materials. Pursuant to the definition of the *full verification*, such a procedure will guarantee that the fully functional software implemented in the software user's enterprise can be built out of the deposited source code files.[105] The *bespoken verification* can be seen as a tailor-made solution, outside the scope of the standard- and full verification procedures, to satisfy the specific wishes and needs of the parties to the escrow agreement. For example, the software user might want to have the user documentation supplied to him checked for usability as well or he might want the concrete software evaluated against software quality standards like SQuaRE[106].

4.3.2.3 Thresholds for a successful deposit

The definition of the threshold for a successful deposit is one of the rather difficult points to be tackled in course of drafting[107] a Software Escrow agreement because up until now there are no universally accepted, accurate standards for the measuring of the quality of software and its source code. It is nevertheless of utmost importance for the software user to obtain a guarantee that the materials deposited at the facilities of the escrow agent are of a certain quality because otherwise the whole purpose of setting-up a Software Escrow would be alienated. Since no absolute figures regarding software quality may be included in the contract, the most viable contractual solution – especially from the software user's point of view – is to align

[101] I.e. *Comité Européen de Normalisation*.

[102] *CEN*, ESCROWGUIDE - Source Code Escrow - Guidelines for Acquirers, Developers, Escrow Agents and Quality Assessors, CWA 13620-4:1999 E, Part 4: A guide to providing a reliable escrow service 17.1 et seq.

[103] E.g. *Stekhoven*, Active Software Escrow's Usefulness for Companies Embracing COBIT 5, COBIT Focus 2012, 4.

[104] *CEN*, ESCROWGUIDE - Source Code Escrow - Guidelines for Acquirers, Developers, Escrow Agents and Quality Assessors, CWA 13620-4:1999 E, Part 4: A guide to providing a reliable escrow service 17.2.

[105] *CEN*, ESCROWGUIDE - Source Code Escrow - Guidelines for Acquirers, Developers, Escrow Agents and Quality Assessors, CWA 13620-4:1999 E, Part 4: A guide to providing a reliable escrow service 17.3.

[106] *ISO/IEC*: ISO/IEC 25000:2005 – Software product Quality Requirements and Evaluation (SQuaRE).

[107] The drafting of the contract takes place in the planning phase, see Section 4.2.

the threshold for a successful deposit with the primary purpose of the escrow agreement, i.e. the safeguarding of the maintainability of the software solution implemented in the software user's enterprise. A contractual clause could be designed as follows:

"The materials submitted to ESCROW AGENT by SOFTWARE DEVELOPMENT COMPANY are to be regarded as being viable for submission into the permanent storage facilities of ESCROW AGENT, if the verification report created by ESCROW AGENT concludes that: Any software developer without having any prior knowledge of the SOFTWARE created by SOFTWARE DEVELOPMENT COMPANY and in possession of average software developing skills may perform OPERATIONS X+Y on the basis of the abovementioned materials given a reasonable timeframe."

The extent of the operations that a software developer will have to undertake on the software in the future will depend on the person of the software user. If that software user is also the end user then those operations will regularly only entail bug-fixing and keeping the software operable. On the other hand, if the Software Escrow agreement is concluded between an agent, a single software developer or a software company focussing solely on development, and a software distribution company (the "software user", in this case), the latter company might also be in need of being able to e.g. add a further functionality to the existing software or create previously unforeseen interfaces to some other software. Moreover, to minimize the risk of litigation arising out of the generality of the clause, it might be advisable to provide more detail on the two elements of the programmer in possession of *average software developing skills* and the *reasonable timeframe* for the execution of maintenance tasks in the concrete Software Escrow agreement.

Of course, this contractual solution is tailored to the software user's expectations regarding the source code. In case a trigger event occurs, source code and adjacent materials with the inherent quality of enabling any other software developer to perform the essential operations will help loosen the user's dependency on the software development company. Deposit materials of sufficient quality will put the user in a position that allows him to contract any other software maintenance company on the market with the on-going maintenance of its software. Furthermore, the generic solution for quality criteria contemplated above carries the advantage for the company of the software user that its managers engaged in concluding the contract with the software development company will be able to participate more actively in the contract drafting negotiations, regardless of their degree of knowledge in the specific domain. The usage of such a generic clause is moreover utile for its ability of being applied to every software solution regardless of its concrete functionality and environment (e.g. operation system and hardware).

4.3.2.4 Information-/notification obligations

A carefully designed network of information and notification obligations serves the goal of ensuring that the materials deposited are always of a quality and in a state – regarding their version number – which enables the software user to create the latest software version implemented in the software user's enterprise upon their release. Such an information structure may only entail the mandatory notification of the software user upon the receipt of the (updated) deposit materials on part of the escrow agent. However – as demonstrated above in course of the case of *Telecom New Zealand* v *Aldous*[108] –such a basic level of

[108] See section 4.1.4.4

communication is not enough to ensure an optimum execution phase. Another useful step would be to introduce an obligation for the software development company to notify both the user and the escrow agent in case the software implemented in the user's enterprise has effectively been updated, as such a step might be carried out without the user's knowledge, which was obviously the case in *Telecom New Zealand* v *Aldous*.

Such an information process could be further aided and simplified by contractually foreseeing a procedure that obliges the contracting parties to regularly build check-sums and carry out automated correlation-checks regarding the software (i.e. binaries) implemented in the enterprise of the software user and the corresponding materials (i.e. corresponding binaries) deposited at the escrow agent's. As soon as a mismatch of the control parameters is established, all contracting parties will be notified automatically. The latter mismatch will moreover trigger the start of a specified timeframe for the software developer to deposit the software development artefacts corresponding to the updated software at the escrow agent's.

In light of the points made above, it may be concluded that the more comprehensive such information-/notification obligations and the closer the parties in the Software Escrow collaborate in that regard, the better the chances for the receipt of useful deposit materials upon the occurrence of an event triggering their release. Proximity in a geographical sense, i.e. between the offices of the contracting parties, might further aid the goal of efficient cooperation and communication. The latter point should thus influence the choice of a conveniently located escrow agent, presupposing, of course, that the other criteria set out for the selection of an escrow agent in Section 4.2.2 above are all met.

4.3.3 Verification - Technical Framework

In order to evaluate software we developed a technical framework that contains an extendable evaluation part with which various measurements can be conducted. These measurements allow assessing the quality of software artefacts with focus on Software Escrow specific concerns. The objective is to evaluate software artefacts that are delivered to the escrow agent holistically in either an automated, semi-automated or manual way in order to support a review and speed up the process of quality evaluation at the escrow agent's side. The goal is to automate as many measurements as possible and reasonable. However some of them will still need partial or complete input from a human reviewer. To achieve this holistic approach of evaluating all artefacts we decided to implement a framework consisting of single plugins that calculate well-defined measurements, which gives us the opportunity to merge specific results if needed (e.g. show the relation between complexity and lines of code). This framework builds on the design of Continuous Integration (CI) tools. It can be seen as an enhancement to a CI-tool, focusing merely on the quality aspects and less on build automation. Thus our focus is on an architecture extendable by plugins, designed for the Software Escrow purpose. If a specific measurement is needed for a software under test, the framework offers interfaces for creating new plugins. The outcome of the evaluation process is the identification of classes that are most interesting for a manual review. We do not argue that the framework will substitute a manual review and automate the process of quality evaluation. It can merely be a first, rough assessment of the artefacts delivered to the escrow agent, with the goal of sorting and highlighting the classes that possibly seem to be most problematic.

An overview of the theoretical framework, its inputs and outputs can be found in the following Section 4.3.3.1. The artefacts used as input in the framework can be found above in Section 4.2.4 as they are part of the Planning Phase. Section 4.3.3.2 describes the plugins suggested for a complete framework. The reports produced by the framework will be described in Section 4.3.3.3.

An implementation of a prototype as well as the plugins developed for it can be found later on in Section 4.6. Experiments done with different plugins to evaluate their applicability can be found in Sections 4.6.2.1 and 4.6.2.2.

4.3.3.1 Structure

The Software Escrow Framework illustrated in Figure 34 has a modular structure, consisting of input, processor subroutines, and different output formats. A container with all software relevant artefacts as well as supporting information (metadata) and requirements that need to be considered is used as input to the framework. The data is analysed and divided into the artefacts needed for evaluation. Different plugins are triggered, which send their results to a reporting tool where they are formatted into human readable ranking lists. These lists support the decision of a human reviewer on which classes have to be looked at more rigorously.

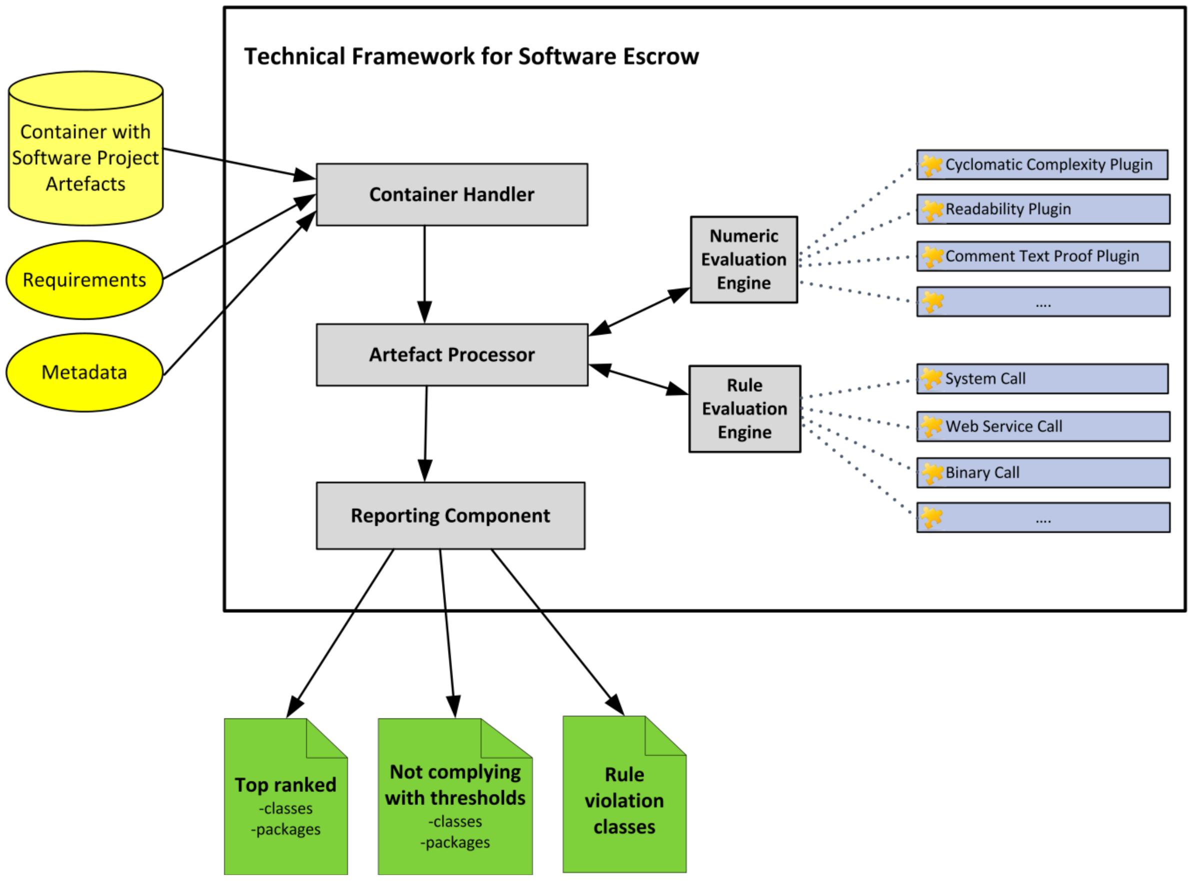

Technical Framework for Software Escrow
Container with Software Project Artefacts
Requirements
Metadata
Container Handler
Artefact Processor
Reporting Component
Numeric Evaluation Engine
Rule Evaluation Engine
Cyclomatic Complexity Plugin
Readability Plugin
Comment Text Proof Plugin
....
System Call
Web Service Call
Binary Call
....
Top ranked -classes -packages
Not complying with thresholds -classes -packages
Rule violation classes

Figure 34: Overview of Technical Software Escrow Framework

The input is a container consisting of different **software artefacts**, which can be source code, documentation, test cases, environments, licenses, or virtual machines (cf. Section 4.2.4). The **Metadata** item has to be delivered with the input artefacts because it contains information on the container and its content. For instance, it is important to know which compiler version was used if it is not provided already in the container. **Requirements** on the other hand contain information on what has to be tested, which can be derived from the escrow contract and the agreements made in there. The main part of the Software Escrow Framework, as depicted in Figure 34, is the **Artefact Analyser**. There, the container first gets parsed and segmented in the **container handler**, which is also responsible for the inspection of the completeness of artefacts (all artefacts agreed upon in the contract need to be put into escrow). The next step, the **Artefact Processor**, processes single artefacts that are sent to one of the two existing test engines: a **numeric evaluation**, which handles artefacts that can be used to calculate statistic measurements like Lines of Code or Cyclomatic Complexity, and a **rule based evaluation**, which checks for occurrences of rule violations and returns the location of the violations. Each measurement as well as the checks are designed as plugins. Their use is specified in the requirements. The plugins of both evaluation models can be run automated, semi-automated, or manually, depending on their requirements and how they evaluate artefacts. Automated evaluations are calculations only requiring artefacts, whereas semi-automated evaluations need manual evaluations from a reviewer for generating the final output. Completely manual evaluations are done solely by the reviewer, the framework only suggests the check and prompts for the results. The results of the evaluations are reported back to the Artefact Processor which can call the engines again if necessary (e.g. the result of a measurement is needed for calculating a second measurement). If all necessary evaluations are done, the Artefact Processor forwards all automated and manual results to the **Reporting Tool** (cf. Section 4.3.3.3). There, the resulting data gets processed and according to the requirements a report is generated.

4.3.3.2 Plugins

The plugins are the centrepiece of the technical software framework. The groups of different plugin families roughly describe the domain they belong to:

- Metric measurements
- Test environment
- Source code comments
- Specification verification
- External references
- Build environment
- System documentation
- Data sources
- Code obfuscation

The list of presented modules cannot be exhaustive but gives an overview of a solid base from which common plugins can be further developed. All groups are of the type *numeric evaluation*, except for the one described in Section 4.3.3.2.5 – “External references”, which is a *rule-based evaluation*.

4.3.3.2.1 Metric measurements

The first group of plugins contains statistical measurements of the source code which build the basis for subsequent measurements. These measurements are based upon physical source code attributes, like the length or how the source is composed. They are used to give a first rough overview of the code’s composition by giving it a value or highlighting code parts that need attention. This group includes for example:

- *Code lines plugin* (see Section 4.5.1) – A line number calculation. It takes in source code and delivers the number of its source code lines. Suitable for: methods, classes, files.
- *Complexity plugin* (see Section 4.5.1) – Calculates the complexity of source code. Takes source code as input and delivers a number representing the complexity. Suitable for: methods, classes, files, packages, the entire project.

According to [57] there are several important quality indicators regarding important aspects of maintainability: analysability, modifiability, verifiability and compatibility. A few examples are:

- *Duplicated Code plugin* – Highlight code areas where a block of at least 40 consecutive source code lines occur several times in the code. Suitable for: methods, classes, files, packages, project.
- *God object plugin* – Find objects like files, classes or methods whose scope is too huge. Suitable for: classes, files, methods.
- *Class incest plugin* – The superclass must not have direct knowledge of its direct or indirect subclasses or sub-interfaces. Suitable for: classes.
- *Prenatal communication plugin* – A virtual method of class *K* gets called directly from the constructor, regardless if it is overridden in subclasses or implemented in *K*. Suitable for: classes.

4.3.3.2.2 Test environment

The plugins of this group are usually standard test in CI tools. Our Software Escrow Framework also evaluates the testing environment which will be needed for maintainability when the escrowed material gets redeployed. Plugins of this group are for instance:

- *Code Coverage plugin* (see Section 4.5.1) – Calculates the degree of source code that was called by a test case. For this it needs source code/binaries and test cases as input and produces a numerical percentage as output, where higher values mean more code tested. Suitable for: in particular for the entire project, also for classes, files, and packages.
- *Unit testing plugin (see for instance* [58]*)*: Checks if all unit tests terminate with a positive test result. Inputs are the test cases, the source code or its binary, and probably test scripts to trigger the tests. Output is textual, stating the correct execution of all tests or the ones that did not pass. Suitable for: classes, files, packages and the entire project.

4.3.3.2.3 Source code comments

Comments within the source code are crucial for the understanding of the source code. Thus they need to be comprehensible for a developer and in case of Software Escrow most likely for a developer who is new to the software. To guarantee understandability, analysis of this (partly) natural language text has to be done taking into account different aspects, including quality of the text (how readable is the text?) as well as quantity (how much comments are there?). The plugins related to this group are for example:

- *Readability plugin* (see Section 4.5.3.3) – Indicates how easy text can be read. Input is of textual form, output a numeric value. Suitable for: comments or other textual writing.
- *Readability:Comment-words plugin* – Sets the readability value in relation to the number of comment words. Input are numerical values, output is numerical as well. Suitable for: methods, classes, files.

- *Language detection plugin* (see Section 4.5.4) – Detects the language and checks for consistency or foreign languages. Input is of textual form, output is/are the detected language/s. Suitable for: comments or other textual writing.
- *Text proofing plugin* (see Section 4.5.4) – Checks the word order in sentences. Input is of textual form, output is a list of sentences with wrong word order and their locations. Suitable for: comments or other textual writing.
- *Comment word plugin* (see Section 4.5.4) – Calculates the number of comment words in a source file. Input is of textual form (comments), output is numerical (number of comment words) and preferably high. Suitable for: comments, respectively ratio calculations like code lines to comment words.
- *Comment-words:Code-lines plugin* - Sets the number of comment words in relation to code lines. Input are number of comment words and LOC values, output is numerical as well. Suitable for: methods, classes, files.
- *Complexity:Code-lines plugin* – Sets the complexity in relation to the code lines. Input are numerical values, output is numerical as well. Suitable for: methods, classes, files.

4.3.3.2.4 Specification verification

Software specifications complement the documentation of the escrowed software. To check if the specifications are in line with the actual implementation and thus proof that they belong together, the plugins of this group provide verification and validation of the appropriate artefacts. This includes for example the following plugins:

- *Architecture model plugin*: Verifies the correct implementation of the software architecture into the final software. Input is a document, containing architecture details in graphical form (like UML or written in an Architecture Description Language (ADL) like the Wright ADL[109], see [59]) and the source code of the software. Output is a list containing the differences between both types. Suitable for: classes, files, packages, the entire project.
- *Architecture description plugin* (see Section 4.5.3.2): Checks if the textual description of the architecture complies with the implementation. Because the structure of the implementation can be tested with the *Architecture model plugin*, only the textual components of both architecture and source code will be examined here. Thus the input consists of the (textual) architecture document and the source code comments, the output again will be a list, containing the differences between both types. Suitable for: classes (if the architecture document is partitioned into equivalent sections), files, packages, the entire project.

