TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D6.1

Collaboration procedures and configuration management roles for development

Martin Murth
Gerson Joskowicz
eva Kühn

October 2, 2006

Executive Summary

This deliverable is split into two main parts:

First, it analyses the state-of-the-art of current software development models, both classical software development as well as software development in geographically distributed online development teams. It then proposes a concrete development model for TripCom that follows successful approaches known from commercial and open source software development practices.

Second, the deliverable identifies the critical success factors for the project. These factors are categorised as scientific/technical, economic, and organisational and will serve as a means to control and evaluate the progress of the project.

Document Information

IST Project Number	FP6 – 027324	**Acronym**	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	6.1	**Title**	Collaboration procedures and configuration management roles for development
Work Package	**Number**	6	**Title**	Triple Space Architecture and Component Integration

Date of Delivery	**Contractual**	M6	**Actual**	31-September-06
Status	version 1.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Martin Murth (TUW) Gerson Joskowicz (TUW) eva Kühn (TUW)			
Resp. Author	Martin Murth		**E-mail**	mm@complang.tuwien.ac.at
	Partner	TUW (Vienna University of Technology)	**Phone**	+44 (30) 838-75225

Abstract (for dissemination)	This deliverable has two main foci: First, it analyses state-of-the-art software development models and proposes a concrete model for the TripCom development process. Second, the critical success factors for the entire project are identified.
Keywords	Collaboration procedure, software development processes, critical success factors

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2006-05-15	1	Martin Murth	First draft structure
2006-05-25	2	Martin Murth	Added state-of-the-art of SD models
2006-06-15	3	Geri Joskowitcz	Added commercial SD models
2006-07-13	4	Geri Joskowitcz	Added TripCom SD models
2006-08-20	5	Martin Murth	Added critical success factors
2006-09-09	6	Martin Murth	Minor corrections
2006-09-15	7	Martin Murth	Restructuring of critical success factores
2006-09-29	8	Martin Murth	Finalization

Project Consortium Information

Acronym	Partner	Contact
Leopold Franzens University Innsbruck http://www.deri.at	LFUI	Prof. Dr. Dieter Fensel Digital Enterprise Research Institute (DERI) Innsbruck, Austria E-mail: dieter.fensel@deri.org
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna university of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finnland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Sara Carro Martinez Telefonica I+D Madrid, España E-mail: tripcom@tid.es

Table of Contents

List of Abbreviations

API	Application Programming Interface
ASD	Agile Software Development
CVS	Concurrent Versioning System
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model
CSF	Critical Success Factor
DCM	Digital Contents Management
DoW	Description of Work, Tripcom Annex I.
EC	European Commission
EDI	Electronic Data Interchange
EU	European Union
UN/EDIFACT	United Nations EDI For Admin., Commerce and Transport
MEP	Message Exchange Pattern
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
PKI	Public Key Infrastructure
PSP	Personal Software Process
RDF	Resource Description Framework
RDFS	RDF Schema
RUP	Rational Unified Process
SVN	Subversion
SW-CMM	Software CMM
TripCom	Triple Space Communication
TSC	Triple Space Computing
TS API	Triple Space API
TSP	Team Software Process
UC	Use Case
UML	Unified Modelling Language
UP	Unified Process
W3C	World Wide Web Consortium
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language
WSML	Web Service Modelling Language
WSMT	Web Service Modelling Toolkit
WSMX	Web Service Execution Environment
XML	Extensible Mark-up Language

1 Introduction

This deliverable is the first deliverable from Work Package 6, Triple Space Architecture and Component Integration.

The objective of this deliverable is the discussion of collaboration procedures and the definition of critical success factors providing a basic understanding of the way the various contributions should be integrated into a single workable system.

Borrowing from an analogy with a factory, Work Package 6 represents the final assembly line. This is the place where all contributions come together, where contributions must fit each other and where the final result is proven in real world scenarios. We argue that architecture and integration require two basic sets of capabilities:

- The capability to create and maintain common concepts. This means that the creation of a common understanding of a single and evolving architecture among all contributors and the capability to resolve conflicts are crucial success factors.
- The capability for each contributor to re-use the concepts and components from all other contributors. This means that cross-fertilization and a unified methodological approach are crucial success factors.

Starting from the role the work package plays within the overall project, the paper describes an iterative process that integrates the results from the contributing work packages and feeds back a holistic view of the overall results to the contributors.

The paper is divided into 2 major parts:

The first part reviews the current state of the art of major software development technologies and discusses ways how these technologies are adapted to best fit the requirements of large, heterogeneous and highly distributed development environments like TripCom. Based on this discussion, it then elaborates on the collaborative processes that are developed in the early phases of the project and the necessary tools to be used in the later phases of the project.

The second part identifies several critical success factors for the project. These scientific, technical, economic, and organisational success factors will serve as a means to control and evaluate the progress of the project and describe functional and non-functional dependencies between the project's particular sub-tasks.

2 Software Development Models

A software development model serves as a means to control the software development process covering all development phases from requirement analysis and conceptual design to industrial operation and maintenance.