[109] http://www.cs.cmu.edu/~able/wright/

4.3.3.2.5 External references

References to external binaries, like libraries or Web Services, need special attention when it comes to Software Escrow, especially regarding hiding source code in libraries. It is important to check that all external references are available, or in case of Web Services to make sure that they are part of the container put into escrow. This module provides rule base evaluation as it checks for occurrences in the source code. These specific occurrences (see also Section 4.1.3.4) are checked by plugins like:

- *Non-standard libraries plugin* – Check for usage of external libraries that were used in the software. External libraries are libraries that have to be included separately (e.g. by an import statement in Java). Input consists of source code and libraries. Output is a list of occurrences where external libraries are called. Additionally the license used in the library can be automatically detected. Suitable for: entire project.

- *System calls plugin* – Check for system calls. Input consists of source code, output is a list of system call occurrences. Suitable for: classes.

- *Binary call plugin* – Check for binary calls, e.g. calling an executable file or also a script or batch file from within the source code, independent of the availability of the called binary. Because this call references another program or source code, it has to be checked separately. Input consists of source code and the binary or script called (if available, e.g. if the commands do not get executed from within the source code already), output is a list of occurrences in the source code. Suitable for: classes.

- *Web service call plugin* – Check for a Web service call. Input consists of source code, output is a list of Web service calls from within the source code. Suitable for: classes.

4.3.3.2.6 Build environment

The source code put into escrow also has to be compilable in order to guarantee its full functionality and compliability with the software used by the customer. Compiling the sources is also an important requirement for many other plugins. For example, the following plugins can be added to this group:

- *Compiler plugin* – Checks if all necessary compilers are available by compiling the software. Input is the source code of the entire project and its compilers as well as a compile script (e.g. ant build file or make file). Output is a runnable binary of the software that has to be verified against the customer's version of the software.
- *Operating systems plugin* – Tests the software on the operating systems it was designed for. Because the framework only runs on one system at a time, usage of virtual machines is recommended. Inputs are the compiled binaries of the software; output is a list of operating systems the tests failed on.
- *Runtime plugin* – If a specific runtime is necessary for the software, this plugin checks if it is part of the artefacts and if it works. Thus it needs the runtime as well as the compiled software as input to check against, and delivers a list of errors as output.

- *Licenses plugin* – Licenses for libraries or the build environment are probably needed for further development. Therefore this plugin checks if the needed licenses are part of the artefacts. Input are the licenses agreed on in the escrow contract. Output is a list of found licenses, which includes possible unknown licenses, and the classes they were found in.
- *Release plugin* – The software put into escrow has to be the same as the software run by the customer. Thus it has to be verified that both are equal. For this the sources at the escrow agent have to be built and compared to the binaries delivered to the customer. Input are the software sources and the release from the customer, output are the different files of both versions.

4.3.3.2.7 System documentation

Documentation of software is necessary to provide access to considerations and decisions made during the development in natural language. It is especially important if the software in question has to be maintained and possibly enhanced at an unknown time in the future by a programmer with unknown programming experience. Thus the documentation has to be easily readable and easily understandable. Plugins supporting these attributes can be for example:

- *Readability plugin* (see Section 4.5.3.3) – Indicates how easy text can be read. Input is the documentation in textual form, output a numeric value. Suitable for: textual writing.
- *Language detection plugin* (see Section 4.5.4) – Detects the language and checks for consistency or foreign languages. Input is the documentation in textual form, output is/are the detected language/s. Suitable for: textual writing.
- *Word order plugin* (see Section 4.5.4) – Checks the word order in sentences. Input is of textual form, output is a list of sentences with wrong word order and their locations. Suitable for: textual writing.

4.3.3.2.8 Data sources

Data sources like databases can be a major part of software, e.g. library software depending on a book database. Thus storing a proper version of the database with the other escrow artefacts is important. Plugins of this group therefore have to test the data sources to verify that they will still be functional when the software and its sources are delivered to the customer. These plugins can be for example:

- *Data source verification plugin*: Verify the software's data sources like databases for their availability and check specific attributes, like schema or stored procedures. Input are the data sources, output is a list of incorrect data sources that need further attention. Suitable for: data sources.

4.3.3.2.9 Code obfuscation

As mentioned in Section 4.1.3.3, code obfuscation in Software Escrow is not recommended. If there is still functionality of the code obscured, the support of the plugins in this group is needed. Amongst them are for instance:

- *Code obfuscation plugin* (see Section 4.1.3.3): A plugin checking for code obfuscating techniques in the source code, like pattern matching for meaningless variable names. Commonly used obfuscation names like character concatenation (e.g. "aaa") can be blacklisted, while variable, method, or class names can also be checked for meaningful words with a dictionary. Input is only the source code of the software. The output contains a list with occurrences and which obfuscation method was used. Suitable for: the entire project, method, class, file, package.

4.3.3.3 Reports

The reports of the Escrow Framework are outputs generated for supporting a reviewer in deciding which artefacts need special attention. Thus the reports need to summarize the findings of the framework compactly. We suggest three different kinds of output:

1. *Top ranking*: A list of artefacts, e.g. classes or packages, which received the worst measurement values in its project. Each artefact listed is therefore a "top-ranked" candidate to contain parts that can lead to understanding problems or increased maintenance effort when the software has to be redeployed.
2. *Threshold compliance*: A list of artefacts, e.g. classes or packages, which exceed or fall below the thresholds defined in the agreement (e.g. the code coverage of test cases is below 70%). Again each artefact listed is a candidate to contain parts that can lead to understanding problems or increase maintenance effort when the software has to be redeployed.
3. *Rule violation*: A list of artefacts that violated a rule, e.g. a class containing a Web service call. The difference to the other reports is that with rule violation there is no ranking. Each class listed is an equally interesting candidate that contains one or more violations which can lead to understanding problems or increase maintenance effort in the future.

A combined measure, including different measurements, can be obtained by using utility analysis [60]. With this method essential measures can receive a higher weight than less important ones. Based on the result of this weighting, projects can be compared with each other more easily.

4.4 Phase III – Redeployment

4.4.1 Overview

The redeployment phase is the last phase in the lifecycle of a Software Escrow agreement. Its main task is to ensure the quick release of the deposit material to the software user in case a trigger event occurs. Its aim is to prevent the downtime of the crucial business process the Software Escrow agreement was signed for. In the planning phase and after the risk analysis all releasing events are stipulated in the escrow agreement. In case one of the trigger events occurs the escrow agent follows the release proceedings and examines the occurrence of the event. When the result is positive he hands over the material to the software user. As the escrow agreement contains the software user's right regarding the released materials it enables him to continue his business.

4.4.2 Release of the escrowed materials

The release of the deposit material depends on the occurrence of one of the contractually established trigger events. Different Software Escrow agreements can contain different trigger events depending on the interests of the contracting parties which must be covered. Therefore, it is necessary to determine under which circumstances the source code should be handed over. Besides a trigger event, which is in favour of the software user, some escrow agreements may contain specific contracting previsions which allow releasing the deposit materials to the software developer (under specific circumstances described below).

4.4.2.1 Release Events

The risk of insolvency of the software developer is one of the most important reasons for the software user to sign an escrow agreement. So the release/trigger event formulated in the Software Escrow agreement is the opening of bankruptcy proceedings over the assets of the software developer. It is important that in the Software Escrow agreement this trigger event is defined very precisely. Hence, it must be clear that the request for the opening of bankruptcy proceedings over the assets of the software developer is no trigger event itself. The release of the source code and the deposit material requires a prior decision of the competent insolvency court to open the bankruptcy proceedings. Another trigger event that should be established is the dismissal of the opening of bankruptcy proceedings due to lack of assets. In both cases it is crucial for the escrow agent that the escrow agreement specifies which documents must be submitted to demonstrate the occurrence of the trigger event. The escrow agent must know if an extract of the trade register meets the requirements or if the existence of the trigger event must be proved through a legally binding decision or even a notification of the software developer.[110] As the last mentioned condition requires the active participation of the software developer to enable the release of the source code and the deposit materials, it is not recommendable to include it. The software developer might not be interested in fulfilling this requirement, with the consequence that the software user might not get the materials or has to initiate time-consuming and costly legal proceedings against the developer.

When the software developer committed himself to provide computer documentations developed in the future, this contract clause is not insolvency-proof. According to § 91 gInsO it is not possible to acquire rights in objects which form a part of the assets involved in the insolvency proceedings after the insolvency proceedings have begun. However, a disposal under a condition precedent[111] regarding future things or rights shall be insolvency-proof in case the condition occurs after the opening of the insolvency proceeding, if the item in question has already been created before the opening[112]. Hence, rights to adapt already existing software can be assigned in an insolvency-proof way under a condition precedent, the trigger event.[113] At least in Germany it is important not to mention insolvency of the software developer explicitly as

[110] *Schneider*, Handbuch des EDV-Rechts(2009)^4 M Rz 122 f.

[111] A contractually determined event must take place before something else occurs.

[112] BGH IX ZR 162/04 MMR 2006, 386 = CR 2006, 151 = GRUR 2006, 435 = NJW 2006, 915.

[113] *Hoeren*, IT-Recht 10/2012, 323 f, http://www.uni-muenster.de/Jura.itm/hoeren/materialien/Skript/Skript_IT-Vertragsrecht_Oktober_2012.pdf (6.12.2012).

a release event because German Insolvency law[114] establishes it is not possible to limit the insolvency administrator’s right to choose whether to fulfil or to reject performance of the contract. Therefore, clauses which contain the explicit mentioning of insolvency of the software developer as a release event are not insolvency-proof. In case of an international contract, clauses which exclude the insolvency administrator’s right to option established in § 103 gInsO, would infringe the German public order principle determined in Art. 6 EGBGB.[115]

It is also possible that the software company is liquidated and no longer exists. The liquidation of the software developer’s company is a trigger event which must be defined as release event in the escrow agreement.

The escrow agreement should also contain the cessation of the software developer’s business or the part of his business which is connected with the software for which the escrow agreement is established as trigger event. A similar case is the assignment of intellectual property rights in the material to a third party which does not enter into the existing escrow agreement or agree a new escrow agreement with the user within a predefined period of time. In case the third party enters into the existing escrow agreement or offers to sign a new escrow agreement with the user, the refusal of the user to accept this offer of the third party within a certain period of time after being notified, shall not constitute a release event.[116]

The unjustified refusal of the software developer to maintain the software or to maintain the software through a third party as the fulfilment of an existing maintenance contract or as obligation of the software licence agreement is a release event. The releasing process requires the user to notify the software developer. If the software developer fails to remedy such notice of default within a reasonable time, the escrow agent has to fulfil his releasing obligation.[117]

The unjustified denial to adapt or change the software to provide the interoperability of the software with other software programs or a new hardware must be explicitly mentioned as releasing event, because the safeguard of the interoperability is a part of the license contract. This clause does not contain debugging because debugging is part of the maintenance contract. The refusal of the software developer to program new functions or to improve existing functions which is not included in the debugging obligations established by the maintenance contract, does not constitute a trigger event, in case the changes are not necessary to provide interoperability of the program.[118]

In case of the existence of a notarial certificated document which contains the agreement of the software developer to release the deposit materials to the software user, the escrow agent can hand over the material to the user.

---

[114] § 119 gInsO.

[115] See *Graef*, Insolvenz des Lizenzgebers und Wahlrecht des Insolvenzverwalters – Lösungsansätze aus der Praxis, ZUM 2006, 104 (105); *Abel*, Filmlizenzen in der Insolvenz des Lizenzgebers und Lizenznehmers, NZI 2003, 121 (128); *Straßer*, Gestaltung internationaler Film-/Fernsehlizenzverträge, ZUM 1999, 928 (933).

[116] See NCC, Single Licensee Software Escrow Agreement, http://www.nccgroup.com/Libraries/Escrow_Agreements_Added_Feb_2011/SL_Nov07.sflb.ashx (04.04.2012).

[117] *Karger* in *Kilian/Heussen* Rz 107 f.

[118] *Wolner* in *Hausmaninger/Petsche/Vartian* 360 ff.

Another trigger event could be the submission of a final judgement with regard to interlocutory injunction to discharge the consent of the software developer to release the deposit material to the user.[119] Decisions in arbitration proceedings should have the same effect.

As mentioned above, it is also possible that the Software Escrow agreement contains specific trigger events whose occurrence leads to the release of the source code and the deposit materials to the software developer. One of the trigger events can be the agreement of the software user to hand over the deposit material the software developer. This agreement must be documented in a notarial certified document which is sent to the escrow agent. The escrow agent has to handover the deposit material to the software developer in case a final judgement or a decision in arbitration proceedings determined this obligation. If the contracting parties have agreed that the software developer can use the deposit material as a backup in case of disaster, the escrow agent has the obligation to make a copy of the deposit material at the expense of the software developer and hand it over to him. The originals of the deposit material must remain with the escrow agent.

The termination of the Software Escrow agreement should not lead to the release of the deposit material to the software developer. In case of contract termination, the escrow agent has the obligation to destroy the deposit material.

In case the contracting parties decide a change of the escrow agent, the prior escrow agent has the obligation to deliver the deposit materials to the new escrow agent who is previously designated by software developer and software user.

4.4.2.2 Release process

It is very important that the Software Escrow contract contains the determined proceeding the escrow agent has to follow in case of the occurrence of one of the trigger events. Therefore, the documents which must be provided to prove the occurrence of the trigger event must be explicitly mentioned. The escrow agent has the obligation to examine them and to decide on base of the document presented if the handover should be carried out or must be denied.

The trigger event established for the handover of the deposit material to the software user can be distinguished into hard or soft criteria. The first category is called hard criteria because the facts can be proved by submitting certificate documents which if necessary can be examined by a notary. Soft criteria are technical issues or questions regarding the content of the software or contractual obligations. The soft criteria cannot be proved alone by the presentation of documents, because the occurrence of one of these soft criteria may not be documented in a certificate document.[120]

The software user has to inform the escrow agent about the occurrence of a releasing event. How this notification has to take place will be explained below. The notification must contain a substantiated description of the trigger events as well as detailed information about the continuing entitlement to use the

[119] *Böhmer*, „Hinterlegung“ von Software, NJW 1998, 3321 (3323).

[120] *Schneider*, EDV-Recht M Rz 119 ff.

software. The software user has to provide the necessary documentations to prove the occurrence of the trigger event[121].

In case of the opening of bankruptcy proceedings over the assets of the software developer or the dismissal of the opening of bankruptcy proceedings due to lack of assets, the software user has to submit the original or a certified copy of the enforceable decision of the applicable court.

When the liquidation of a business is decided and the liquidation resolution is registered in the commercial register, the software user has to present a certified extract from the Commercial Register.

The software user has to prove the cessation of business or part of his business which is connected with the software for which the escrow agreement is established and that the software developer was not able to furnish an equivalent replacement within the set period time.

The case of an unjustified refusal of the software developer to maintain the software can be proved by the software user by submitting a registered letter with acknowledgment of receipt of the user to the software developer with the substantiated request to fulfil the contractual obligations resulting from the software maintenance contract.

The user should present the same piece of evidence in case of an unjustified denying to adapt or change the software to provide the interoperability of the software with other software programs or a new hardware.

To prove the agreement of the software developer to release the deposit material to the user, the user has to provide the prior written approval of the software developer. For purpose of evidence it is advisable that the consent of the software developer is presented in a notarial certificated document.

In case of a final judgement with regard to interlocutory injunction to discharge the consent of the software developer to release the deposit material to the user, the user must submit this document to the escrow agent.

The notification of the trigger event must contain the signed confidentiality obligation of the software user regarding the content of the deposit material. In each case of violation of the confidentiality obligation the contractor is liable for, the contractor incurs a contractual penalty. This penalty is incurred regardless of whether damage is actually caused or not.[122]

The escrow agent is obliged to inform the software developer in case the user claims that the trigger event has occurred.[123]

The escrow agent is authorized to release the deposit material to the software developer in case the software developer does not contradict it immediately or at least within 14 workdays from the day a copy of the releasing request of the software user was transmitted to him.[124] The denial of the occurrence of the trigger event must take place in written form containing the following details:

- Substantiated description why the claimed release event did not take place in the opinion of the software developer.

[121] *Wolner* in *Hausmaninger/Petsche/Vartian* 361.

[122] See *Grapentin-Noerr*, Software-Lizenzvertrag (Kauf), in *Weise/Krauß (ed)*, Beck'sche Online-Formulare Vertragsrecht^{22} (2012) Fn 34.

[123] *Wolner* in *Hausmaninger/Petsche/Vartian* 361.

[124] *Grützmacher*, Hinterlegungsvereinbarung, in *Redeker (ed)*, Handbuch der IT-Verträge (2010) Rz 80.

- The software developer has to provide all adequate documents which are necessary to prove the non-existence of the trigger event.

When the escrow agent receives within the above mentioned timespan the statement of opposition of the software developer, he shall have the on-going obligation to store the deposit material. The escrow agent shall notify the software user immediately about the existence of the statement of opposition and submit it to the user. The escrow agent is not allowed to hand over the material to the software user. The release of the deposit material can only be carried out when an irreversible final arbitrament of an arbitration court is made, which indicates that the source code and the deposit materials have to be hand over to the software user. The release of the deposit material can also take place when the software developer subsequently agrees to it[125] In case of the release of the deposit material the escrow agent has to hand it over to the software user.

In case of a handover of the deposit material to the software developer in the event of the agreement of the software user, the escrow agent can release the materials without the obligation of giving prior notice to the software user.

If the software developer needs the deposit material as a backup, the escrow agent has to inform the software user of the fact that he will make copies of the deposit objects and hand these copies over to the software developer. As the originals of the material remains with the escrow agent the handover of the copies has no effect to the escrow agreement itself. Therefore, the announcement of hand over a copy to the software developer serves only to provide this information to the software user. As he has already agreed previously that the software developer can use the deposit material as a backup he has no possibility to deny the handover of the copies. The copies will be made at the expenses of the software developer.

4.4.3 Information-/notification obligations

The software user has the obligation to inform the escrow agent about the occurrence of any of the above established trigger events. The notification has to be given in a closed and sealed envelope either by registered letter with acknowledgement of receipt[126] or by hand delivery against receipt. Then the escrow agent has to fulfil his obligation to notify the software developer about the announcement of the software user that a trigger event has occurred. The notification is essential to give the software developer the possibility to dissent the handover of the deposit material. The proceeding and form is described above.

In general, any notification has to be in writing in order to prevent the occurrence of problems regarding the burden of proof. In urgent cases the parties should be informed orally in advance.

The escrow agent must inform the other contracting party, if one of the parties does not fulfil their contractual obligations concerning the payment of the fees for depositing and verifying. Non-payment should not lead to the end of the escrow agreement. Therefore, the other party must be able to intervene.

[125] *Wolner* in *Hausmaninger/Petsche/Vartian*, 361 f.

[126] See *Bömer*, NJW 1998, 3321 (3323).

The escrow agent is obliged to inform the software developer in case that the user claims that the trigger event has occurred.[127] The software developer must be able to respond within a reasonable period of time (please see above Section 4.4.2 for detailed information concerning the release of the escrowed material).
In case the escrow agent is obliged to make the deposited material available to third persons by a judicial or administrative decision, the agent must immediately inform the software developer as well as the user.[128]

4.5 Related work

During the discussion and agreement on the requirements for a software project different objectives are agreed upon. To fulfil some of the objectives measurements will be necessary. These quality measurements, respectively their values, shall help to agree upon certain minima/maxima or ranges which the software has to comply to. Different measurement approaches, including statistic measurements based on the attributes of a source code and rule-based measures checking if the source code adheres to certain rules, will be described in more detail below. This section concludes with the presentation of a broader approach, focusing on the maturity of software for reuse.

4.5.1 Statistic measurements

Statistic code metrics can be used as first reasonable and easily computable indicators to see if the software's underlying source code has unwanted defects, like too many lines of code that indicate a high software complexity.

Lines of Code (LOC) as the most prominent measurements counts the number of lines a certain file has. A related metric can be found in *method lines of code*, which can be used to find methods that are too long to be understandable. There are different measurement methods including or excluding empty lines or comment lines. Lines containing only a curly brace, like it is common after a method declaration, usually also count as coding line, although this can be subject to discussion. In this document, empty lines and comments do not count as code, so a LOC measure only includes lines interpreted by the compiler. The only exception is made with the **NCSS** (*non-commenting source statements*) measure, which is provided by JavaNCSS[129] and differs to LOC in the tests. Because different programming languages need a different amount of space to state the same algorithm, the lines of code measure is not ideal for comparing implementations in different languages. A programmer's skills also influence the number of lines he uses for an implementation. Thus, the comparison has to be done carefully and possibly only within similar environments.

According to [61], large files contain a high number of defects. A high LOC value can therefore be an indicator for low software quality. In [62] the *source line code* metric was one of the most interesting metrics

[127] *Wolner* in *Hausmaninger/Petsche/Vartian* 361.

[128] *Wolner* in *Hausmaninger/Petsche/Vartian* 359.

[129] The used implementation, provided by JavaNCSS, has an explanation what is counted and what not: http://www.kclee.de/clemens/java/javancss/#specification.

to evaluate quality and results were similar to those of the manual evaluation done by an expert, although it has to be mentioned that only one single expert was used for the evaluation.

Another commonly used measure to determine maintainability of source code is the **Cyclomatic Complexity** developed by Thomas J. McCabe in 1976 [63], based on the idea that humans can understand source code only until a certain amount of complexity of the code is reached. Instead of looking only at the syntactic elements in it, source code is seen as a directed graph with nodes and edges, nodes representing commands and edges representing direct connections between commands. With the following formula the complexity of graph g can be computed.

$$v(G) = e - n + 2p$$

e being the number edges, *n* the number of nodes, and *p* the number of connected components. According to McCabe a “reasonable, but not magical” [63] upper limit for the cyclomatic complexity is ten. When measuring source code with many if-clauses (like a switch statement), the counter increases by one for each statement. Therefore the upper bound has to be set carefully, depending on what kind of code has to be evaluated. For a metric similar to the cyclomatic complexity, see also Halstead’s software metrics in Section 4.5.3.1. These metrics include program volume or program difficulty and look at code from another point of view.

What can also be used as a statistic measurement is **Code Coverage**: It specifies how many lines of a source code are tested. For this an applicable tool usually highlights the lines/statements or parts that were executed during a test case in another colour than the ones that were not executed. The impact of code coverage is controversial, see for instance [64].

4.5.2 Rule-Based measurements

Rule-based measurements usually have a list of rules the code must not violate. Known tools building on this approach can be found in *Checkstyle*[130], *FindBugs*[131] or *PMD*[132]. As it will be shown later in Section 4.6.2, these tools can be useful to find source code passages that might contain bugs or are difficult to understand because they do not stick to common coding rules.

Checkstyle, a static code analysis[133] tool, is a Java code analyser built for automatically checking if source code adheres to given coding standards. With this, the use of coding guidelines like the Java Code Conventions can be tested. Checkstyle can check for self-defined or already implemented issues, like design problems, duplicate code, or bug patterns. Analysing documentation like Javadoc is also supported. Following the comparison of [65], Checkstyle targets coding conventions like naming, comments or format conventions. More focusing on bad practices [65] is PMD, which analyses source code to find behaviour that might lead to problems over time. It can find unused variables or empty catch blocks and helps to keep the

[130] http://checkstyle.sourceforge.net/

[131] http://findbugs.sourceforge.net/

[132] http://pmd.sourceforge.net/

[133] Static code analysis is analysing the code without actually running the build software.

code tidy. **FindBugs** on the other hand addresses, as its name suggests, potential bugs. The developers of FindBugs discriminate between **style checkers** (like the tools mentioned before) that examine code according to style rules and **bug checkers** on the other hand. The latter analyse code in order to find violations of specific correctness properties which may cause the program to misbehave at runtime [53]. Thus, they extend the checking of a compiler and are able to catch errors the compiler ignores.

4.5.3 Readability

Readability denotes how easily a text can be read and understood, which has to be distinguished from legibility, which measures the appearance of text, like the recognisability of characters. Focusing on software, two kinds of readability were found to be interesting: *source code readability*, which addresses machine readable text, and text in natural language, which subsumes source code documentation like comments as well as software documentation, like technical writings. A third method, which can be located between the two, evaluates what comments, respectively documentation, and source code have in common, e.g. if the comments contain an explanation for the following code or vice versa. Section 4.5.3.2 focuses on these approaches.

4.5.3.1 Source code readability

Maintainable software has many attributes it has to adhere to, one of them being an easily understandable source code. The programmer who writes the code has to make sure that his chain of thoughts can outlast the time the source code is kept in escrow. Besides natural language explanation in form of comments, which will be discussed in Section 4.5.3.3, the most appropriate way is to write source code that can be easily understood.

There have been many ideas on how to provide easily readable source code, that are targeting complexity or volume and do not mention readability explicitly. This section on the other hand deals with work that explicitly focuses on readability, sometimes relying on basic statistic code metrics itself.

The most exhaustive analysis was done by Buse and Weimer in [66]. Their approach sees complexity and readability as not being closely related, with complexity being essential and readability being enhanceable, as it can be addressed more easily. Moreover, in their previous work [67], they even propose that readability of code is not related to the size of the code fragment. Their automatic evaluation is based on a survey among 120 students with different experience evaluating 100 code snippets, showing that there exists a correlation between readability of code and conventional software quality metrics. With this they argue that, e.g. average line length is an important factor regarding the understanding of source code.

Because the provided readability tool was only built to evaluate short code snippets with a few statements, larger scale experiments were impractical to do. Nevertheless, the listing/ordering of code quality features that correlate with higher and lower readability can be interesting for further experiments.

Using the manually completed survey of [66], Posnett et al. [68] pursued another approach. They make use of the Halstead software metrics [69], including metrics like *Program Volume*, *Difficulty* and *Effort-To-Implement*. Contrary to the cyclomatic complexity proposed by McCabe (see also Section 4.5.1) these metrics are based on lexical measures, namely the number of total/unique operators (N_1/n_1) as well as the total/unique operands (N_2/n_2). Operators include e.g. methods, operands on the other side are e.g. method arguments. Metrics are simple equations, containing these objects:

- Volume: $V = N \log_2 n$, with N being the Program Length $N = N_1 + N_2$
- Difficulty: $D = \frac{n_1}{2} \frac{N_2}{n_2}$
- Effort: $E = DV$

Their results show that readability, as explained by Buse and Weimer, can be more easily measured by using Halstead's metrics, respectively their proposed model, which also includes entropy and the size of the snippet.

This model was observed to fit the data reasonably well and even outperform the model Buse proposed. It is worth mentioning that Halstead's work and its fundamentals have been criticized a few years after their publication [70] because of Halstead's misapplication of cognitive psychology results. Nevertheless, the proposed metrics are very useful for this application although they are not commonly used as can be seen in the low number of tools that support their calculation. Therefore implementing a tool to calculate Volume or Effort had to be made from scratch (cf. Section 4.6.2).

Supporting the use of the Halstead metrics, Coleman et al. [71] argue that Volume and Effort metrics are the best predictors of maintainability for their test data. They propose a maintainability metric, based on average Effort, average Volume, average extended Cyclomatic Complexity, average lines of code and percentage of comments. Similar to the Halstead metrics, the maintainability index MI is often not implemented by software quality tools. The only implementation found was in Microsoft's Visual Studio[134].

4.5.3.2 Comment and code combination

Programming rules are instructions for programmers to develop homogenous code. Code defects can be introduced more easily if not all involved coders adhere to the guidelines so an automatic evaluation of how well these rules are adopted can help to develop higher quality code. Research has been done to mitigate these risks and find similarities between comments and code. One example for this research is the work of [72], which extracts implicit programming rules without prior knowledge about the software. This is done by using *frequent itemset mining*, with which *association rules* are generated subsequently. With these methods, violations between the extracted rules and the source code can be detected.

In [73], formal specifications of what code is doing, is in the focus of research. Formal specifications are necessary to find defects and to formally describe the program as well as its behaviour afterwards. These specifications can be represented as a finite state machine. The specification mining method they use analyses actual program behaviour and can construct formal specifications of correct program behaviour from that. Their *specification miner* estimates the quality of code fragments, lifts those quality judgments to

[134] According to http://www.projectcodemeter.com/cost_estimation/help/GL_maintainability.htm VS seems to be using a slightly different formula for the calculation.

traces, which are examples of program behaviour like terms of sequences of function calls, and finally weights each trace. The produced output is a set of candidate specifications which have to be evaluated by a human. With their approach they can improve the performance of existing miners, decreasing the rate of false positives.

Focusing on traceability links, which can be described as links between different software artefacts, is a publication about recovering these links in code as well as in free text documents like documentation [74]. One example of use of these links is maintenance, which brings this method also into the focus of Software Escrow. The work mainly analyses mnemonics for identifiers. For this it uses the extracted source code component identifiers as a query to retrieve documents that are relevant for this component. An important assumption is that programmers use meaningful names. Both applied Information Retrieval (IR) models (one probabilistic model and one vector space model) perform similarly well and the results support the hypothesis that IR provides a practicable solution to semi-automatically recovering traceability links between code and documentation. It is worth mentioning that comments were not taken into account for this study, only mnemonics used for classes, attributes, methods, and parameters were regarded.

4.5.3.3 Natural language text readability

The readability of text has been a task for researchers for a long time. As early as in the 1920s, vocabulary difficulty and sentence length was used to predict how easily a text could be read. But it would take until the late 1940s, when Rudolf Flesch published his work that set the ground for his now well-known readability formula and its successors. By the 1980s there existed over 200 of these formulas and over a thousand published studies that backed up their theoretical and statistical validity. [75]

Flesch developed his readability formula, which was the most widely used formula at this time, containing two parts. The first part is built by the ***Reading Ease formula***, which predicts the reading ease on a scale from 1 – 100 with higher value denoting better readability. The second part predicts human interest by counting the number of personal words (pronouns and names) and personal sentences (like quotes or explanations). The first part holds interest for applying it on source code documentation, or more precisely comments, so it will be explained in more detail.

Basically, the formula needs the number of words, syllables and sentences of the text, respectively the average number of each. Flesch stated that titles, headings, section or paragraph numbers, captions, date lines and signature lines should be skipped. Regarding syllables, he stated that they should be counted as the words are pronounced; words on the other hand are abbreviations, numbers, or hyphenated words. [76] The figures have to be put in the formula:

$$Score = 206.835 - 1.015\ x\ ASL - 84.6\ x\ ASW$$

with *ASL* being the average sentence length (or the number of words divided by number of sentences) and *ASW* being the average word length in syllables (or the number of syllables divided by number of words). A score below 30 indicates that the text is very difficult to read, whereas a score above 70 indicates fairly easy to very easy readability. [75]

Another popular readability formula, based on Flesch's formula and developed by a study commissioned by the U.S. Navy in 1976, is the **Flesch-Kincaid formula**. It provides a grade level, denoting the years of education one needs to understand the text. Using the average sentence (ASL) and word (ASW) length of Flesch's formula, the level is calculated by:

$$GradeLevel = 0.39\ x\ ASL + 11.8\ x\ ASW - 15.59$$

Both formulas are implemented in Microsoft Word, which already includes a method to count syllables and words and was therefore used for the calculations.

Another approach, combining NLP, machine learning, statistics and program analysis, was done in [77]. The authors extracted source code comments containing keywords like "lock", "acquire", "release", or "hold", which were then partly manually labelled accordingly (e.g. lock-related, call-related, etc.). With this labelled data a classifier was trained to produce a model with which diverse comments can be classified because programmers tend to write similar comments. Subsequently the automatically classified comment data got manually evaluated due to the lack of a proper ground truth. Even though the focus was only on certain keywords that indicate important steps in the programming, the analysis showed good accuracy. Extending this approach on other important comment topics would be of great interest.

In [78] an approach to evaluate source code by combining both language quality assessment and consistency between source code and its comments is presented. For this Javadoc comments get extracted and are marked-up so the following NLP tool is able to process it easily. GATE[135] is used for this task, and the JavadocMiner is implemented as a pipeline, using individual processing resources. Readability of comments is also analysed using the Flesch metric, which is the least correlated with the number of bug defects. Other readability metrics, namely Fog Index[136] and Flesch-Kincaid Grade Level Score achieve much better correlation values. The difference is dramatic for all tested examples, although both Flesch-Kincaid and Flesch metrics are calculated using the same values. Further development of this approach can be found in [79]. There the authors provide an integration of natural language processing services in Eclipse, a common Java development environment. The plug-in integrates their developed Semantic Assistants architecture, which offers semantic analysis services like named entity detection or quality analysis of source code comments[137] by using a Web service.

4.5.4 Comments

A work focusing on the **comment density** of open source projects can be found in [80]. Its assumption is that the amount of comments in source code indicates the quality and maintainability of the program. One conclusion is that comment density is independent of the size of the project as well as of the size of the

[135] A software for text processing, see http://gate.ac.uk/

[136] Robert Gunning developed an equation for grading the „fog", the unnecessary complexity of text, in newspapers. $GradeLevel = 0.4\ (ASL + hard\ words)$, where *hard words* is the number of words with more than two syllables for each 100 words.

[137] Information on the features can be found on http://www.semanticsoftware.info/semantic-assistants-eclipse-plugin.

programmer team size. The average density found in over 5000 projects was 19%, which can be said to be representing successful projects because the authors only used active projects with activity in the previous year.

A totally different point of view is described by [81], which is using descriptions of how source code evolves over time, called evolutionary annotations. The method developed includes an automatic creation of these annotations, out of change logs, defect tracking systems and mailing lists. The difference to normal comments is that these methods explain the change until now and not the current role of the system.

Regarding **text categorization** and thus determine the language a text was written in, the paper of Cavnar and Trenkle [82] deals with n-grams. N-grams are *n*-character slices of a word[138], generated for each text that has to be categorized. With these slices a profile gets generated for the document, which gets compared to profiles from other documents that are already categorized. The shortest (profile) distance between two profiles indicates the highest similarity, thus the most probable language for the text. Implementations can be found online[139].

Text proofing is also of interest when analysing text in natural language because largely incorrect text sections will be difficult to understand. Commonly used approaches include the usage of part-of-speech (POS) syntactic patterns [83] or word sequence patterns that are compared to entries in error corpora [84]. These can be generated by the usage of *Markov chains*. Markov chains are random processes, depending only on the current state and not the sequence preceding this state. They can be of different orders, denoting on how many past states the future state depends (e.g. in a Markov chain of order three the next word depends on the previously three). Thus in combination with POS tagging and a preliminary *n-gram* generation on word level, it is possible to check if the text contains constructs that are not allowed according to the grammar used.

Another view on comments, concerning malicious behaviour from the documentation's author, can be found in [85], where three different methods are presented that are able to identify **artificially generated text**. Besides looking for basic lexicometric features, like standard word or sentence length, and the perplexity of language models, where a high perplexity indicates a fake content, the authors suggest using a variety of features and techniques, including a method based on a relative entropy measure which captures short range dependencies between words. It will not be possible to verify automatically if the content of documentation actually fits, thus a manual evaluation will be necessary. However, if the documents put into escrow need to be pre-selected, e.g. if there is a vast amount of documentation, these filtering methods presented in [85] can be of help.

[138] For completeness it needs to be stated that n-grams are more generic, they are a sequence of units drawn from a longer sequence. For text the units (or levels) are usually words or characters. In the case of text categorization as described in [82] only characters are used.

[139] E.g. http://odur.let.rug.nl/~vannoord/TextCat/

4.5.5 Maturity of software for reuse

The Software Reuse Working Group of NASA's Earth Science Data Systems proposed a set of Reuse Readiness Levels (RRLs) in [86]. They also identified nine topic areas that are important for measuring reuse maturity of software: Documentation, Extensibility, Intellectual Property Issues, Modularity, Packaging, Portability, Standards Compliance, Support, and Verification and Testing. Each of these areas has at most nine levels of reuse readiness, ranging from 1 (least mature) to 9 (most mature). For example documentation on level 1 indicates that there is little or no documentation available, whereas on level 9 there exists documentation on design, customization, testing, use, and reuse. Mapping the area's levels to our Software Escrow approach, most of them can be used to grade the artefacts delivered to the escrow agent. A use case can be found in the planning phase, in which developer and customer agree on the level of different artefacts. There are only a few exceptions to the use of levels. Regarding extensibility we can only evaluate how the software was designed to provide extension possibilities. A demonstration of extensibility as it is mentioned in level 7 to 9 is not feasible and necessary. The area of support does not make sense in the domain of Software Escrow, because support contracts have to be dealt with outside of the escrow agreement. The Verification and Testing area as described in the RRL report, aims mainly at evaluating the functionality of the software. Level 1 and 2 ("no testing", respectively "software application formulated and unit testing performed") are the only levels reasonable for escrow. The most important aspect about functionality in Software Escrow is to preserve it, not to check if the software operates as desired. In all other areas the detail of how much has to be provided in each level increases. Thus it offers a reference point for the Software Escrow parties and the possibility to agree on the level of detail each artefact has to fulfil.