A common characteristic of all software development models is that they consider the process a collaborative task. This aspect is of importance in research projects consisting of project partners that are geographically distributed all over Europe. In [12], Adele Goldberg comes to the conclusion:

> In order to understand (online team) collaboration, we need to consider it in all of its senses: conceptual, practical, and educational.
>
> - *Conceptual collaboration* is sharing information and work, and sharing responsibility and leadership.
> - *Practical collaboration* is the decomposition of the work, the integration of work results, and the management coordination given differences in expertise with the practice.
> - *Educational collaboration* is helping one another learn on the job and learn on the demand, with activities that support reflecting on the lessons learned from practical experiences.
>
> Goldberg, [12]

All three kinds of collaboration are vital for TripCom. The responsibilities of Work Package 6 in particular clearly lie in managing the practical collaboration, i.e. defining a development and integration framework which allows for controlling and validating the development progress.

In the following sections we briefly present the state-of-the-art of most common software development models and analyse these models with respect to their applicability to large, heterogeneous, and highly distributed development groups.

2.1 Classic Software Development Models

2.1.1 Waterfall Model

The Waterfall model [27] prescribes a software development process which is strictly divided into five different phases. Each phase needs to be completed before the next phase can be entered. In Royce's original proposal [27] each phase has clearly defined starting points and deliverables. Also, he intended the model to be employed iteratively. However, the waterfall model is usually referred to as a strictly sequential software development process covering the following phases:

- Requirement analysis and specification
- System design and specification
- Coding and module testing
- Integration and system testing

- Delivery, deployment and maintenance

One of the biggest criticisms of the Waterfall model is that in most software projects the proposed phases cannot be exactly separated. Parts of a system could still be in the design phase while others are already implemented.

Practical experience also showed that the single development phases cannot always be followed strictly but stepping back to the previous phase is often required. However, stepping back in the Waterfall models implies that (most of the) features developed since this phase cannot be reused.

In general, the Waterfall model is not suitable for projects with potentially changing requirements, which is the default in current software development and especially true in scientific projects of highly distributed research teams.

2.1.2 Spiral Model

The Spiral model [4] is a meta-model for existing software development models that better supports changeability in that it defines the used process to be iterative. Each iteration consists of the following tasks:

- Definition of goals, alternatives and constraints
- Evaluation of the alternatives and identification of risks
- Implementation and validation
- Planning of the next iteration

Most often the Spiral model is used together with the Waterfall model. Due to the shorter iterations and the inherent risk management, the spiral model reduces overall risk factors of software projects.

However, the Spiral model can also be applied to other, more online team centric development models which makes it an potentially interesting approach for the TripCom development process.

2.1.3 Unified Process

The Unified Process (UP) [29] is a very common, iterative and incremental, precisely defined software development process framework. Its most well known refinement is the Rational Unified Process (RUP) [17] of IBM.

The UP should be seen as an extensible development framework, which is then customised for specific types of projects. This framework divides software projects into four main phases:

- **Inception phase**: The inception phase should be the shortest phase in the project and covers establishment of a justification or business case for the project, project scope and boundary conditions, outline of use cases and key requirements that will drive the design tradeoffs, outline of one or more candidate architectures, risk identification, and preliminary project schedule and cost estimate.

- **Elaboration phase**: The main goal of the elaboration phase is to validate the system architecture. Risks have to be analysed and the core and the most critical parts of the system are constructed iteratively. The system already has to exhibit functional and non-functional behavior according to the defined requirements, e.g. performance, scalability, and cost. The final deliverable of this phase is a plan for the following phase, the construction phase.
- **Construction phase**: In the construction phase, the largest phase in the project cycle, the remainder of the system is implemented. The features and components are developed in a series of short iterations, each of which results in an executable release of the system.
- **Transition phase**: In this last phase the system is deployed. The subsequent feedback can result in further refinements of the system which are implemented during additional transition phase iterations.

Key to the UP (and especially RUP) are a series of software development principles, best practices, and disciplines including iterative and incremental software development, use case driven development iterations, architecture-centric development, and risk-focused software engineering.

Although the UP is often used for bigger development teams, it does not put a special focus on online team collaboration. While many of its underlying principles can certainly be effectively applied to collaborative development models, other principles of the UP like the strict planning of tasks seem to be difficult to realise within highly distributed development teams.

2.1.4 Agile Software Development

The term *Agile Software Development* denotes a whole group of software development models rather than specifying a single model. These models are also called lightweight methodologies[1]. What they all have in common is that they employ an iterative approach to software development in order to minimise risk. As opposed to the spiral model, their approach usually is more radical, making each iteration a single little software project of its own. This way, a new version of the system can be released at the end of each development cycle.

Another main principle in agile software development is frequent and direct, preferably face-to-face, communication between all people involved in the project, i.e. from the programmer's, the contractor's, and the customer's side.

Also common to all agile models is that the primary measure of progress is working software. Compared to other models, they produce rather little written documentation which is one of the main criticisms against it.

Important representatives of agile software development models are Extreme Programming [3], Scrum [28], Crystal Clear [10], and Feature Driven Development [9, 23].

2.2 Collaborative Software Development Models

The introduction to Section 2 refers to the three kinds of collaboration described in [12]. Adele Goldberg further describes:

[1] Based on the Manifesto for Agile Software Development, http://agilemanifesto.org/

> The challenge to an online project community then is to combine, not separate, the three kinds of collaboration. A successful on line project community is a place in which the practical gives immediate access to the conceptual. And it is a place where on-the-job mentoring, discussion about decisions and trade-offs, and knowledge reuse, all together, from learning opportunities. They become educational collaborations among the project's team members, and may encourage individual reflection in the form of skills assessments, seminars, and courses.
>
> Goldberg, [12]

In practice, two groups of software development models turned out to provide this combination of collaborations: development models for industrial collaboration and development models for open source community projects.

2.2.1 Commercial Models

The ultimate goal of a commercial organisation engaged in the production of software is to create the right software solution or software product in the most efficient way.

Anecdotal evidence asserts that 50% of large software projects have overruns of more than 100% in time and budget and that 30% of all projects never reach the production phase [13]. Software development models are therefore highly important in commercial settings and a large body of research material has been produced in this area:

Software project management methods like the Capability Maturity Model (CMM) [15], software production valuation like COCOMO 2 (Constructive Cost Model) [5], software strategy evaluations like Balanced Scorecards [19], and software quality drives like Six Sigma [2] are popular with corporations.

Industrial processes always depend on separation and specialization and are by their very nature collaborative.

For TripCom, we limit this section to two practical examples. We will look into two aspects of commercial collaborative processes which might provide additional ideas for our project:

- **The Software Product Line Process (PLP)** [30]: the way a complex software project is split and the parameters used to control the collaboration of the individual software production parts
- **The Team Software Process (TSP)** [21]: the general rules and criteria used to manage the collaboration of software development teams as used in a typical commercial setting

Basically, a collaborative software development process can be split along two main divisions:

- Horizontal division along layers such as application layer versus system/framework layer. For example, a loan application could be structured on top of an XML publishing framework. Two teams — the application team and the framework team — collaborate to create one product. One team builds on top of the work of the other team, with the application team providing the requirements for the framework team.

- Vertical division of a software solution along complementary software products. For example, a booking system is integrated with a payment system to provide a complete accounts payable/receivable solution. Two teams — the booking system and the payment system team — collaborate to create the complete solution. Both teams integrate with the work of the other team.

Both types of divisions can be handled by the software product line process: A software product line is a set of software systems sharing a common, managed set of features that are developed from a common set of core assets in a prescribed way [8]. The core asset developers create and maintain the product line core assets, such as the business case, requirements, software architecture, test cases.

The product developers use the core assets to develop the specific products in the *product line*. The product line and the collaboration per se is controlled by a *production plan*. The production plan tells product developers how those core assets are applied to build a specific product in terms of:

- inputs needed to build a product
- activities that result in a completed product
- roles and responsibilities of the product developers
- interactions needed with other groups in the organization
- schedule and resources

Since multiple product variants may emerge from the same underlying core product, the PLP is strongly related to reuse concepts.

Table 2.1 (from Zubrow [34]) shows the criteria used to control the process. The process presumes a clear mission, a sufficiently good definition of the customers and of the requirements, as well as a pre-established business plan. Although this kind of data is usually not available in research projects like TripCom, the criteria listed in the table below are useful to manage the relationship of inter-dependent teams, e.g. the management of teams where one team uses the results of other teams.

While the PLP defines roles and responsibilities within collaborations, the TSP adapts the well known Capability Maturity Model for Software (SW-CMM) [25] to a collaborative development process and provides us with guidance on the interaction of the team members.

The TSP is layered on top of the Personal Software Process (PSP) [16]. Its main idea is to make each participant/engineer responsible, then to build self directed responsible teams and to provide an environment and a set of tools to allow for successful collaboration of the self-directed teams.

The TSP has therefore 2 main components: a team building component and a team management component. The team building component relies on participants that have been trained in the Personal Software Process and that implement its basic principles:

Each engineer is able

1. to follow a process discipline and to provide quantitative data on activities,
2. to plan personal goals and to estimate the necessary efforts, and

Objective	Product Line Manager	Asset Development Manager	Product Development Manager
Performance	Total product development cost, Productivity, Schedule deviation, Time to market, Effort distribution across lifecycle, activities Number of products (past, present, and future), Trends in defect density	Cost to produce core assets, Cost to produce infrastructure, Schedule deviation, Defect density in core assets, Number and type of artifacts, in asset library, Core asset quality	Direct product cost, Defect density in application, artifacts, Percent reuse
Compliance	Mission focus, Architectural conformance, Process compliance	Mission focus, Process compliance	Process compliance
Effectiveness	Return on Investment, Market satisfaction, Market feature coverage	Core assets utility, Core assets cost of use, Percent reuse	Customer satisfaction

Table 2.1: Objectives and Criteria used in the Software Product Line Process [34]

3. to engage in a quality cycle.