4.6 Prototype implementation

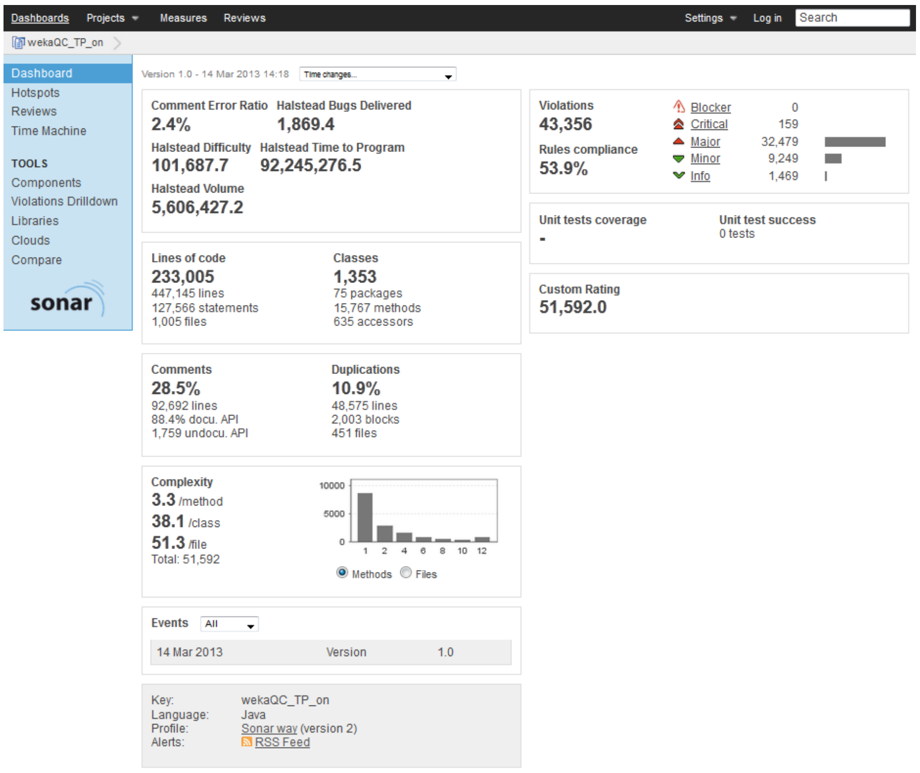
A prototypical framework based on the model described in Section 4.3.3 will be presented here. This section describes practical implementations of the plugins and their usefulness. The implementation of the framework is built on a Continuous Integration tool, which we extend with additional plugins.

The following Section 4.6.1 describes the system developed. A section about experiments on open source projects follows in 4.6.2, including results and discussion.

4.6.1 Implementation

The implementation of the prototype was built on an existing framework, namely Sonar[140]. Because the model described in Section 4.3.3, respectively the structure given in Figure 34, is generic enough it can still be applied to this framework. A screenshot of how the overview of a project looks like in Sonar can be found in Figure 35. It contains a rough summary of the project's performance. Figure 36 shows the violation's overview, which contains links to the classes and their detailed violation reports.

[140] http://www.sonarsource.org/



Version 1.0 - 14 Mar 2013 14:18

Comment Error Ratio 2.4%
Halstead Bugs Delivered 1,869.4
Halstead Difficulty 101,687.7
Halstead Time to Program 92,245,276.5
Halstead Volume 5,606,427.2

Violations 43,356
Rules compliance 53.9%

Blocker	0
Critical	159
Major	32,479
Minor	9,249
Info	1,469

Unit tests coverage -
Unit test success 0 tests

Custom Rating 51,592.0

Lines of code 233,005
447,145 lines
127,566 statements
1,005 files

Classes 1,353
75 packages
15,767 methods
635 accessors

Comments 28.5%
92,692 lines
88.4% docu. API
1,759 undocu. API

Duplications 10.9%
48,575 lines
2,003 blocks
451 files

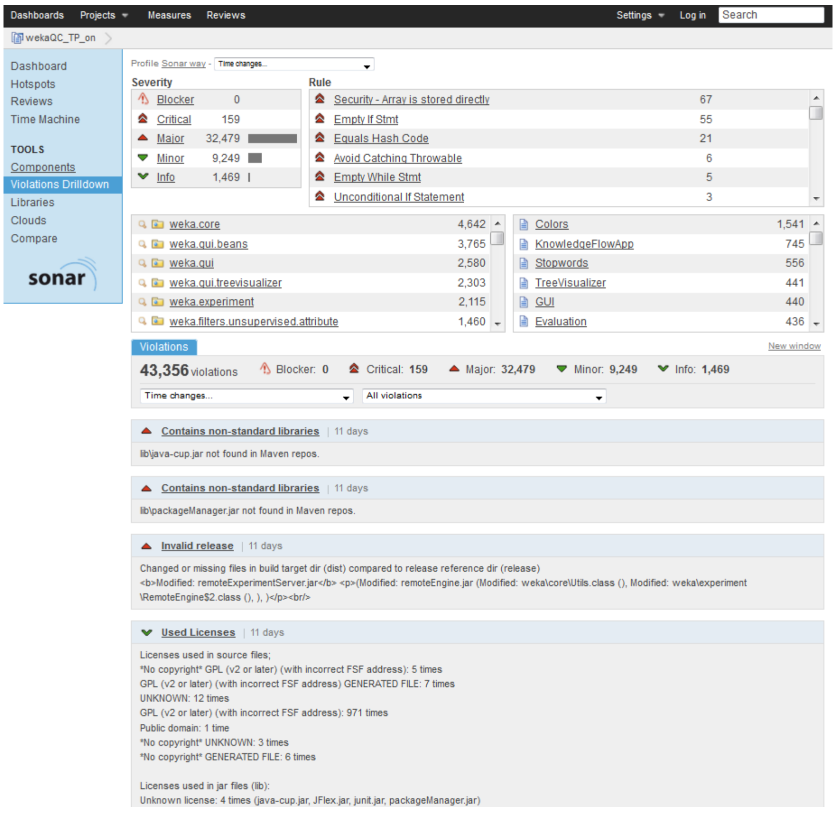
Complexity
3.3 /method
38.1 /class
51.3 /file
Total: 51,592

Events All

14 Mar 2013	Version	1.0

Key:	wekaQC_TP_on
Language:	Java
Profile:	Sonar way (version 2)
Alerts:	RSS Feed

Figure 35: Overview of project in Sonar



Profile Sonar way -

Severity	
Blocker	0
Critical	159
Major	32,479
Minor	9,249
Info	1,469

Rule	
Security - Array is stored directly	67
Empty If Stmt	55
Equals Hash Code	21
Avoid Catching Throwable	6
Empty While Stmt	5
Unconditional If Statement	3

weka.core	4,642
weka.gui.beans	3,765
weka.gui	2,580
weka.gui.treevisualizer	2,303
weka.experiment	2,115
weka.filters.unsupervised.attribute	1,460

Colors	1,541
KnowledgeFlowApp	745
Stopwords	556
TreeVisualizer	441
GUI	440
Evaluation	436

Violations

43,356 violations Blocker: 0 Critical: 159 Major: 32,479 Minor: 9,249 Info: 1,469

Time changes... All violations

Contains non-standard libraries | 11 days

lib\java-cup.jar not found in Maven repos.

Contains non-standard libraries | 11 days

lib\packageManager.jar not found in Maven repos.

Invalid release | 11 days

Changed or missing files in build target dir (dist) compared to release reference dir (release)
<b>Modified: remoteExperimentServer.jar</b> <p>(Modified: remoteEngine.jar (Modified: weka\core\Utils.class (), Modified: weka\experiment \RemoteEngine$2.class (),),)</p><br/>

Used Licenses | 11 days

Licenses used in source files;
No copyright GPL (v2 or later) (with incorrect FSF address): 5 times
GPL (v2 or later) (with incorrect FSF address) GENERATED FILE: 7 times
UNKNOWN: 12 times
GPL (v2 or later) (with incorrect FSF address): 971 times
Public domain: 1 time
No copyright UNKNOWN: 3 times
No copyright GENERATED FILE: 6 times

Licenses used in jar files (lib):
Unknown license: 4 times (java-cup.jar, JFlex.jar, junit.jar, packageManager.jar)

Figure 36: Overview of violations in Sonar

4.6.1.1 System setup

Sonar Version 3.4.1 [141], a platform for continuous quality inspection, has been used as base system. It is written in Java and licensed under the LGPL. The architecture of Sonar is based on the client/server model. The server manages the results of the analyses and provides a web based presentation of the issues and measurements found in the examined projects. The client, also called runner, analyses the project and transmits the results to the server.

The quality of a project is measured using metrics and rules, which produce value results and violations, respectively. For several languages common metrics and rules are provided out of the box, but it also allows integration of custom plugins.

4.6.1.2 Sonar Plugins

All of the plugins described in this section are programming language independent. The language used is taken into account already during the extraction phase (cf. the Container Handler in Figure 34). The plugin list includes a subset of the plugins presented in Section 4.3.3.2 to showcase the most important and feasible features.

Sonar provides bundled plugins and metrics which overlap with our requirements, and therefore, some of these existing plugins have been used without modifications. These include:

- LOC (cf. Sections 4.3.3.2.1 and 4.5.1) - The LOC are reflected by Sonar's Java/Lines of code metric [87].
- NCSS (Non Commenting Source Statements, cf. Sections 4.3.3.2.1 and 4.5.1) - Sonar does not use JavaNCSS [88] anymore because of various flaws [89]. An alternative metric in Sonar is the Java/Statement metric, which does not count import declarations, class and method definitions, etc.
- Cyclomatic Complexity (McCabe) (cf. Sections 4.3.3.2.1 and 4.5.1) - The Complexity method of Sonar uses McCabe's metric.
- Duplicate Code - The number of duplicate lines is counted by the Sonar Duplicated lines metric. According to the documentation [90] the detection is done on a per statement basis.
- Prenatal communication (cf. Section 4.3.3.2.1) - PMD, which is integrated in Sonar, provides the suitable ConstructorCallsOverridableMethod [91] rule.
- Coupling - To determine the coupling, Afferent couplings and Efferent couplings are calculated.
- Cohesion - Sonar's Lack of cohesion of methods is used to measure the cohesion.

For the other measurements which are not covered by Sonar plugins, the following prototypes have been implemented[142]:

[141] http://www.sonarsource.org/downloads/, accessed 31.01.2013

[142] Note: The development of the prototype was a continuous process. Thus some of the plugins were not ready when the experiments were done. The Halstead and Custom Rating Plugin could not be included in the tests. For the sake of completeness they are mentioned here.

- *Build Plugin* (cf. Section 4.3.3.2.6) - The build plugin raises a violation if it is not possible for the Sonar runner to execute the build process successfully. The preconditions are that the system the runner is executed on does meet all required preconditions, like dependencies, compilers in correct version, etc. To be generic, the plugin executes a custom command in the project root directory (e.g. *mvn install*). It then parses the response of this command for (also custom) messages that suggest the failure of the build. Furthermore the return value of the build command is evaluated.
- *Release Plugin* (cf. Section 4.3.3.2.6) - This plugin checks if the output generated by the Build Plugin matches a provided reference. For this purpose two directories are compared, where one is the build target directory of the Build Plugin, and the other a directory which contains the expected result. The Release Plugin compares XML files by ignoring the element ordering, as it may differ between subsequent builds. Zip-compressed files are extracted and contents compared individually. This is needed because of XML files which are embedded in jar files. Other file types are compared using the FileUtils helper in the Apache Commons IO library[143]. For files, which do not have to be checked for changes (e.g. META-INF files in jars), it is possible to exclude them from comparison.
- *Comment Language Detection Plugin* (cf. Section 4.3.3.2.7) - The Comment Language Detection Plugin detects the language of comments and raises a violation if a language other than an allowed one is found. This is done either on a per-file basis (to minimize false positives because of too short text) or on a per-comment basis, depending on the configuration. The comments are extracted using SSLR [92] and analysed using the Java Text Categorizing Library [93], which uses an algorithm based on n-grams as described in Section 4.5.4. To minimize the rate of false positives, comments are stripped from HTML elements as well as from Javadoc tags, including associated data. Additionally, short comments can be either ignored or checked by using a word list (dictionary approach) to determine if the comment is in the desired language.
- *Nonstandard Libraries Plugin* (cf. Section 4.3.3.2.5) - This plugin raises a violation if dependencies are found which may not originate from official sources. For this, the hash values of all jar files in a pre-defined directory are calculated and looked up in the Maven Central Repository [94]. If the search does not find a library in the repository that corresponds to the jar file's hash value, the plugin assumes that this jar file originates from unofficial sources. For other libraries than jar, the plugin looks up hashes in the National Software Reference Library[144].
- *Unsafe Calls Plugin* (cf. Section 4.3.3.2.5) - The idea of the Unsafe Calls Plugin is that potentially dangerous calls like System calls (e.g. calling the copy command of the system), Binary calls (e.g. calling a text editor or other program from within the source code), or Web Service calls, can be detected by a text-based search over source files. The plugin raises a violation if one of the provided regular expressions matches in any of the source files.

[143] http://commons.apache.org/io/api-release/index.html?org/apache/commons/io/package-summary.html
[144] http://www.nsrl.nist.gov/

- *Comment Text Proof Check Plugin* - This plugin uses the *LanguageTool* [95] to find various issues like misspellings, wrong grammar, uncommon phrases, etc. in source code comments. The comments again are extracted using SSLR. It can be specified which language to use for checking the comments and which aforementioned issues are excluded from the results. As in the Comment Language Detection plugin, HTML and Javadoc elements and their associated data can be stripped from the source code before analysis.
- *License Check Plugin* - The header of each source file is expected to contain some information about the license under which the according file is published. Those licenses are extracted using the Perl script licensecheck[145], which is part of the Debian *devscripts* package. For jar files, like libraries, that contain information on their license, the License Maven Plugin[146] is used. Violations informing about the detected licenses are issued for each source file and summarized in an overview.
- *Halstead Plugin* - The Halstead Plugin does not yet consider all possible statements, but can be used to approximate the Halstead metrics Volume, Difficulty, Effort, Time to program and Bugs delivered. As in languages like Java, not for every component of possible statements it can be unambiguously determined if it is either operator or operand[147], the results may be different compared to other tools.
- *Custom Rating Plugin* - The custom rating plugin provides the possibility to calculate a measure using a custom defined formula that can make use of other measures and violation counts. Measures and violation counts are replaced and passed to the Math Expression Parser of the Symja project[148], where the formula is evaluated.

4.6.2 Experiments

Different experiments were conducted to evaluate the plugins of the framework. We do not have a ground truth for each project's evaluation, containing e.g. Web services that have to be found or languages used in the comments. For this it would be necessary to examine each project manually, preferably combining results by multiple judgments regarding the quality of artefacts, as it is common for example in Information Retrieval evaluation. Because this would exceed the scope of this work, we decided to focus only on the true and false positives (retrieved) quality attributes of artefacts, leaving aside true and false negatives (not retrieved). This approach also reflects our intention to the framework: Find as many violations as possible to present to the reviewer, in order to facilitate and quicken his work.

[145] http://www.beathovn.de/licensecheck, accessed 11.02.2013; Note for the attentive and maybe wondering reader: If we feed our framework with its own code, the License Check Plugin code would raise an Unsafe Call violation because of the Perl calling part.

[146] http://mojo.codehaus.org/license-maven-plugin/

[147] http://www.virtualmachinery.com/sidebar2.htm, accessed on 2013-03-21

[148] https://code.google.com/p/symja/wiki/MathExpressionParser, accessed on 2013-03-21

The following Section 4.6.2.1 describes experiments conducted prior to implementing the framework in order to determine the feasibility of different plugins. The tests in the subsequent Section 4.6.2.2 are of a different nature, dealing with various projects and evaluating the technical framework as a whole. All experiments were conducted under Windows 7 Enterprise, 64 bit.

The discussion and interpretation of the experiments in Section 4.6.2.3 analyses the results of the experiments. Thus the following Sections 4.6.2.1 and 4.6.2.2 only include the raw results without going into detail what they mean for the scope of this work.

4.6.2.1 Feasibility experiments

For this first part of experiments the sources of Apache Lucene[149], an information retrieval software, were used. A description of the metrics used can be found in Section 4.5, or they reference a plugin in Section 4.3.3.2. The experiments include how to obfuscate code (4.6.2.1.1), the readability of comments (4.6.2.1.2), and an analysis of complexity and comment length related ratios (4.6.2.1.3).

4.6.2.1.1 Code Obfuscation

For a code obfuscation experiment we tried to generate obfuscated source code and use specific metrics on it. Because there is no tool to our knowledge that generates obfuscated source code, we were only able to obfuscate binaries and decompile them afterwards. As mentioned in Section 4.1.3.3, the same problems as in [56] were faced when trying to do this experiment – decompiling an obfuscated binary is not always possible or leads to incorrect results. Our experiment also did not succeed due to decompiling problems, nevertheless we will shortly present it to highlight the problems that can be encountered at a technical review.

The obfuscator of choice was the open source software *ProGuard 4.7*[150], for decompiling the *Java Decompiler 0.6.0*[151] was used. ProGuard can only process (binary) *jar* files; its output is also binary. When testing with the source code of Lucene, re-creating the source code from the obfuscated binaries was not possible because the decompiled source code was missing many parts of the original code but included commented paragraphs with machine code instructions. Thus comparing the metric results to the non-obfuscated version of the program is hardly possible because for e.g. LOC some methods are missing. For smaller projects decompilation was easier and resulted in correctly decompiled files. Nevertheless compile errors occurred (e.g. use of not initialised variables or variable names with special characters). Regarding commonly used metrics, obfuscated source code does not differ much compared to its non-obfuscated version. Complexity, Difficulty, Effort or others tend to increase slightly, but not enough to mark source code definitely as obfuscated.

[149] http://lucene.apache.org/core/

[150] http://proguard.sourceforge.net/

[151] http://java.decompiler.free.fr/

4.6.2.1.2 Readability of comments

The experiment regarding readability of comments in source code builds on the assumption that better readable comments are easier to understand and thus improve the overall source code quality. To investigate this we looked at the readability-to-comment-words ratio in order to normalize the comments of each single class and order them.

For this purpose we extended Checkstyle (cf. Section 4.5.2) and extracted both inline comments and Javadoc annotations. For analysis we needed complete sentences because the sentence length is an important factor. Thus we changed so called inline comments (comments next to the source code) by placing a full stop behind them if they had no other lines following. In the case of Javadoc comments we assumed that they already consisted of full sentences with a punctuation mark. Because implementing a functional Flesch- or Flesch-Kincaid-Readability module is not trivial, the extracted texts were analysed using the Microsoft Word Professional Plus 2010 Readability[152] tool with a JAVA-COM Bridge[153]. The readability tool already provides both implementations for text in English. The comment word number was extracted from Word as well.