The initial TSP step calls for the creation of teams through the "Team Launch Event":

- Establish/Clarify overall product and business goals
- Assign team goals and roles
- Define development strategy
- Establish a top down plan
- Develop a quality plan
- Build a bottom up plan and consolidate
- Create a risk management plan
- Prepare the management environments and review schedule

The team management component defines the necessary actions to control and guide the team:

- Conflict resolution
- Continuous improvement and scoping of goals
- Continuous improvement of the precision of planning and estimation data
- Continuous improvement of the consolidated plan (rolling planning).

Like all methodologies, PLP and TSP require training, strong management sponsorship and commitment. They are usually introduced in steps over long time periods.

2.2.2 Open Source Software Development

Open source software development is a community-driven process which aims at software development of extremely distributed development teams. Interaction among the community members is mostly achieved via asynchronous textual communication like mailing lists, newsgroups, browsable archives, and Wikis.

The community driving the project can usually be divided into several subgroups (see also [1]):

- **Core developers**: This group is formed by a few (sometimes only one) core developers. Apart from contributing code, they also guide the project, assess the contributions and decide whether and which contributions will become of the next release.
- **Contributors**: The group of contributors actively adds to the development progress of the software. Contributors usually download and use the latest (unstable) builds, test them and report bugs, provide patches, and also implement new features.
- **Users**: The user community is the largest group and is mainly interested in using the software. The biggest value of this group is to provide feedback about usability, bugs, and to propose improvements and feature requests.

Besides the great success of open source software development, the model itself is also criticised in a number of respects.

First of all, the lack of face-to-face communication hampers tight collaboration and reduces the overall knowledge exchange. Social interaction between developers, which is nowadays proven to be an important success factor in software development [28], is hardly established in open source development environments. Another downside is, that quality of code and documentation is much more difficult to assure in open source projects and neglected in many cases. Thus, it is difficult for new developers to get into a project.

The reduced possibilities of communication and the aggravated project management are the biggest problems of open source software development. However, there are several proposals and approaches to surmount these problems. E.g., the introduction of more industrial-like core and management teams and the use of Semantic Web technologies to improve integration and aggregation of distributed information [1].

3 Collaboration Process in TripCom

3.1 Initial Layout of the Work Packages

According to Annex I. of the *Description of Work* (DoW) [33], the project has been divided into ten work packages (plus the project management work package).

Six of theses work packages are directly involved in the delivery of tangible software, e.g. software deliverables and software integrations (Figure 3.1):

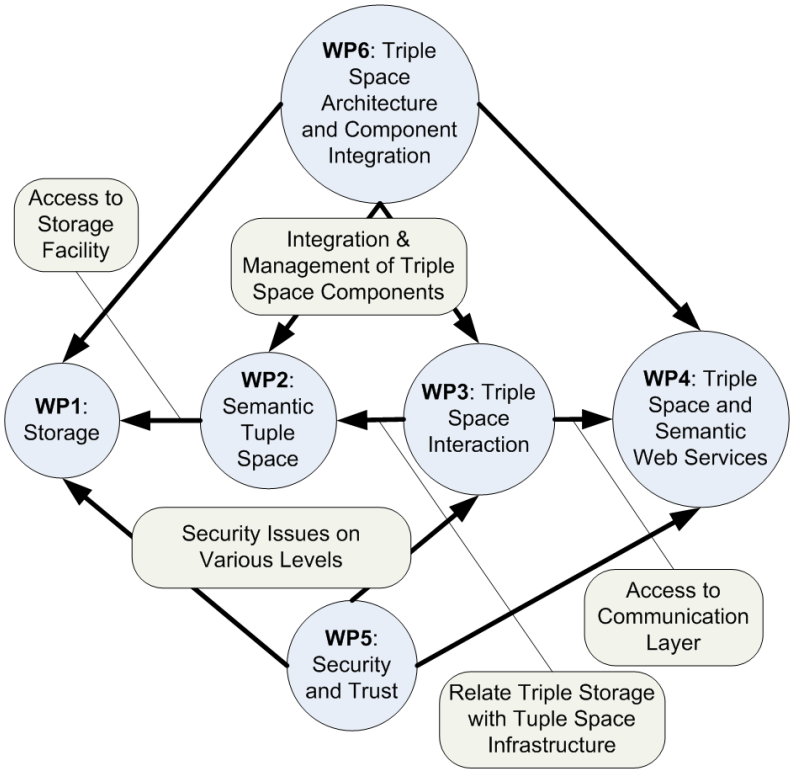

Figure 3.1: TripCom Work Packages [33]

The initial discussion centred on the understanding of the work package dependencies and the way to handle these dependencies:

The high-level responsibilities and dependencies of work packages 1 to 4 (which form the design and prototypical implementation of the Triple Space kernel and its interfaces to the physical storage layer and the message transport layer) and WP 6 (which defines the reference architecture and carries out the actual implementation of Triple Space) were initially understood as illustrated in Figure 3.1. Security and trust (WP5) are seen as orthogonal to this component specification and implementation. This assumption might not hold in future phases of the project, as security and trust issues leverage themselves on the mechanisms of the Semantic Web.

The layout of the work packages reflect an initial high-level architecture as represented in Figure 3.2. The high level TripCom architecture is strongly related to the work package definitions. The scoping, the responsibilities, and the component interfaces are clearly influenced by the work package definitions.

Figure 3.3 illustrates an attempt to map the work package definitions onto layers of the architecture and to provide high level definitions for the interface layers. The

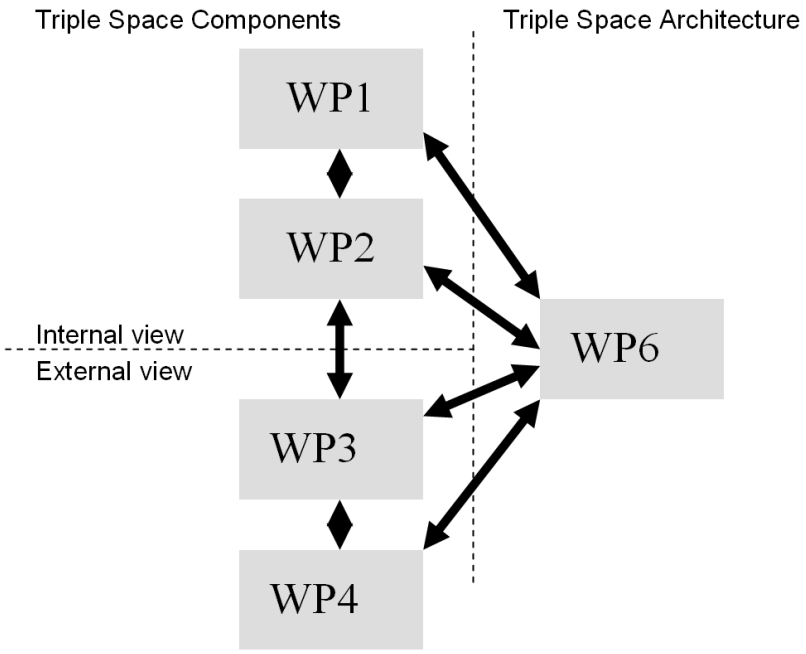

Figure 3.2: High-level architecture dependencies between TripCom Work Packages [32]

integration process, in its role as aggregator of the software deliverables, requires the definition of interfaces.

The next section describes the collaborative requirements and design process that underlies the work in WP6 — the integration work package. In the initial discussions, this work package was understood as the "meeting place", where discussion should lead to fitting and consistent concepts.

3.2 Collaborative Requirements Process

In order to provide a process that would lead to a consistent design, Work Package 6 members proposed an iterative requirement and design process. Such iterative requirement and design processes have been recognized as key success factors in large projects.

For example, an analysis of 250 large software projects, makes the following recommendation:

> Successful change control for applications in the 10,000-function point size range include the following:
>
> - A joint client/development change control board or designated domain experts.
> - Using joint application design (JAD) to minimize downstream changes.
> - Using formal prototypes to minimize downstream changes.
> - Planned usage of iterative development to accommodate changes.
> - ...
>
> Jones, [18]