The results of the metrics were set in relation to the amount of comment words, which led to the generation of the following two ratios:

- Flesch Reading Ease : Comment words
- Flesch-Kincaid : Comment words

For the experiment Lucene version 3.2 was used as test project.

To get an overview of the result range, both maxima and minima of the experiments done on Lucene 3.2 are shown in Table 8 and Table 9. The files are named with their path, starting within the src folder.
Table 8 contains the results for the experiments producing a Flesch-to-Comment-Words ratio, or the Flesch index normalized by the number of comment words. The first part contains the best performing classes, starting with the very best. The second part holds the worst performing files.
Table 9 contains the results of the Flesch-Kincaid-to-Comment-Words ratio experiment. Because Flesch-Kincaid calculates the years of education one needs for understanding the evaluated text, the first part of the table contains the best performing classes, whereas the second part holds the poorer documented classes (including classes listed because of the above mentioned Checkstyle bug).

Regarding the ratios, the experiments showed that there are some Java files with a Flesch Reading Ease or a Flesch-Kincaid grade level of 0.0. In both cases the files were sorted according to the result of the division (readability value : comment words). Because of a bug found in Checkstyle[154] most of the classes that have a Flesch index of 0.0 describe enum-types. *QueryParserConstants* however is an *interface* and

[152] http://office.microsoft.com/en-us/word-help/test-your-document-s-readability-HP010354286.aspx

[153] JACOB is a JAVA-COM Bridge that allows you to call COM Automation components from Java. http://sourceforge.net/projects/jacob-project/

[154] Checkstyle was found to have problems extracting comments out of files that only describe enum-types. It always lists such files to have no comments at all.

WordBreakTestUnicode_6_0_0 is a normal class, obviously with poor readability results. Regarding the Flesch-Kincaid grade a higher value means more difficulty (more years of education for comprehension), nevertheless Microsoft Word measured a value of zero. A reason for this behaviour could not be found.

Altogether the results show that most of the Flesch indexes are roughly between 40 and 60, the average of the reading ease lies at 44.34, which lies between “fairly difficult” and “difficult” according to [76]. For Flesch-Kincaid the average years of education to understand Lucene’s comments lies at 10.38, which means that an average student in the tenth grade (approximately around age 14-16) is able to understand it.

Table 8: Flesch:CommentWords results of readability-ratio experiments for Lucene 3.2

Class	**Ratio**	**Value**
tools\java\org\apache\lucene\validation\LicenseType.java	0.0 : 0	N/A
java\org\apache\lucene\util\Version.java	0.0 : 0	N/A
java\org\apache\lucene\document\FieldSelectorResult.java	0.0 : 0	N/A
java\org\apache\lucene\queryParser\QueryParserTokenManager.java	47.6 : 40	1.1899999
test\org\apache\lucene\search\TestSpanQueryFilter.java	40.1 : 100	0.401
test\org\apache\lucene\index\TestByteSlices.java	39.2 : 98	0.4
test\org\apache\lucene\index\TestPositionBasedTermVectorMapper.java	55.9 : 150	0.3726667
java\org\apache\lucene\index\TermVectorEntry.java	36.0 : 97	0.371134
java\org\apache\lucene\document\LoadFirstFieldSelector.java	35.8 : 98	0.3653061
test\org\apache\lucene\util\TestFieldCacheSanityChecker.java	41.2 : 116	0.35517243
java\org\apache\lucene\analysis\LowerCaseTokenizer.java	10.0 : 420	0.023809524
java\org\apache\lucene\queryParser\QueryParser.java	52.5 : 2442	0.021498771
java\org\apache\lucene\search\Similarity.java	54.7 : 4720	0.011588983
java\org\apache\lucene\index\IndexReader.java	49.9 : 5692	0.00876669
java\org\apache\lucene\analysis\WhitespaceTokenizer.java	2.9 : 365	0.007945206
java\org\apache\lucene\analysis\ASCIIFoldingFilter.java	55.3 : 9036	0.0061199646
java\org\apache\lucene\index\IndexWriter.java	52.1 : 12536	0.0041560307
test\org\apache\lucene\analysis\TestASCIIFoldingFilter.java	42.0 : 10390	0.0040423484
test\org\apache\lucene\analysis\WordBreakTestUnicode_6_0_0.java	0.0 : 29319	0.0
java\org\apache\lucene\queryParser\QueryParserConstants.java	0.0 : 96	0.0

Table 9: Flesch-Kincaid:CommentWords results of readability-ratio experiments for Lucene 3.2

Class	**Ratio**	**Value**
test\org\apache\lucene\analysis\WordBreakTestUnicode_6_0_0.java	20.9 : 29319	7.128483E-4
java\org\apache\lucene\analysis\ASCIIFoldingFilter.java	6.6 : 9036	7.304117E-4
java\org\apache\lucene\index\IndexWriter.java	9.5 : 12536	7.5781747E-4

Class	Ratio	Value
test\org\apache\lucene\analysis\TestASCIIFoldingFilter.java	8.0 : 10390	7.699711E-4
java\org\apache\lucene\index\IndexReader.java	10.3 : 5692	0.0018095573
java\org\apache\lucene\search\Similarity.java	9.5 : 4720	0.002012712
java\org\apache\lucene\queryParser\QueryParser.java	8.5 : 2442	0.0034807534
java\org\apache\lucene\index\DocumentsWriter.java	6.7 : 1865	0.0035924932
java\org\apache\lucene\util\OpenBitSet.java	7.1 : 1938	0.0036635706
java\org\apache\lucene\index\SegmentInfos.java	7.2 : 1955	0.0036828644
java\org\apache\lucene\index\TermVectorEntry.java	12.2 : 97	0.12577319
java\org\apache\lucene\document\LoadFirstFieldSelector.java	12.7 : 98	0.12959184
test\org\apache\lucene\index\TestTermdocPerf.java	14.4 : 98	0.14693877
java\org\apache\lucene\index\TermVectorEntryFreqSortedComparator.java	14.6 : 98	0.14897959
test\org\apache\lucene\search\TestFieldCache.java	14.6 : 85	0.17176472
java\org\apache\lucene\queryParser\QueryParserTokenManager.java	7.3 : 40	0.1825
java\org\apache\lucene\queryParser\QueryParserConstants.java	20.6 : 96	0.21458334
java\org\apache\lucene\document\FieldSelectorResult.java	0.0 : 0	N/A
java\org\apache\lucene\util\Version.java	0.0 : 0	N/A
tools\java\org\apache\lucene\validation\LicenseType.java	0.0 : 0	N/A

4.6.2.1.3 Complexity and comment length

In this experiment we investigate the ratio of complexity of a class and the comment length (or number of comment words) because we want to see if a class with high complexity is compensated with more comments. For this we look at the *Complexity:Codelines* ratio as well as at the *CommentWords:Codelines* ratio. Code lines are LOC and NCSS.

For providing the NCSS and cyclomatic complexity values, the tool JavaNCSS version 32.53[155] was used. It needs to be stated that this JavaNCSS version did not work without any problems[156], but due to the lack of a newer or patched version or a comparable tool it was used nonetheless. The LOC value was provided by CLOC[157] version 1.56. The length of comments, which is the sum of comment words, was provided by Microsoft Word, as in the readability calculations (cf. Section 4.6.2.1.2). The resulting ratios are:

- Comment words : LOC
- Comment words : NCSS
- Complexity : LOC

[155] See http://www.kclee.de/clemens/java/javancss/ for more information.

[156] One error found was a parsing exception occurring when a for-each loop was used. Because of the exception the program terminated without stating a right NCSS number.

[157] http://cloc.sourceforge.net/

- Complexity : NCSS

Regarding the ratio of comment words to code lines, normalization makes sense because the more code lines there are, the more explanatory comment words there need to be. Of course a high number of words does not automatically imply that the content of the comment describes the annotated source code lines or even makes sense. But for a general evaluation, we assume that very short comments are not enough for a useful description, thus the corresponding classes have to be checked manually afterwards.

Regarding the ratio of complexity to code lines, a low complexity for many LOC is optimal, therefore the ratio should be rather low. A high result on the other hand indicates that the complexity is proportionally too high for this amount of code lines.

Table 10: Comment words : LOC results of complexity and comment length experiments for Lucene 3.2

Class	Ratio	Value
java\org\apache\lucene\search\Collector.java	805 : 9	89.44444
java\org\apache\lucene\index\IndexDeletionPolicy.java	561 : 7	80.14286
java\org\apache\lucene\search\DocIdSetIterator.java	457 : 8	57.125
java\org\apache\lucene\analysis\tokenattributes\PositionIncrementAttribute.java	322 : 6	53.666668
java\org\apache\lucene\index\TermPositionVector.java	235 : 5	47.0
java\org\apache\lucene\util\AttributeReflector.java	182 : 4	45.5
java\org\apache\lucene\search\Similarity.java	4720 : 111	42.522522
java\org\apache\lucene\search\Weight.java	576 : 14	41.142857
java\org\apache\lucene\index\TermFreqVector.java	356 : 9	39.555557
java\org\apache\lucene\index\TermPositions.java	394 : 10	39.4
test\org\apache\lucene\search\spans\TestFieldMaskingSpanQuery.java	124 : 273	0.45421246
test\org\apache\lucene\search\spans\TestSpans.java	171 : 379	0.45118734
test\org\apache\lucene\index\TestFieldsReader.java	218 : 487	0.4476386
test\org\apache\lucene\index\TestTermVectorsWriter.java	161 : 384	0.41927084
java\org\apache\lucene\analysis\standard\UAX29URLEmailTokenizer.java	1273 : 3414	0.3728764
test\org\apache\lucene\search\spans\TestPayloadSpans.java	155 : 427	0.36299765
java\org\apache\lucene\queryParser\QueryParserTokenManager.java	40 : 1228	0.03257329
tools\java\org\apache\lucene\validation\LicenseType.java	0 : 30	0.0
java\org\apache\lucene\util\Version.java	0 : 24	0.0
java\org\apache\lucene\document\FieldSelectorResult.java	0 : 10	0.0

Table 11: Comment words : NCSS results of complexity and comment length experiments for Lucene 3.2

Class	Ratio	Value
test\org\apache\lucene\search\TestMultiPhraseQuery.java	416 : 0	Infinity
java\org\apache\lucene\search\Collector.java	805 : 8	100.625
java\org\apache\lucene\index\IndexDeletionPolicy.java	561 : 6	93.5
test\org\apache\lucene\analysis\TestASCIIFoldingFilter.java	10390 : 113	91.9469
java\org\apache\lucene\analysis\tokenattributes\TermAttributeImpl.java	135 : 2	67.5
java\org\apache\lucene\search\DocIdSetIterator.java	457 : 7	65.28571
java\org\apache\lucene\analysis\tokenattributes\PositionIncrementAttribute.java	322 : 5	64.4
java\org\apache\lucene\search\Similarity.java	4720 : 75	62.933334
java\org\apache\lucene\util\AttributeReflector.java	182 : 3	60.666668
java\org\apache\lucene\index\TermPositionVector.java	235 : 4	58.75
test\org\apache\lucene\index\TestIndexReaderReopen.java	485 : 762	0.63648295
test\org\apache\lucene\index\TestIndexWriterExceptions.java	506 : 797	0.6348808
test\org\apache\lucene\index\TestStressIndexing2.java	284 : 473	0.60042286
test\org\apache\lucene\index\TestFieldsReader.java	218 : 410	0.53170735
test\org\apache\lucene\index\TestTermVectorsWriter.java	161 : 315	0.51111114
test\org\apache\lucene\search\spans\TestPayloadSpans.java	155 : 363	0.42699724
java\org\apache\lucene\queryParser\QueryParserTokenManager.java	40 : 991	0.04036327
tools\java\org\apache\lucene\validation\LicenseType.java	0 : 11	0.0
java\org\apache\lucene\util\Version.java	0 : 4	0.0
java\org\apache\lucene\document\FieldSelectorResult.java	0 : 2	0.0

Table 12: Complexity : LOC results of complexity and comment length experiments for Lucene 3.2

Class	Ratio	Value
java\org\apache\lucene\util\AttributeReflector.java	1.0 : 4	0.25
java\org\apache\lucene\messages\NLSException.java	1.0 : 4	0.25
java\org\apache\lucene\store\LockStressTest.java	16.0 : 67	0.23880596
java\org\apache\lucene\analysis\NormalizeCharMap.java	6.0 : 28	0.21428572
java\org\apache\lucene\index\TermPositionVector.java	1.0 : 5	0.2
java\org\apache\lucene\analysis\CharStream.java	1.0 : 5	0.2
java\org\apache\lucene\document\FieldSelector.java	1.0 : 5	0.2
java\org\apache\lucene\index\InvertedDocEndConsumerPerField.java	1.0 : 5	0.2
test\org\apache\lucene\util\TestVersion.java	2.0 : 11	0.18181819
test\org\apache\lucene\index\TestRollingUpdates.java	8.0 : 45	0.17777778

Class	Ratio	Value
test\org\apache\lucene\index\TestIndexWriter.java	3.81 : 2460	0.0015487805
java\org\apache\lucene\index\IndexWriter.java	3.43 : 2240	0.0015312501
test\org\apache\lucene\analysis\TestASCIIFoldingFilter.java	2.33 : 1596	0.0014598997
java\org\apache\lucene\analysis\standard\UAX29URLEmailTokenizer.java	3.47 : 3414	0.0010164031
test\org\apache\lucene\analysis\WordBreakTestUnicode_6_0_0.java	1.0 : 1963	5.0942437E-4
test\org\apache\lucene\search\TestMultiPhraseQuery.java	0.0 : 427	0.0
java\org\apache\lucene\analysis\tokenattributes\TermAttributeImpl.java	0.0 : 4	0.0
java\org\apache\lucene\util\Attribute.java	0.0 : 3	0.0
java\org\apache\lucene\queryParser\QueryParserConstants.java	0.0 : 76	0.0
java\org\apache\lucene\document\FieldSelectorResult.java	0.0 : 10	0.0

Table 13: Complexity : NCSS results of complexity and comment length experiments for Lucene 3.2

Class	Ratio	Value
test\org\apache\lucene\search\TestMultiPhraseQuery.java	0.0 : 0	N/A
java\org\apache\lucene\store\LockStressTest.java	16.0 : 42	0.3809524
java\org\apache\lucene\util\AttributeReflector.java	1.0 : 3	0.33333334
java\org\apache\lucene\messages\NLSException.java	1.0 : 3	0.33333334
java\org\apache\lucene\analysis\NormalizeCharMap.java	6.0 : 22	0.27272728
java\org\apache\lucene\index\TermVectorEntryFreqSortedComparator.java	3.0 : 11	0.27272728
java\org\apache\lucene\index\TermPositionVector.java	1.0 : 4	0.25
java\org\apache\lucene\analysis\CharStream.java	1.0 : 4	0.25
java\org\apache\lucene\document\FieldSelector.java	1.0 : 4	0.25
java\org\apache\lucene\index\InvertedDocEndConsumerPerField.java	1.0 : 4	0.25
test\org\apache\lucene\search\TestSort.java	1.97 : 680	0.0028970588
test\org\apache\lucene\index\TestIndexReader.java	2.88 : 1161	0.0024806203
test\org\apache\lucene\queryParser\TestQueryParser.java	1.49 : 689	0.0021625545
java\org\apache\lucene\index\IndexWriter.java	3.43 : 1637	0.0020952963
test\org\apache\lucene\index\TestIndexWriter.java	3.81 : 2063	0.001846825
test\org\apache\lucene\analysis\WordBreakTestUnicode_6_0_0.java	1.0 : 982	0.00101833
java\org\apache\lucene\document\FieldSelectorResult.java	0.0 : 2	0.0
java\org\apache\lucene\queryParser\QueryParserConstants.java	0.0 : 40	0.0
java\org\apache\lucene\util\Attribute.java	0.0 : 2	0.0
java\org\apache\lucene\analysis\tokenattributes\TermAttributeImpl.java	0.0 : 2	0.0

Table 10 and Table 11 show the results of the experiment done on Lucene 3.2 of comment-words-to-code-lines, for LOC and NCSS respectively. The first block of ten classes listed holds the best performing classes, starting with the very best. Zero source code lines were found in the test run done with NCSS for

TestMultiPhraseQuery because of the JavaNCSS bug mentioned above[158]. There were also some files listed of only having one or no comment words although they contained many more annotations. The discrepancy could be traced back to Java files containing only *enum* specifications, no class files (cf. Section 4.6.2.1.2 and footnote 154).

Table 12 and Table 13 contain the results of complexity-to-code-lines, again for both LOC and NCSS respectively. Here lower numbers are better, thus the first block contains the worst performing classes, starting with the class with the poorest score. The JavaNCSS bug mentioned above affected the result again. Classes with a complexity of 0 are short classes or interface declarations without any content.

4.6.2.2 Prototype experiments

For this second part of the experiments different projects were used in order to compare the output of the plugins on a larger scale. Information on the plugins tested can be found in Section 4.6.1.2.

4.6.2.2.1 Projects

The set of test projects was mainly taken from the *Qualitas Corpus* (QC) [96], a curated collection of software systems provided on a website hosted by Ewan Tempero[159]. In addition three projects were chosen because their QC version proved to be not buildable or they contained especially interesting artefacts. The most current version of the QC, 20120401r, was used. From the contained 111 systems especially medium size and large projects were chosen. An overview of them can be found below in Table 14. For Sonar it is only possible to run unit tests for Maven projects (containing a Maven build script). We are planning on extending this functionality to also work with other build scripts, like they are needed for Apache Ant. For the tests done, however, we distinguish between availability of tests and their executability, which depends on the project containing a Maven build script or not. The projects were used as is. Provided build scripts for Ant or Maven and folder structures were used and not altered except for cases where it concerned properties of particular need for our plugins. Attributes for our implemented plugins were set to useful values, so that the violations found by the plugins indicated significant problems, i.e. the language detection only gets triggered for comments with at least 10 words, the language detection uses a spell checker for detection if the comment is less than 10 words long, excluded categories of text proofing are “possible typos” and “miscellaneous”, and the text proofing only triggers a violation if the *comment-words to errors* ratio is higher than 20. Other attributes helping to sort out insignificant violations, like disregarding Javadoc mark-up, were turned on.

A more detailed overview of the results of each project can be found later on in distinct sections, including the comment density; the complexity per method, class, and file; the number of Sonar violations; the percentage of code complying with the rules; and the unit tests coverage, if available. A short summary of the project, as well as a comparison with the artefact list (cf. Section 4.2.4), an analysis of the documentation readability, and an evaluation of our implemented plugins complete each section.

[158] The file contains a for-each loop.