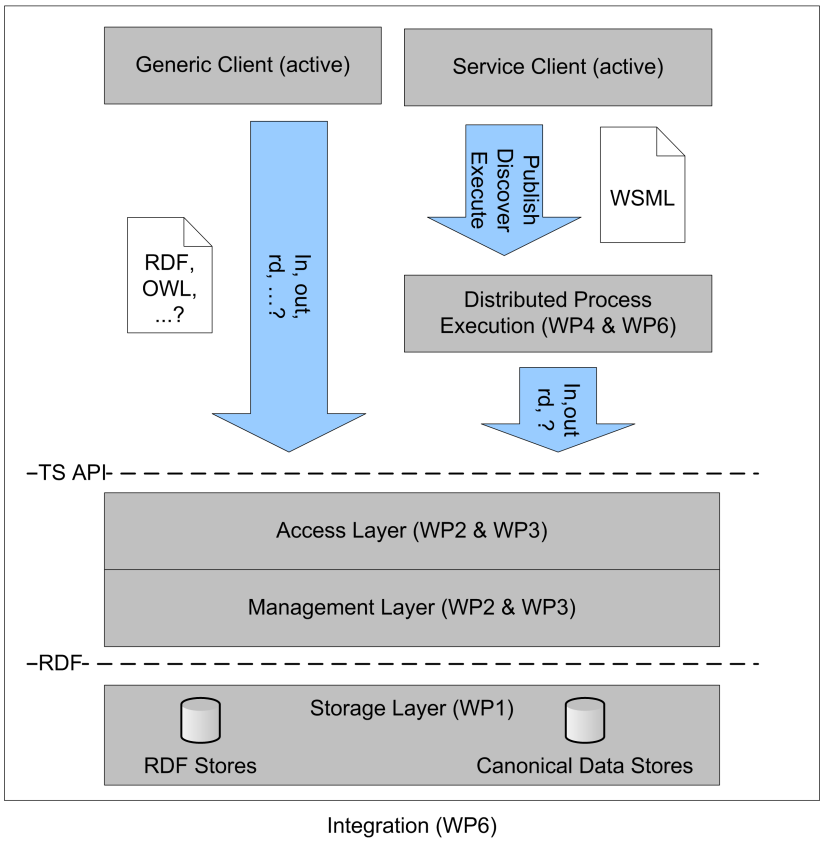

Figure 3.3: TripCom Layered Architecture

The sketch in Figure 3.4 summarises the ideas for cooperation between the contributors (WP1 ... WP5) in Work Package 6 (For a detailed description see Table 3.1).

The aim of this process is to optimise the synchronization between the participants while maintaining the maximum freedom in terms of terms of experimentation for the individual participants.

In analogy to the "Product Line Process" presented in the collaborative techniques section, the iterative loops are divided into 3 loops:

- **A high level iterative loop**: provides the general orientation
- **An applicative loop**: maps the specific requirements as provided by the use case Work Packages WP 8A and 8B
- **A framework loop**: generalises the discovered mechanisms and re-factors them into framework components

These iterative processes lead to baseline design specifications for applicative and framework components, e.g. to a consistent set of specifications as a starting point of the change control process.

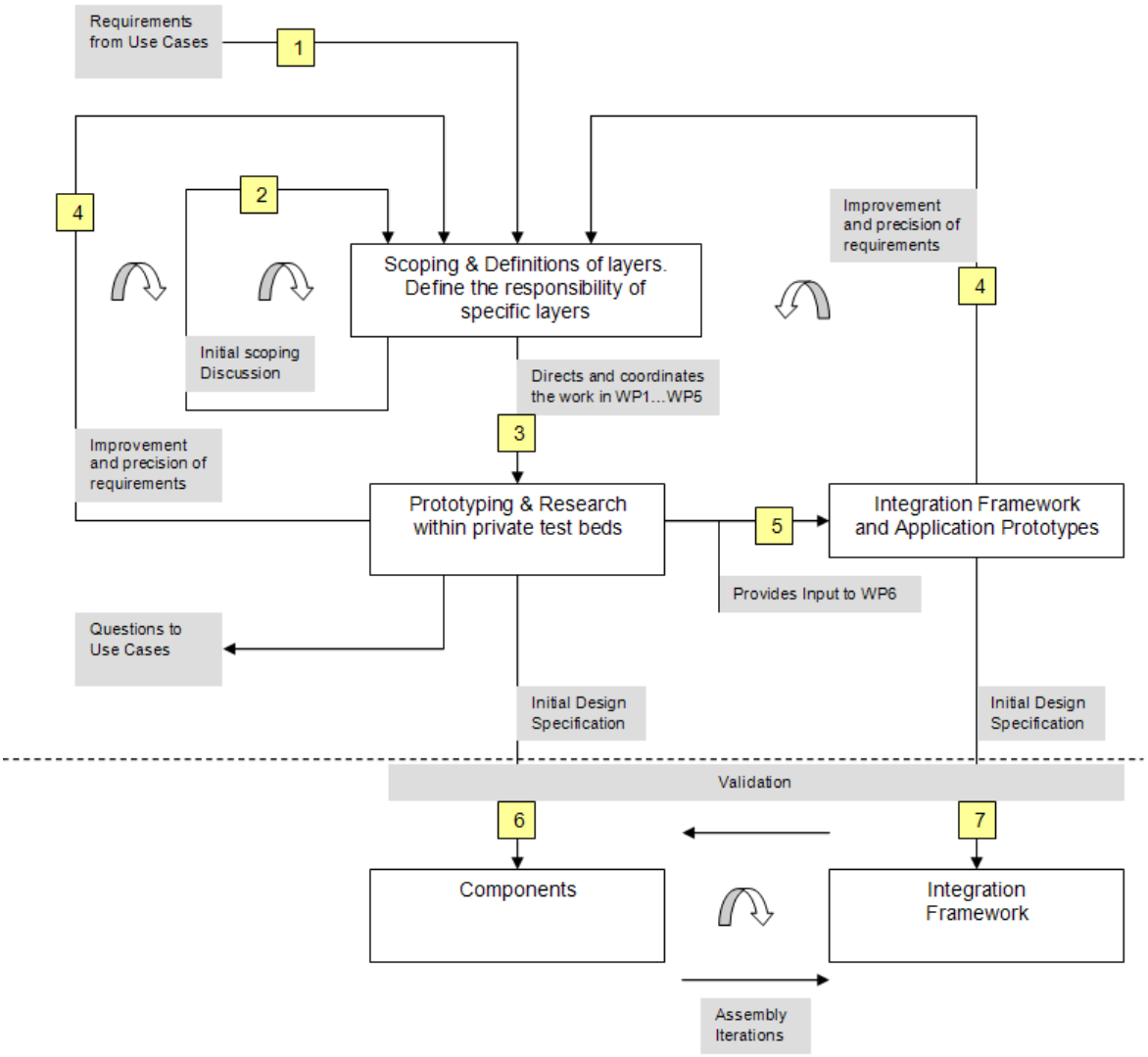

Figure 3.4: Cooperative Flows and Process

3.3 Iterative Software Development Process

An iterative design process as proposed in Section 2, depends on a common understanding on the level of precision for each iteration. Iterative development uses incremental steps to broaden the breadth and the complexity of the system.