[159] http://qualitascorpus.com/

Table 14: Overview of projects

Project	Version (source)	Size (LOC)	Documentation	Tests available	Tests executable
Azureus/Vuze	4.8.1.2([160])	520,007	No	Yes	No
Weka	3.7.5 (QC)	233,005	Yes	Yes	No
Lucene	3.5.0 (QC)	68,622	No	Yes	No
Megamek	0.35.18 (QC)	242,836	No	Yes	No
Xalan	2.7.1 (QC)	172,300	Yes	No	No
JFreeChart	1.0.14([161])	93,460	No	Yes	Yes
Derby	10.6.1.0 (QC)	371,942	Yes	Yes	No
JasperReports	3.7.4 (QC)	169,821	No	No	No
Sonar	3.4 Snapshot([162])	57,744	No	Yes	Yes

4.6.2.2.1.1 Vuze

Vuze[163], previously named Azureus, is a widely used P2P file sharing client for the BitTorrent protocol, including a media player and other media related extensions. The comparison to our artefact list:

- **Intellectual property**: GPL information included about the terms and conditions for copy, distribution and modification of the program.
- **Documentation**: No explicit documentation, only online available.
- **Test environment**: Test cases are provided but there is no documentation about them.
- **Design environment**: Nothing provided.
- **Build environment**: Ant script (build.xml) provided.
- **Applications**: Source code provided.

According to our Sonar run, Vuze is one of the largest projects in the QC, although with only 8.2% comments the comment-code ratio is rather low. About two-thirds of the code complies with the Sonar rules which is an average result compared to other projects. The number of violations triggered by the license plugin is exceptionally high (2,714), because without a violation triggered it would not be possible to look at the results of the plugin for a class.

- **Non-standard library:** triggered for four libraries out five that could not be found in the Maven repository
- **Build**: was able to build Vuze with the provided Ant script and produce a distributable jar
- **Release**: binaries for comparison downloaded from the project's web site[164], which were not identical to the compilation done by the Sonar run

[160] Downloaded from http://svn.vuze.com/public/client/branches/BRANCH_4812/ on 17.01.2013

[161] Downloaded from http://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/1.0.14/jfreechart-1.0.14.zip/download on 24.01.2013

[162] Downloaded master branch from repository https://github.com/SonarSource/sonar/tree/branch-3.4 on 19.03.2013

[163] http://sourceforge.net/projects/azureus/

[164] http://sourceforge.net/projects/azureus/files/vuze/Vuze_4812/Vuze_4812.jar/download, accessed on 13.03.2013

- **Unsafe calls:** 38 binary calls found, several *Runtime* and *ProcessBuilder* calls, some of them in comments
- **Language detection:** detected 10 violations, most of them containing special mark-up or commented source code
- **Text proof**: 42 issues detected, most of them stating that a sentence did not start with an upper letter
- **License check (direct output of plugin)**:

```
o   GPL (with incorrect FSF address): 261 times
o   *No copyright* GPL (v2 or later) (with incorrect FSF address): 3 times
o   *No copyright* GPL (with incorrect FSF address): 487 times
o   UNKNOWN: 2 times
o   *No copyright* GPL (v2) (with incorrect FSF address): 2 times
o   MIT/X11 (BSD like): 1 time
o   GPL (v2 or later) (with incorrect FSF address): 1466 times
o   Apache (v2.0): 2 times
o   *No copyright* UNKNOWN: 562 times
o   *No copyright* GENERATED FILE: 1 time
o   GPL (v2) (with incorrect FSF address): 490 times
```

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
8.2 %	3.0 / 27.4 / 32.8	84,590	67.1 %	–

4.6.2.2.1.2 Weka

Weka, developed by the University of Waikato, offers a collection of machine learning algorithms for data mining problems.

The comparison to our artefact list:

- **Intellectual property**: GPL information included about the terms and conditions for copy, distribution and modification of the program.
- **Documentation**: a specific documentation for Weka is provided (as pdf in the zip file provided in the QC); Javadoc of the sources is also included
- **Test environment**: Test cases and data are provided but there is no documentation about them. JUnit is used.
- **Design environment**: nothing provided
- **Build environment**: Ant script (build.xml) provided, contains targets for test cases as well
- **Applications**: Source code provided

According to our Sonar run, Weka's comment to code ratio, according to [80], is above average. Most of the Software Escrow violations (991) are triggered by the license plugin, because through the violation it is possible to get the exact license of each source file. The other most violated rule is *magic number*.

- **Non-standard library:** triggered for two libraries out of four, that could not be found in the Maven repository
- **Build**: was able to build Weka with the provided Ant script, no runnable jar war generated though
- **Release**: binaries taken from the QC were used for comparison, which were not identical to the compilation done with Sonar
- **Unsafe calls:** three binary calls found, all of them *Runtime* calls

- **Language detection:** no violations
- **Text proof**: no violations
- **License check (direct output of plugin)**:
  ```
  o   *No copyright* GPL (v2 or later) (with incorrect FSF address): 5 times
  o   GPL (v2 or later) (with incorrect FSF address) GENERATED FILE: 7 times
  o   UNKNOWN: 12 times
  o   GPL (v2 or later) (with incorrect FSF address): 971 times
  o   Public domain: 1 time
  o   *No copyright* UNKNOWN: 3 times
  o   *No copyright* GENERATED FILE: 6 times
  ```

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
28.5 %	3.3 / 38.1 / 51.3	43,356	53.9%	–

4.6.2.2.1.3 Lucene

Apache Lucene[165] is an information retrieval software library that powers Twitter's search function and is used within Solr[166], an open source enterprise search platform. The comparison to our artefact list:

- **Intellectual property**: An Apache License is contained, stating terms and conditions for use, reproduction, and distribution.
- **Documentation**: no specific documentation; although stated in the readme, no Javadoc can be found or is generated during the build process
- **Test environment**: Test cases and a test framework are provided but there is no explicit documentation provided.
- **Design environment**: nothing provided
- **Build environment**: Ant script (build.xml) provided, contains targets for test cases as well
- **Applications**: Source code provided

According to our Sonar run, Lucene's comment density is well above the average of 18.67% measured in [80]. As before most of the Software Escrow related violations are triggered by the licenses plugin due to its functionality.

- **Non-standard library:** not triggered (four libraries to check)
- **Build**: was able to build Lucene with the provided Ant script, runnable jar generated as well
- **Release**: binaries taken from the QC were used for comparison, which were not identical to the compilation done with Sonar
- **Unsafe calls:** none found
- **Language detection:** one violation (a class containing information on character encoding with special characters in the comments)
- **Text proof**: none
- **License check (direct output of plugin)**:
  ```
  o   Apache (v2.0) GENERATED FILE: 7 times
  o   Apache (v2.0): 487 times
  o   *No copyright* GENERATED FILE: 7 times
  ```

[165] http://lucene.apache.org/core/

[166] http://lucene.apache.org/solr/

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
30.5 %	3.1 / 21.8 / 34.6	11,012	70.3%	–

4.6.2.2.1.4 Megamek

Megamek [167] is a network based Java implementation of BattleTech, a turn-based board game for multiple players.

The comparison to our artefact list:

- **Intellectual property**: GPL information included about the terms and conditions for copy, distribution and modification of the program as well as a copyright notice in the readme file.
- **Documentation**: no development documentation, only documentation related to the game itself
- **Test environment**: Test cases are provided but there is no explicit documentation on how to run them.
- **Design environment**: nothing provided
- **Build environment**: Ant script (build.xml) provided, contains no explicit targets for test cases
- **Applications**: Source code provided, data (maps) for the game included

Megamek's complexity compared to other projects is slightly higher, although still below the propagated threshold of 10 when it comes to methods. The rules compliance is quite high. A lot of escrow specific violations belong to the license plugin (1,627), which are triggered due to its functionality.

- **Non-standard library:** none of the 7 libraries was found in the Maven repository
- **Build**: was unable to build the program because of an error in the compilation
- **Release**: could not be compared because build did not succeed
- **Unsafe calls:** none found
- **Language detection:** no violations
- **Text proof**: 161 violations, mostly because sentences did not start with an uppercase letter
- **License check (direct output of plugin)**:

```
o   GPL (v2 or later) GENERATED FILE: 2 times
o   GPL (v2 or later): 1621 times
o   GPL (v2 or later) (with incorrect FSF address): 3 times
o   *No copyright* UNKNOWN: 30 times
```

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
14.7 %	4.7 / 30.9 / 34.2	49,440	71.1%	–

4.6.2.2.1.5 Xalan

Xalan [168] is a XSLT processor for transforming XML documents into HTML, text, or other XML document types.

The comparison to our artefact list:

[167] http://megamek.info/

[168] http://xml.apache.org/xalan-j/

- **Intellectual property**: There is an Apache License included.
- **Documentation**: no development documentation, only documentation related to the tool itself (in xml files)
- **Test environment**: Test cases are provided but there is no explicit documentation on how to run them.
- **Design environment**: nothing provided
- **Build environment**: Ant script (build.xml) provided
- **Applications**: Source code provided

Xalan's comment ratio is above average of the projects in this experiment. Most of the escrow related violations are cause by the used licenses plugin due to its functionality (933).

- **Non-standard library:** triggered for two unknown libraries out of five
- **Build**: was able to build the program with the submitted ant script and produce a distributable jar
- **Release**: binaries taken from the QC were used for comparison, which were not identical to the compilation done with Sonar
- **Unsafe calls:** none found
- **Language detection:** no violation
- **Text proof**: 170 violations, all in one file; again, mostly because a sentence did not start with an uppercase letter
- **License check (direct output of plugin)**:

```
o   Apache (v2.0) GENERATED FILE: 5 times
o   UNKNOWN: 1 time
o   Apache (v2.0): 925 times
o   *No copyright* GENERATED FILE: 2 times
```

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
31.4 %	3.3 / 28.8 / 34.1	25,308	69.2%	–

4.6.2.2.1.6 JFreeChart

JFreeChart provides a chart library, including charts like bar chart, pie chart, scatter plots, histograms, and others. It comes shipped with a Maven build file, thus the test coverage can be generated.
The comparison to our artefact list:

- **Intellectual property**: LGPL information included about the terms and conditions for copy, distribution and modification of the program.
- **Documentation**: The readme contains a reference to the online available installation instructions and a notice where the developer guide can be purchased.
- **Test environment**: JUnit tests are provided and can be run with the included Maven build file.
- **Design environment**: nothing provided
- **Build environment**: Maven build file (pom.xml) and Ant script (build.xml) provided, both contain targets for test cases
- **Applications**: Source code provided

The rule compliance is exceptionally high, with almost 90%. Again, the license plugin triggers most of the escrow related violations (595). Six non-standard libraries related violations and one violation from the invalid release plugin are also found. The most violated rule coming shipped with Sonar is the *magic number* rule (2,605 violations).

- **Non-standard library:** six of the nine libraries were not found in the Maven repository
- **Build**: building was successful, runnable jar created
- **Release**: binaries taken from the QC were used for comparison, which were not identical to the compilation done with Sonar
- **Unsafe calls:** none found
- **Language detection:** no violations
- **Text proof**: no violations
- **License check (direct output of plugin)**:
  - `LGPL (v2.1 or later): 593 times`
  - `LGPL (v2.1 or later) GENERATED FILE: 1 time`

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
33.1 %	2.8 / 38.1 / 39.5	6,830	89.6 %	53.6 %

4.6.2.2.1.7 Derby

Apache Derby[169] is an open source relational database implemented in Java. The comparison to our artefact list:

- **Intellectual property**: An Apache License is contained, stating terms and conditions for use, reproduction, and distribution.
- **Documentation**: The QC version did not contain a documentation folder, but when downloading from the website there is a substantial HTML and PDF documentation included.
- **Test environment**: Tests are provided and a comprehensive documentation on testing issues is included.
- **Design environment**: nothing provided
- **Build environment**: Ant script (build.xml) provided
- **Applications**: Source code provided, although in several subfolders

Comment density is good, whereas the complexity is higher on average than in other tested projects. Most remarkably again is the number of violations triggered by the license plugin (1,729). Almost one quarter of all violations are *magic numbers*, which is again the most violated rule.

- **Non-standard library:** four non-standard libraries were not found in the Maven repository out of six in total
- **Build**: building was successful
- **Release**: binaries taken from the QC were used for comparison, which were not identical to the compilation done with Sonar
- **Unsafe calls:** none found

[169] http://db.apache.org/derby/

- **Language detection:** one violations in JavaCC generated file that did not include many comments
- **Text proof**: no violation triggered
- **License check (direct output of plugin)**:
```
o   Apache (v2.0) GENERATED FILE: 16 times
o   *No copyright* Apache (v2.0): 3 times
o   *No copyright* UNKNOWN: 15 times
o   Apache (v2.0): 1699 times
o   *No copyright* GENERATED FILE: 10 times
```

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
30.2 %	3.4 / 49.3 / 51.3	66,537	68.1 %	-

4.6.2.2.1.8 JasperReports Library

JasperReports Library is a popular open source reporting engine, used to process data coming from different sources and producing pixel-perfect documents that can be viewed, printed or exported in various document formats like HTML, PDF, or OpenOffice.
The comparison to our artefact list:

- **Intellectual property**: GPL information included about the terms and conditions for copy, distribution and modification of the program.
- **Documentation**: a specific documentation is not included; there is data for demos and sample files
- **Test environment**: nothing provided
- **Design environment**: nothing provided
- **Build environment**: Ant script (build.xml) provided, Maven build script (pom.xml) provided
- **Applications**: Source code provided

Compared to other projects the number of comments in the source code is rather low. However, the complexity measurement and the rule compliance, which is one of the highest in the experiments, indicate that the source code was written to be understandable. Again, most of the escrow related violations were triggered by the license plugin (1411).

- **Non-standard library:** triggered for 22 libraries out of 48 that could not be found in the Maven repository
- **Build**: building the project worked, however generating jar files from the build was impossible due to an internal error by the application
- **Release**: runnable jars were not created so the comparison was not possible
- **Unsafe calls:** one binary call found (a *Script* expression gets executed in a method, which triggers the regex search for "*exec")*
- **Language detection:** no violations
- **Text proof**: 24 issues detected, most of them triggered because an alleged sentence did not start with an upper letter
- **License check (direct output of plugin)**:
```
o   UNKNOWN: 3 times
o   LGPL (v3 or later): 1408 times
o   LGPL (v3 or later) GENERATED FILE: 1 time
o   LGPL (v2.1 or later) (with incorrect FSF address): 1 time
```

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage
11.5 %	2.0 / 17.9 / 21.3	12,283	89.6%	–

4.6.2.2.1.9 Sonar

Sonar is the open source tool used for the analysis of the project. Freely adapting Sonar's "eat your own food"[170] policy, we built and analysed the 3.4 branch from Git, because we used version 3.4.1 for our test runs.

The comparison to our artefact list:

- **Intellectual property**: GPL information included about the terms and conditions for copy, distribution and modification of the program as well as a copyright notice.
- **Documentation**: no documentation included; the Javadoc can be found on the website
- **Test environment**: tests are included in the modules
- **Design environment**: nothing provided
- **Build environment**: Maven build script (pom.xml) provided
- **Applications**: Source code provided

A Sonar analysis of itself can be found on the website[171], which states only 233 violations and 98.8% rule compliance. In our runs this excellent result can almost be reproduced, but because our runs were done with our self-implemented plugins additional violations were found. Most of them (1,123) are triggered by the license plugin again. Besides this, only 36 issues in the comment text and one binary call was found.

- **Non-standard library:** Interestingly, Sonar does not contain any jar library we could test.
- **Build**: Building the project worked, for each module a separate jar file was created
- **Release**: no runnable jars for a comparison were created; for the generated jar files we only found one match in the file downloadable from the website[172] (*sonar-applicaton-3.4.1.jar*). After the run the name of the generated jar indicated that we built a snapshot of version 3.4.2. The comparison also showed that the files were not identical.
- **Unsafe calls:** one found (a *ProcessBuilder* expression*)*
- **Language detection:** no violations
- **Text proof**: 36 found, most of them triggered because a sentence did not start with an uppercase letter
- **License check**: Sonar consists of multiple modules and only provides license check results for each single module. The number of licenses in total, counted manually (direct output of plugin):
  - `LGPL (v3 or later): 1176 times`
  - `LGPL (v3 or later) GENERATED FILE: 1 time`

Comments	Complexity (method/class/file)	Violations	Rules compliance	Test coverage

[170] http://www.sonarsource.org/forge/

[171] http://nemo.sonarsource.org/dashboard/index/org.codehaus.sonar:sonar, accessed on 31.01.2013

[172] http://dist.sonar.codehaus.org/sonar-3.4.1.zip

8.5 %	1.8 / 9.6 / 10.5	1,890	97.1%	70.0%

4.6.2.3 Discussion and interpretation of results

This section is split in two parts, according to the previous section containing the experiments and their results. First we will present the results from the experiments done without our Prototype Software Escrow Framework and afterwards we will compare the outcome of the analysis of different projects.

4.6.2.3.1 Feasibility experiments

Code obfuscation promised to be an interesting method of hiding knowledge in source code by making it difficult to understand. Unfortunately the obfuscation method of the used tool was working too well. Thus we were not able to fully decompile the obfuscated sources again.

The **readability of comments** experiments were conducted for the Flesch index as well as for the Flesch-Kincaid grade level. The comments were used as is, including annotations (e.g. for Javadoc or html tags) that make the raw comments less human readable. The normalization on number of comment words is questionable because both metrics already include the average sentence length. Therefore, calculating Flesch or Flesch-Kincaid for comments alone, without normalization, will be enough to evaluate their understandability. Classes with outstanding poor performance regarding their comments will then have to be checked by the human reviewer to decide if they are actually badly written or for instance only contain sentences that metrics falsely interpret to be of bad quality. Examining all comments of Lucene together, the Flesch(-Kincaid) values indicate that a sophisticated user is needed to understand them. This result also includes classes with annotated comment text (e.g. for Javadoc), which decreased the readability values.

Regarding the experiments done with **ratios of comment words or complexity** to code lines the most interesting results were found when using different code lines calculations. The number of code lines found by LOC and NCSS differ unproportionally, e.g. *TestTermVectorsWriter.java* has 384 LOC and 315 NCSS, whereas *Version.java* has 24 LOC and only 4 NCSS. Thus the set of classes found is also not very similar: Leaving out the erroneous classes with zero source code lines, only four of seven classes are in both comments-words:code-lines result sets containing the poor performing classes. The result set of the best classes are overall the same, however they are not as interesting for our purpose. The complexity:code-lines result sets show the same problems. It is necessary to agree on one method for counting lines before evaluation.

For both LOC and NCSS result sets, many of the best performing classes contain less than 10 code lines. We argue that, compared to huge classes, only a few code lines in small classes do not cause understanding problems. By using a threshold, small or erroneously found classes can be sorted out from the evaluation process.

4.6.2.3.2 Prototype experiments

For the Software Escrow Prototype experiments, different open source projects were used, starting from about 60,000 to 520,000 LOC. A comparison to our artefact list shows that the range of maintainability related artefacts is rather low. This applies on one hand to the usage of free available software with no agreement on the submitted artefacts, as would be usual in a customer-furnished software agreement, and

on the other hand we assume that their developers do not have a lot of resources for costly maintainability tasks.
All of the projects under test contained information on the license used, usually in a file in the top level directory. Still, as our license plugin showed, some classes within the project did contain unknown license information or no copyright at all. Two-thirds of the projects provided a documentation, which was not tested due to the diversity in format and the limit of Sonar's plugins. It is interesting to note that one project, JFreeChart, only provided a paid-for developer guide. Tests were available for almost all projects, although Sonar was only able to evaluate the test cases of the two Maven projects. We plan on implementing a plugin to analyse JUnit tests of other software as well. If test cases were available, a corresponding target in the build script was also provided. Regarding design environment no project included anything related to this artefact. It is probable that the developers working on the source code do not have a dedicated design document due to lack of resources. For building the projects, Ant scripts are the most typical information provided. The build artefacts are lacking the compiler used for build, which leads to problems when it comes to comparing the built results with the release results (see below). Regarding the application's artefacts all projects provided their source code. Only Megamek needs to have maps for the game additionally to the source code for full functionality, which were also included.

Analysing commonly used quality metrics is not in the focus of this work. Nevertheless we will give an overview of these results as well to show that they are not significant enough for the Software Escrow use case.
Commonly used quality metrics, including comment density, complexity on different levels, Unit test coverage, as well as violations to rules or rule compliances found by Sonar offer a comparison between projects. The range of how many comments are included in a project differs widely, not depending on the project size. Projects with more LOC, like Azureus or Megamek, have a rather low **comment density**, compared to Derby that includes many more comments. Sonar on the other hand is the smallest project tested and has a comment density under 10%. The density of all projects is 21.8% on average. These findings reflect the results of [80], where an average comment density of 25.87% was found for Java open source projects. Looking only at the quantity of comments, quality is hard to evaluate as the percentage does not reflect the content of the comments. If the framework does not take into account license information in comments at the beginning of a class, then long license headers will increase the comment density. Because Sonar only measures the number of comment lines, a fixed number of characters per line has to be used in order to compare the number of comment lines across projects. One way to come by this problem is to agree on a comment line length in the escrow contract. Another approach could include formatting the source code to a pre-defined format before analysis. As formatting source code is a common feature in development environments to automatically convert the source code to a desired style, this step can quickly be done.
Regarding **complexity**, the source code quality can be measured more easily. More complexity means more effort for understanding. Sonar calculates complexity on different levels. For methods, this lies below complexity of five on average, but looking at class complexity the projects diverge much more. For instance the average method complexity in Azureus is higher than for JFreeChart, but when it comes to classes the

second project gets more complex. Complexity on file level is usually higher than on class level, which leads to the assumption, that when using more than one class per file the code gets more difficult to understand. Compared to McCabe's stated upper limit of ten (cf. Section 4.5.1) for complexity, all projects, or at least their methods, lie below this on average. Because there is no reliable limit as to when a source code definitely gets too complex to understand, this metric can only be used to give a rough overview of source code quality.

The number of **violations** reflects how often coding best practices are disregarded. The **rule compliance** supports this number by normalizing it to the size of the source code. When our text proofing plugin was turned on without setting its attributes to reasonable numbers, the violations increased enormously. Thus when implementing plugins, we have to make sure that the number of violations found does not exceed a certain limit. Otherwise the results get confusing. Following this recommendation, the rule compliance is able to show whether the programmer adhered to common coding practices and where improvement is needed the most.

The **test coverage** for JFreeChart is lower than for Sonar, probably because the former contains more LOC and writing tests is expensive. As mentioned in Section 4.5.1, the impact of code coverage is controversial[173]. Besides, we only have a sample of two, which is not significant enough for a conclusion.

Our plugins were tailored for analysing software put into escrow with regard to maintainability after their re-deployment. Other than the abovementioned standard quality metrics, these help in finding artefacts that need further attendance by a human reviewer.

The **non-standard library** plugin worked as expected. It found libraries that were not listed in the Maven repository. We were not able to test the National Software Reference Library because the projects tested did not use them. A functionality check of the plugin, if the repository actually does not know the library, turned out to find the library, although only when searching for its name. However, on a binary level the two files differed. We assume that in open source projects programmers do not want to hide code on purpose in libraries. A possible explanation for these incidences can be that the (also open source) libraries were compiled before deployment and thus differ slightly from the ones available in the Maven repository. For our purpose this behaviour is desirable however. Altered libraries have to be found, even if their names are the same. On the other hand, there are libraries that do not need further attendance, because they are not different to the ones stored in the Maven repository. This reduces the time for the review.

The **build** plugin was able to compile most of the projects. In some cases no runnable jar was created. All of the build errors could be repeated when doing the compilation without Sonar. Thus our plugin worked accurate.

The **release** plugin directly depends on the build plugin, as it compares any release binary to the output of the compilation. Although preliminary tests indicated that a compilation with different machines and the same compiler should lead to identical libraries, we were not able to reproduce this result in the prototype experiments. We compared the compiled binaries to the jar files provided in the QC when using the sources

[173] Testivus wisdom on code coverage can be found here: http://googletesting.blogspot.co.at/2010/07/code-coverage-goal-80-and-no-less.html

from there as well, and to binaries downloadable from the internet. Even if the jar files' sizes were the same, as it happened in the Derby experiment, the checksum of the files were not equal. We checked the results more thoroughly and came to the conclusion that a different compiler was used than in our system setup. As the class files did not differ in the header section, which would suggest that they were compiled by different compiler versions, it is possible that e.g. an open source implementation of the Java Platform like OpenJDK[174] was used. This indicates the importance of knowing which compiler was used for the development.
The **unsafe calls** plugin listed classes that contained runtime or similar calls. Even if it triggers wrongly, it can save a reviewer a lot of time because with Sonar it is quickly possible to inspect the source of the violation and mark it as false positive.
The **language detection** plugin was set to analyse the comments on class level. With the additional dictionary based approach turned on, the quality of the results could be increased.
Text proofing was done for each project with a third party tool, embedded in a Sonar plugin. However it quickly turned out that without thresholds, it increases the number of violations found tremendously. Thus we decided to implement different attributes that e.g. sorted out Javadoc, ignored the file header (containing license information), or added only classes with a higher error count to the result set. The most frequently found errors were sentences not starting with an upper case letter, a writing style rather common in comments. We would have to exclude the entire capitalization category to filter out this error. Because upper case letters are important for readability, we decided to leave these findings in the result set.
The **license check** plugin was able to extract the licenses used in the source files, as well as find classes that did not contain a license and/or were generated. Thus it happens that classes with a name consisting of "generate" get accidentally labelled as being generated. Because classes with no known copyright notice are rare in our test set, it is still feasible to manually look them through and use the plugin only as an indicator. Regarding the licenses of libraries, in many cases these could not be identified, because no license information was provided. Our plugin had a minor bug which caused problems to view which license a library uses. Because it could not be fixed in time we left out the information on the library's licenses.

4.7 Conclusion and outlook

The previous section analysed details necessary for a successful Software Escrow. We examined legal and technical considerations related to the entire process. The three phases of escrow were discussed and analysed from both points of view, beginning with planning and setting up an agreement, executing the escrow by depositing the material at the escrow agent and evaluating it, and finally redeploying the software.

The planning phase greatly involves legal and economic aspects. A cost-benefit analysis helps to evaluate if the software is crucial for the execution of a business process and therefore needs to be preserved. Requirements for selecting the right escrow agent and the data security requirements needed are part of this phase as well. The contents of the material to be deposited in order to guarantee the maintainability

[174] http://openjdk.java.net/

also have to be agreed on in this phase. Part of this also involves the clarification of licenses and rights to the deposited artefacts.

The deposit procedure of the escrowed material was described in the execution phase. From a legal point of view the thresholds for quality turned out to be crucial. If verifying the deposited software, it is important to agree upon a level of quality that has to be fulfilled. Extending existing verification procedures, which were described shortly, we proposed a framework. Thus a great part in the execution phase involves the verification of the deposited material. The quality of the escrowed artefacts has to be verified in order to guarantee their full redeployability once they get used again. To support this quality verification part a framework was proposed, based on our investigation of the related work. This Technical Framework for Software Escrow is able to automatically check all deposited artefacts related to the software in escrow. The quality evaluation focuses mainly on maintainability to guarantee that the software can be developed once it gets redeployed. For a technical expert reviewing the software, the framework is able to support his work by pre-screening the material and reporting artefacts that need further investigation. The goal is to decrease the time a costly human reviewer takes to evaluate hundreds of source files and other artefacts. For better extensibility the framework was designed to include plugins for different calculations. These include commonly used quality metrics like the number of code lines in a source code or its complexity. Other plugins specific for Software Escrow were also proposed, including checks for external dependencies like Web service calls or for functionality hiding.

The last phase of Software Escrow, the redeployment, involves only legal aspects again. The release of the escrowed materials is the main part. We covered different events that trigger the release of the software, including bankruptcy of the developer, unjustified refusal of maintenance, or unjustified denial of adaption. How to advance in such cases was also described in detail. Different information obligations by the software user and escrow agent concerning the trigger events were mentioned.

To evaluate the usage of the theoretical basis for the Technical Framework for Software Escrow, a prototype was implemented and tested with different projects. For each project, completeness of artefacts and escrow specific quality of the artefacts were evaluated. The results showed that many projects were not built to take into account short or long term preservation. For our experiments this helped testing all plugins, however.

Although being open source, not all of the projects provided a proper documentation needed for understanding and developing the source code. Documentation in natural language, that explains how the software is constructed, is the most essential artefact for comprehension. When setting up an escrow contract, the scope of documentation has to be clearly stated. As our results show it is not enough to agree on a quantitative attribute, e.g. the comment density in the source code. The quality and therefore the content of the annotations have to be evaluated. To support this check we worked on a readability tool. Analysing the results indicated that pre-processing the text before analysis, e.g. stripping special mark-ups from the comments, will improve the reliability of the calculations. Complexity of source code or other metrics could also not be identified as reliable indicators for quality in deposited material. Plugins looking for Software Escrow specific problems like hidden code in libraries or calls to external services were able to highlight potential troublesome artefacts and point them out appropriately, e.g. by explaining what kind of call triggered the violation and where exactly to find it. The license check was also successful on file and library level. We were able to identify classes without a proper license and check the licenses of libraries. These finding can be compared to the licenses agreed on in the contract and objected to when necessary.

The Technical Framework for Software Escrow was intended to support the software review by pointing out artefacts and parts of artefacts that needed further investigation. With the theoretical basis provided in this deliverable and the implementation of the prototype we were able to give a proof of concept of such a framework. For future work in the readability domain we plan on evaluating if standard readability metrics need to be adapted to be more suitable for special linguistic features found in source code comments. We are also planning on making the readability tool part of the prototype framework to fulfil the holistic approach. Looking for a formula to combine plugin results in order to represent the deposit-quality of all artefacts will also be part of our future work.

To improve the framework in future work, we propose to address the following goals:

- develop improved metrics
- integration of a high level quality concept to
  - increase the degree of automation in the assessment
  - keep the understanding of quality more stable over time
  - improve the degree of integration between the software development processes and the Software Escrow

For future work in the readability domain, the evaluation should be forced, if standard readability metrics need to be adapted to be more suitable for special linguistic features found in source code comments and other collaterals needed for redeploying a software development process. The prototype framework can then be extended with the newly realized readability tool.

The developed and implemented metrics are already a useful basis for the assessment of the product’s insights by an expert. With the implementation of a high level quality concept, the contractual basis is expected to be more robust and the effort for the quality verification will be reduced by a higher degree of automation. The idea of the quality concept will be illustrated in the following, to give a more concrete impression of the planned improvements.

As described above, with the implemented framework every result needs to be interpreted by an expert. In the concept suggested for the future, quality will be defined in an early stage of the project. The implicit understanding of maintenance quality used in the current framework could be made more objective by the usage of a well-defined quality model. The benefit of an objective quality model is, that it can be part of the software escrow contract and that it can be interweaved with the quality models already used in the development projects. Quality models are structured in a hierarchy, refining the quality concepts from the general term quality to the fine granular leaves named quality attributes. The quality attributes in the level *n* will be refined in the tree level *n+1*. The quality in the level *n* will be aggregated by the weighted quality of the root of its sub tree.

With the resulting homogenous understanding of quality in a project, a consistent basis for balancing interests of the different stakeholders is defined. These stakeholders can be interested in the Software Escrow as well as in the original development project. It is obvious that a consistent taxonomy for quality will improve the integration of Software Escrow in the development. The applicability of quality models like the ISO 9126 [97] is proven in different development processes like the V-Model 97.
In the meantime, the ISO 9126 was replaced by its successor the ISO25010 [98]. The ISO 25010 defines two quality models. The quality model for quality in use and one quality model for the product quality. For the Software Escrow framework the product quality model is expected to fit perfect. The structure of the product quality model is illustrated in Figure 37 and a detailed description of the quality attributes can be found in the ISO 25010.

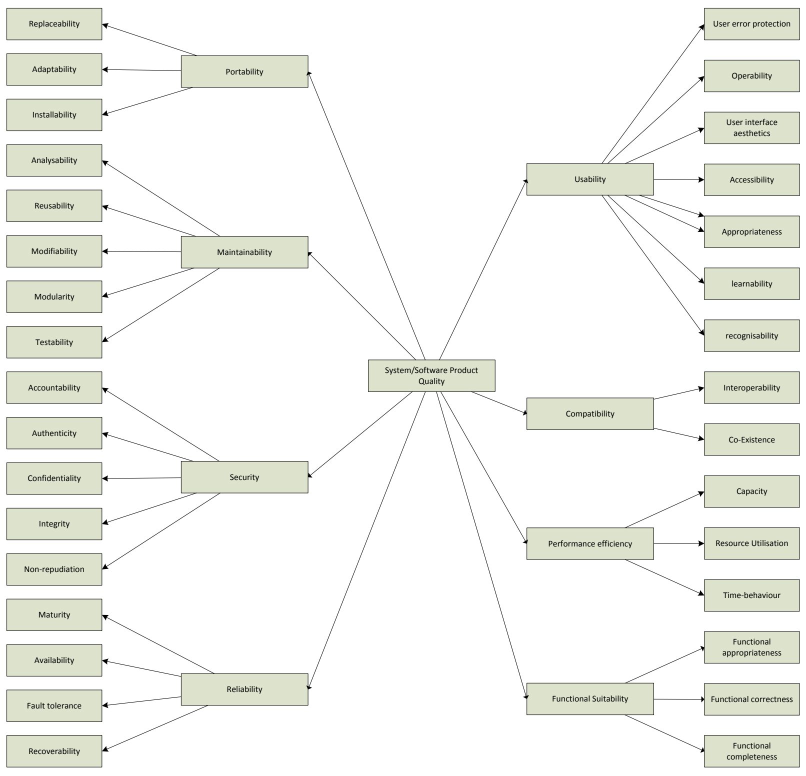


Figure 37: ISO 25010 Product Quality Model

Besides the fact, of the simpler integration of the Software Escrow it is expected that the escrow goal can be defined much easier and more flexible. The quality characteristics (or attributes) will be weighted depending on the escrow goal. For example a Software Escrow addressing the goal to only rerun the software (e.g. for forensic analysis) may be mainly interested in compatibility. Another potential escrow goal may be the reuse of the software. In this case, functional suitability and usability may be the characteristics of interest. But in the most probable case, maintainability (like it is addressed by the current framework) is the quality characteristic of interest.

Even if quality models are helpful in defining a common understanding of the term quality, they don't solve the problem that the application of the quality model on a project is still subjectively in its quantitative shaping. A well-established practice for improving the objectivity and reducing the manual effort is the usage of bi-directional quality models. They combine metrics, collected by tools like they are described earlier in this work, with the quality models like the product quality model from ISO 25010. The glue between the relatively volatile metrics and the stable quality model are called quality indicators. They map a set of metrics to a concrete quality attribute and give them a concrete meaning. The schematic concept is illustrated in Figure 38.

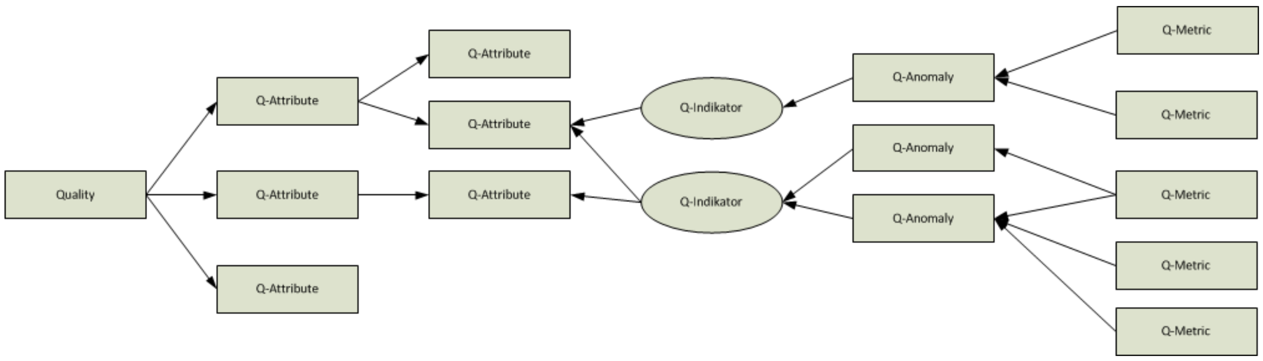


Figure 38: Schematic bi-directional quality model

The quality models are maintained and adjusted during the complete systems lifetime. That's caused by the changes in the metrics, changes in the measuring tools and by growing experience. Changing quality goals also influence the quality model and drive changes of the quality models. For example if Software Escrow has to be contracted for an existing project, the quality model changes or extends its quality focus.
In summary, the combination of often changing metrics with stable quality models will lead to a more systematic and consistent assessment of quality for the long term.

5 Conclusion and outlook

The goal of TIMBUS is the preservation of business processes for the long term. This deliverable presents the TIMBUS process framework that species the process steps to digitally preserve business processes (presented in Section 3). A special problem for the preservation of processes is posed by the use of external services. Section 4 presents a novel approach to address the preservation of external dependencies by using Software Agreement.