A practical introduction to iterative development processes can be found in [20].

The iterative elaboration of the architecture is a central element of the iterative development process, especially in the context of the integration work-package and in multi-site integrations [22].

In order to define the architecture incrementally, we defined a three-level approach. Considering an initial collection of work package functions and of requirements, we propose an initial high level concept of the TripCom architecture consisting of layers that will be sketched around 3 dimensions:

- **The "Generic Use Cases"**: Whatever the real world use cases are, a semantic services architecture must provide the capability to publish, to provide goal-driven discovery mechanisms and to execute the discovered processes/services.

Flow	Target process	Meaning
Use Cases and Requirement (1)	Scoping and Definition of Layers and Components	The uses cases are checked "on paper" against the layers and components as defined in the overall architecture. The demarcation of the layers is clarified
Initial discussion (2)	Scoping Definition of Layers and Components	The initial discussion is the most inner loop of the iterative process. It will end, when the participants have reached a reasonable agreement on the basic principles of the architecture. Each participant should then be able to focus on its research and prototyping work.
Focused research questions (3)	Prototyping and Research in private test beds	The set of research questions formulated initially in the work package assignments are evaluated and prototyped in WP specific environments (maximum freedom). Results are fed back into the definition of the architecture.
Research results and Prototyping results (4)	Scoping and Definition of Layers and Components	Improvement of definitions based on evaluation and studies.
Framework requirements (5)	Framework prototyping	The requirements for the framework and the factored functions are driven by results from prototypes and evaluations. The results are fed back into the definition of the architecture.
Design requirements (6,7)	Design specification and implementation for Components and Framework	Once the prototyping and research stage has been validated against requirements and use cases, we enter the design stage.

Table 3.1: Cooperative Flows and Process

We call these basic capabilities "generic use cases". These capabilities are understood from the point of view of a client (a user, another system, etc.). We base the generic use cases on generalisation of use cases as proposed by WPs 8A and 8B.

The architecture must implement the principles of the behavior of the individual layers when executing these generic use cases: their responsibilities and their collaboration.

In this use case dimension, we discuss the outside interface in the form of the interaction between a client — which might have different incarnations — and the overall system.

The methodology shall be based on UML use case analysis and BPMN (Business Process Modeling Notation)[1].

- **The logical architecture**: The logical architecture describes the properties and the responsibilities of the logical components and their interaction.

[1] http://www.bpmn.org/

The logical architecture treats the system as a single structure with the aim of distributing it at a later stage. The logical architecture will not discuss transport systems or other forms of distributed infrastructures.

The methodology shall be borrowed from UML class and interaction diagrams.

- **The deployment architecture**: TripCom proposes a Web aligned, highly distributed system based on Triple Spaces. This means that the individual logical layers (components) are repackaged into cooperating, deployable physical entities over a distributed infrastructure. These entities are described as *nodes* in distributed systems.

  Such a physical entity (node) is addressable on the Web and provides specific functions:

  - Some nodes may provide goal driven discovery—in cooperation with other nodes.
  - Nodes are physical embodiments of a distributed space, e.g. a set of nodes implement virtual spaces.
  - Nodes may provide execution support for complex business processes—or cooperate with other nodes.
  - Nodes are deployed on physical systems.
  - A single physical system may support multiple nodes.

All views of the architecture can be described, defined and documented with UML. The UML diagrams provide custom views of the same architecture.

3.4 Integration Process and Test Process

Although the first phases of the project are dominated by the requirement and design processes, the multi-site integration and testing process must be prepared in time.

Multi-site integration is feasible if:

- Common information bases and communication between participants are maintained. [31] shows the importance of communication in a multi-site experimental set-up.
- The integration platform and the components to be integrated can be easily transferred between participating sites. This can be handled by version control systems, which are designed for collaborative coding are already used in the project for collaborative document editing.
- The used configuration management and version control system connects to all sites and is used as the central backbone.
- The bug tracking and resolution process is defined between the participants.

In practical, technical terms, a multi-site integration process should employ:

- A highly interactive communication system for the exchange of developer specific information. *Developer Wikis* have been shown to be successful in this area.

- An implemented Project Object Model (POM) [6] that contains information about the project and configuration details
- A central repository for software artifacts coupled to the POM. The repository is the ultimate reference for the complete system.
- A bug-tracking system

From a process point of view, the minimum requirements of a multi-site project are:

- A group of integrators – in analogy to the committers in open-source projects (Section 2.2.2).
- A clear understanding of the software artifact hand-over procedures
- A set of integration procedures
- A set of quality testing procedures (scope, testers, etc.)