The TIMBUS process framework guides through the three phases of the TIMBUS approach: plan, preserve and redeploy. Driven by an enterprise risk management perspective, Digital Preservation is considered as mitigation strategy to address potential loss of information over time. Thus potential risks associated with business processes are identified and approached. The TIMBUS project provides methods, guidelines and tools to document, acquire, preserve, and later redeploy processes. To preserve the process for the future, its influence factors and implementation need to be understood. Hence the context of the process is acquired addressing the business, application, and technology layer. Relevant aspects of the process are identified that need to be maintained for the future. Potential preservation strategies are identified and tested considering the specific conditions, requirements, and goals of a setting. Different approaches can be combined to maintain the significant properties of the process over time. Business processes are often implemented by using a service-oriented architecture implemented on a distributed infrastructure. The deliverable discusses preservation approaches that address the specific needs of process preservation such as the preservation of external services. Escrow agreements for example provide a business model to ensure access and usage of third party services over time. Driven by the risk management perspective, our framework allows for potential solutions to be evaluated for their abilities to treat identified risks. Furthermore, economic aspects are taken into account. The execution phase performs required preservation actions and prepares the process for archival storage. The redeployment reruns the archived process in a new environment in some time in the future. The process needs to be adapted according to the conditions of the new environment, which affects technical as well as organisational aspects. The behaviour of the redeployed process needs to be validated by comparing its measurements with the measurements taken from the original process.

The TIMBUS process framework provides a guideline specifying the required steps to plan and perform the preservation and the later redeployment. The framework provides high degree of flexibility and customizability for applying it in different domains and settings. It specifies the fundamental concepts of the process steps by specifying the input and the resulting output of each phase. Thus the implementation of the process steps can be realised for the specific needs of the setting. The framework should allow an organisation to implement the preservation process according to their special requirements, while still ensuring that all relevant aspects of the preservation process are addressed. The organisational implementation of the framework is guided through a generic stakeholder specification and assignment in the framework.

The process framework is used for the TIMBUS use cases, WP8 – “Civil Engineering” and WP9 – “eHealth”. This deliverable shows examples of the use cases for the instantiation of the process steps. Through the flexibility of the framework both processes can be supported without any bigger adjustment, even though the two use cases present different scenarios with different focuses, goals, and implementations. The work on use cases is continued in WP8 and 9, which follow the process framework of this deliverable.

Within this deliverable the modules of the TIMBUS architecture that support the process step are identified. These tools help to reduce the effort for the preservation process steps. Examples for tools in development are extraction and population tools for the context model (e.g. installed software on system) or recommender services for potential virtualisation and emulation strategies. Further existing tools, e.g. virtualisation tools supporting the preservation process, are identified and tested within WP6 and the use cases. Additional work is done on a framework to validate and verify the behaviour and characteristics of redeployed processes within TIMBUS. The results will be presented in the upcoming deliverable *D4.7 Validation of Digitally Persevered Business Processes & Verification of Exhumed Business Processes*.

The second part of this deliverable aimed at analysing details necessary for a successful Software Escrow. For this purpose we examined legal and technical considerations. We also looked into the three different phases of Software Escrow, beginning with planning and setting up an agreement, depositing the escrow material and evaluating its quality with regards to Digital Preservation, and finally redeploying the software.

From a legal point of view, the planning and redeployment phases are the most important. Careful drafting of the contract in terms of the circumstances is needed for a successful escrow agreement that becomes effective under certain trigger events. This starts with specific requirements that need to be taken into account when selecting the escrow agent. What material to put into escrow has to be agreed upon as well, because if something is missing after the redeployment, then the software will become unmaintainable. Furthermore, possible release events and how to proceed when releasing the software from escrow were discussed, including information obligations by the involved parties.

The deposit procedure in the execution phase involves a greater technical part, i.e. the verification of the deposited material. Thresholds, stipulated in the contract, are needed to make it possible to agree on a certain level of software quality. For supporting the quality verification part in the execution phase we proposed a Technical Framework for Software Escrow to highlight artefacts that do not comply with the desired quality level. The theoretical basis for this purpose included how to evaluate the deposited software to guarantee that its quality supports maintainability needed to develop and maintain it in the future. To support these considerations we implemented a prototype including the previously stated evaluation mechanisms. Tests were conducted to verify the functionality and feasibility of the prototype. With these results we were able to give a proof of concept regarding our framework.

Our next steps include the application on larger software with more relation to real-world scenarios. We will evaluate the use cases presented in WP 8 and 9 (civil engineering and eHealth, respectively) with our framework. If we can show where problems occur if the software was put in escrow, we will be able to contribute to its overall preservability evaluation.

References

[1] TIMBUS consortium, *D8.1: "Use Case Definition and Digital Preservation Requirements",* 2012.

[2] TIMBUS consortium, *D9.1: "Use Case Definition and Digital Preservation Requirements",* 2012.

[3] ISO/IEC, "ISO 14721:2012 - Space data and information transfer systems - Open archival information system (OAIS) - Reference model," 2003.

[4] ISO/IEC, *ISO 31000:2009 - Risk management-principles and implementation of risk management,* 2009.

[5] IEEE Computer Society, "IEEE Standards 1012-IEEE Standard for Software," IEEE, New York, 2005.

[6] TIMBUS consortium, *D5.5: "Refined Preservation Architecture",* 2012.

[7] *Business Process Model and Notation (BPMN),* http://www.omg.org/spec/BPMN/2.0/, accessed 14.03.2013.

[8] The Open Group, "ArchiMate® 1.0 Specification," 2009.

[9] TIMBUS consortium, *D4.9: Refined Business Process Contexts.*

[10] TIMBUS consortium, *D4.2: "Dependency Models Iter. 1",* 2012.

[11] TIMBUS consortium, *D4.5: "Business Process Contexts",* 2012.

[12] S. Higgins, "The DCC Curation Lifecycle Model," *The International Journal of Digital Curation,* vol. 3, pp. 134-140, 2008.

[13] H. Brocks, A. Kranstedt, G. Jaschke and M. Hemmje, "Smart Information and Knowledge Management," *Studies in Computational Inteligence,* pp. 197-226, 2010.

[14] E. Conway, B. Matthews, D. Giaretta, S. Lambert and M. Wilson, "Managing Risks in the Preservation of Research Data with Preservation," *The International Journal of Digital Curation,* vol. 7, pp. 3-15, 2012.

[15] D. Marcum, "The preservation of digital information," *The Journal of Academic Librarianship,* vol. 22, pp. 451-454, 1996.

[16] The SCAPE Project, *Identification and selection of large-scale migration tools and services (D10.1),* http://www.scape-project.eu/wp-content/uploads/2011/09/SCAPE_D10.1_KEEPS_V1.0.pdf, accessed 27.02.2013.

[17] S. Granger, *Emulation as a Digital Preservation Strategy,* http://www.dlib.org/dlib/october00/granger/10granger.html, accessed 27.02.2013.

[18] J. van der Hoeven, B. Lohman and R. Verdegem, "Emulation for Digital Preservation in Practice: The Results," *The International Journal of Digital Curation,* vol. 2, pp. 123-132, 2008.

[19] J. Rothenberg, *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation,* http://www.clir.org/pubs/reports/rothenberg/contents.html, accessed 27.02.2013.

[20] H. K. Christoph Becker and A. Rauber, "Trustworthy Preservation Planning with Plato," *ERCIM News,* vol. 80, pp. 24-25, 2010.

[21] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. M. Miles, M. Paolo, P. B.

Jim, Y. Simmhan, E. Stephan and J. Van den Bussche, "The Open Provenance Model core specification (v1.1)," Elsevier, 2010.

[22] TIMBUS consortium, *D4.3: "Dependency Models Iter. 2",* 2013.

[23] TIMBUS consortium, *D6.3: "Legalities Lifecycle Management",* 2013.

[24] TIMBUS consortium, *D4.8: "Refined DP & Intelligent Enterprise",* 2013.

[25] TIMBUS consortium, *D4.1: "DP & Intelligent Enterprise Risk Management",* 2012.

[26] ISO/IEC, *Guide 73. Risk management. Vocabulary. Guidelines for use in standards.,* 2002.

[27] TIMBUS consortium, *D8.2: "Use Case Specific Risks".*

[28] TIMBUS consortium, *D9.2: "Use-Case Specific Risks".*

[29] Object Management Group, "Business Process Model and Notation (BPMN)," 2011.

[30] IEEE Task Force on Process Mining, "Process Mining Manifesto," 2011.

[31] The Open Group, "TOGAF® Version 9.1," 2011.

[32] TIMBUS consortium, *D6.5: "Populating and Accessing Context",* 2013.

[33] M. A. Neumann, H. Miri, J. Thoms, G. Antunes, R. Mayer and M. Beigl, "Towards a Decision Support Architecture for Digital Preservation of Business Processes," in *International Conference on Preservation of Digital Objects (iPres2012)*, 2012.

[34] R. Treinen and S. Zacchiroli, "Description of the CUDF Format," *CoRR,* vol. abs/0811.3621, 2008.

[35] C. Becker, H. Kulovits, M. Guttenbrunner, S. Strodl, A. Rauber and H. Hofman, "Systematic planning for digital preservation: evaluating potential strategies and building preservation plans," *International Journal on Digital Libraries,* vol. 10, no. 4, pp. 133-157, 2009.

[36] B. Matthews, A. Shaon, B. Juan and C. Jones, "Brian Matthews, Arif Shaon, Juan Bicarregui and Catherine Jones," *The International Journal of Digital Curation,* vol. 5, no. 1, pp. 91-105, 2010.

[37] M. Guttenbrunner, "Preserving Interactive Content: Strategies, Significant Properties and Automatic Testing," in *Workshop on Data Analysis (WDA'2008)*, Dedinky, Slovakia, 2008.

[38] M. Guttenbrunner and A. Rauber, "A Measurement Framework for Evaluating Emulators for Digital Preservation," *ACM Transactions on Information Systems (TOIS),* vol. 30, no. 2, 3 2012.

[39] G. Knight and M. Pennock, "Data Without Meaning: Establishing the Significant Properties of Digital Research," *The International Journal of Digital Curation,* vol. 4, no. 1, pp. 159-174, 2009.

[40] InSPECT consortium, "Final Report," 2009.

[41] P. Trezentos, I. Lynce and A. L. Oliveira, "Apt-pbo: solving the software dependency problem using pseudo-boolean optimization," in *ASE '10 Proceedings of the IEEE/ACM international conference on Automated software engineering*, New York, NY, USA, 2010.

[42] J. A. Stafford, D. J. Richardson and A. L. Wolf, "Chaining: A Software Architecture Dependence Analysis Technique," Technical Report CU-CS-845-97, Department of Computer Science, University of Colorado, 1997.

[43] J. A. Stafford and A. L. Wolf., "Architecture-level dependence analysis in support of software

maintenance," in *Proceedings of the third international workshop on Software architecture (ISAW '98)*, New York, NY, USA, 1998.

[44] PREMIS Editorial Committee, "PREMIS Data Dictionary Version 2.2," 2012.

[45] T. Miksa, R. Mayer and A. Rauber, "Ensuring Sustainability of Web Services Dependent Processes," *International Journal of Computational Science and Engineering (IJCSE),* 2013.

[46] TIMBUS consortium, *D9.3: "Definition of an eHealth business process and requirements specification for Digital Preservations",* 2013.

[47] S. Strodl, P. Petrov and A. Rauber, " Research on Digital Preservation within projects co-funded by the European Union in the ICT programme," 2011.

[48] BCG The Boston Consulting Group, *M&A: Ready for Liftoff? A survey of European Companies' Merger and Acquisition Plans for 2010,* 2009.

[49] D. Draws, S. Euteneuer, D. Simon and F. Simon, "Short Term Preservation for Software Industry," in *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPres 2011)*, 2011.

[50] TIMBUS consortium, *D4.4: Legal/Contractual process for execution and exhumation - Overview.*

[51] *ESCROWGUIDE - Source Code Escrow - Guidelines for Acquirers, Developers, Escrow Agents and Quality Assessors - Part 2: A guide to using source code escrow to protect procurement,* 1999.

[52] *ESCROWGUIDE - Source Code Escrow - Guidelines for Acquirers, Developers, Escrow Agents and Quality Assessors.*

[53] D. Hovemeyer and W. Pugh, "Finding bugs is easy," in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, New York, NY, USA, 2004.

[54] D. Low, "Protecting Java code via code obfuscation," *Crossroads,* vol. 4, no. 3, pp. 21-23, #apr# 1998.

[55] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proceedings of the 10th ACM conference on Computer and communications security*, New York, NY, USA, 2003.

[56] N. A. Naeem, M. Batchelder and L. Hendren, "Metrics for Measuring the Effectiveness of Decompilers and Obfuscators," in *Proceedings of the 15th IEEE International Conference on Program Comprehension*, Washington, DC, USA, 2007.

[57] F. Simon, O. Seng and T. Mohaupt, Code-Quality-Management - technische Qualit{\"a}t industrieller Softwaresysteme transparent und vergleichbar gemacht, dpunkt.verlag, 2006, pp. I-XVII, 1-340.

[58] P. Hamill, Unit test frameworks, First ed., M. Hendrickson, Ed., O'Reilly Media, Inc., 2004, p. 4 ff..

[59] R. Allen, "A Formal Approach to Software Architecture," 1997.

[60] P. Weirich, Decision Space: Multidimensional Utility Analysis, Cambridge University Press, 2001.

[61] H. Zhang, "An investigation of the relationships between lines of code and defects," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009.

[62] R. Vivanco and N. Pizzi, "Identifying effective software metrics using genetic algorithms," in *Electrical*

and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on, 2003.

[63] T. J. McCabe, “A Complexity Measure,” *IEEE Transactions on Software Engineering,* vol. 2, no. 4, pp. 308-320, December 1976.

[64] T. Williams, M. Mercer, J. Mucha and R. Kapur, “Code coverage, what does it mean in terms of quality?,” in *Reliability and Maintainability Symposium, 2001. Proceedings. Annual*, 2001.

[65] Sonar, *What makes Checkstyle, PMD, Findbugs and Macker complementary ?.*

[66] R. P. Buse and W. R. Weimer, “Learning a Metric for Code Readability,” *IEEE Transactions on Software Engineering,* vol. 36, no. 4, pp. 546-558, July/August 2010.

[67] R. P. Buse and W. R. Weimer, “A metric for software readability,” in *Proceedings of the 2008 international symposium on Software testing and analysis*, New York, NY, USA, 2008.

[68] D. Posnett, A. Hindle and P. Devanbu, “A simpler model of software readability,” in *Proceedings of the 8th Working Conference on Mining Software Repositories*, New York, NY, USA, 2011.

[69] M. H. Halstead, Elements of Software Science (Operating and programming systems series), New York, NY, USA: Elsevier Science Inc., 1977.

[70] N. S. Coulter, “Software Science and Cognitive Psychology,” *IEEE Transactions on Software Engineering,* vol. 9, no. 2, pp. 166-171, March 1983.

[71] D. Coleman, D. Ash, B. Lowther and P. Oman, “Using Metrics to Evaluate Software System Maintainability,” *Computer,* vol. 27, no. 8, pp. 44-49, August 1994.

[72] Z. Li and Y. Zhou, “PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code,” in *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, New York, NY, USA, 2005.

[73] C. Le Goues and W. Weimer, “Measuring Code Quality to Improve Specification Mining,” *IEEE Transactions on Software Engineering,* vol. 38, no. 1, pp. 175-190, January 2012.

[74] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia and E. Merlo, “Recovering Traceability Links between Code and Documentation,” *IEEE Transactions on Software Engineering,* vol. 28, no. 10, pp. 970-983, October 2002.

[75] W. H. DuBay, *The Principles of Readability,* 2004.

[76] R. Flesch, *How to Write Plain English,* http://www.mang.canterbury.ac.nz/writing_guide/writing/flesch.shtml, accessed 13.03.2013.

[77] L. Tan, D. Yuan, G. Krishna and Y. Zhou, “/*icomment: bugs or bad comments?*/,” in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, New York, NY, USA, 2007.

[78] N. Khamis, R. Witte and J. Rilling, “Automatic quality assessment of source code comments: the JavadocMiner,” in *Proceedings of the Natural language processing and information systems, and 15th international conference on Applications of natural language to information systems*, Berlin, Heidelberg, 2010.

[79] R. Witte, B. Sateli, N. Khamis and J. Rilling, “Intelligent software development environments:

integrating natural language processing with the eclipse platform," in *Proceedings of the 24th Canadian conference on Advances in artificial intelligence*, Berlin, Heidelberg, 2011.

[80] O. Arafat and D. Riehle, "The commenting practice of open source," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, New York, NY, USA, 2009.

[81] D. M. German, P. C. Rigby and M.-A. Storey, "Using evolutionary annotations from change logs to enhance program comprehension," in *Proceedings of the 2006 international workshop on Mining software repositories*, New York, NY, USA, 2006.

[82] W. B. Cavnar and J. M. Trenkle, "N-Gram-Based Text Categorization," in *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 1994.

[83] G. Heyer, U. Quasthoff and T. Wittig, Text Mining: Wissensrohstoff Text, Balzert, Ed., Bochum: W3L Verlag, 2005.

[84] M. Miłkowski, "Developing an open-source, rule-based proofreading tool," *Softw. Pract. Exper.,* vol. 40, no. 7, pp. 543-566, 2010.

[85] T. Lavergne, T. Urvoy and F. Yvon, "Filtering artificial texts with statistical machine learning techniques," *Language Resources and Evaluation,* vol. 45, pp. 25-43, 2011.

[86] J. J. Marshall, "Reuse Readiness Levels as a Measure of Software Reusability," in *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008.*, Boston, Massachusetts, USA, 2008.

[87] SonarSource SA, *Java Metric Definitions,* http://docs.codehaus.org/display/SONAR/Java+Metric+Definitions, accessed 15.11.2012.

[88] C. C. Lee, *JavaNCSS - A Source Measurement Suite for Java,* http://www.kclee.de/clemens/java/javancss/, accessed 15.11.2012.

[89] F. Mallet, *Why you should (not?) upgrade to Sonar 1.9,* http://www.sonarsource.org/why-you-should-not-upgrad-to-sonar-19/, accessed 15.11.2012.

[90] Sonar, *Manage Duplicated Code with Sonar,* http://www.sonarsource.org/manage-duplicated-code-with-sonar/, accessed 26.02.2012.

[91] InfoEther Inc., *Design,* http://pmd.sourceforge.net/pmd-5.0.1/rules/java/design.html, accessed 15.11.2012.

[92] SonarSource SA, *SonarSource - sslr,* https://github.com/SonarSource/sslr/, accessed 10.1.2013.

[93] knallgrau GmbH, *Java Text Categorizing Library,* http://textcat.sourceforge.net/, accessed 10.1.2013.

[94] Sonatype Inc., *The Search Engine for The Central Repository,* http://search.maven.org/, accessed 10.01.2013.

[95] D. Naber, *LanguageTool - Style and Grammar Checker,* http://www.languagetool.org/, accessed 10.01.2013.

[96] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton and J. Noble, "Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies," in *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, 2010.

[97] *International Standard ISO/IEC 9126, Part 1, Software engineering – Product quality – Quality model,* 2001.

[98] ISO, ISO 25010: Systems and software engineering —Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 2010.