4 Critical Success Factors for TripCom

The term "critical success factor" (CSF) was first used by Rockart [26] who proposed to define CSFs to identify the management's information needs. CSFs are those performance factors which need to be constantly monitored by the management in order to guarantee competitiveness. They are used to determine where management attention should be directed and to define development measures.

This chapter summarizes and categorizes the critical success factors identified within the TripCom work packages.

4.1 Technical and Scientific Success Factors

The following technical, scientific success factors have been identified:

4.1.1 Storage Model

The development of scalable and linkable Triple Space storages, based on improving and combining current RDF Stores and Tuple Space infrastructures represents a critical success factor in TripCom.

- **Flexible Storage Abstraction Model**: The most demanding research area in the field of Triple Space storage is the design of a flexible abstraction model that supports multiple storage implementations suitable for arbitrary persistence and reasoning strategies and scenarios. Furthermore, this model must allow for the integration of legacy data systems, where replication import is not feasible, and provide support for virtualized views of clusters of distributed sets of files.

  The storage implementation must be efficient, scalable and allow for high performance processing of data and metadata.

- **Semantic Stores**: Persistent semantic stores will need to be able to hold large quantities of semantic data and make it available in reasonable time to the Triple Spaces kernel. In connection with this, reasoners are necessary to enable semantic queries over the stored data so that retrieval of tuples from the space is possible.

4.1.2 Semantic Technologies

The work in Linda and tuplespaces is almost 20 years old, however its integration with semantic technologies is new and innovative. Aspects of Semantic Web and Semantic Web services research are still immature or incomplete. It is critical for the success of the Triple Spaces kernel that appropriate research results, methodologies, tools, and software might be available to make the implementation of the kernel feasible.

- **Semantic Web**: The Semantic Web must become a widespread reality in which truly large numbers of clients communicate through the exchange of semantically encoded data (e.g. RDF [14] or OWL [24]) if the realisation of a Triple Space kernel is to be justified.

- **Semantic Web Services**: It is important to provide a *platform for invocation of Semantic Web services*. This includes multiple sub-factors:
  - providing a registry for publishing of Web service descriptions and corresponding discovery methods to facilitate discovery of Semantic Web services,
  - defining mechanisms for communicating with TripCom-based Semantic Web services,
  - providing support for common Web service message exchange patterns, and
  - providing suitable transport mechanisms.
- **Integration with WSMX**: An important technical CSF regarding invocation of Semantic Web services is *integration of TripCom and WSMX* [11] to enable TripCom-based communication between service requester, WSMX and service providers, TripCom-based resource management in WSMX and inter-WSMX communication.

4.1.3 Web-Scale

Orthogonal to the factors of technology availability is technology capability. A general requirement of TripCom is operation at a global scale. Therefore, it is important to find a set of interoperable mechanisms, techniques, and tools that allow for the implementation of such a scalable Triple Space architecture.

- **Distribution and Parallelization**: For a Web-scale Triple Space architecture, it is essential that data is distributed over numerous data stores and that searches can be parallelized to improve the overall processing time. Usage scenarios like the European Patient Summary (EPS) [7] will need to query big amounts of data within seconds, e.g. in emergency services. For implementing such a scalable TripCom infrastructure, it will be important to seamlessly integrate the storage systems with the Triple Space kernel architecture.
- **Publish-and-Read**: The central idea of TripCom is to bring Semantic Web service technology to a global scale by adopting the publish-and-read principle from the Web. Therefore, it is important to enhance available tuplespace technologies in such a way that they allow for storage, retrieval, and interaction via semantic data while fully conforming to the publish-and-read principle. In this way, the applicable caching mechanisms pave the way for high scalability.

4.1.4 Security

Security itself is a critical success factor for TripCom. Triple Space Computing would remain an unacceptable means for the automation of real-world business process integration without ensuring appropriate security and trust mechanisms. Therefore, we need to guarantee protection to data in the space, e.g. assuring that only qualified subjects can access data, and that data is authentic and has not been tampered with. Data communication channels and storage need to be properly protected.

- **Open and Closed World Security**: A major challenge in this is to give proper security guarantees in an "open" world: Triple Space Computing can be profitably used in "closed" environments, but cannot be restricted to them. Just like the Web, it needs to be an open infrastructure. This means that we need to provide security measures that are effective also when there is no central authority or control point. Two issues appear particularly important in such a situation:
  - Generally, identities and authentication, i.e. security measures cannot rely on the establishment of something like a single global PIK (Public Key Infrastructure), since experience has shown that this is infeasible on Internet scale. However, for some applications, there will be a requirement for explicit registration before any access is allowed (For example, this is true for patient records, preferred customers, and employee sites).
  - Trust establishment, i.e. in an open world, where many actors are unknown, a mechanism to establish trust and decide with whom to interact (and to what extent) is needed. Reputation can be a feasible and effective means of doing this in some instances
- **End-to-End Security**: Triple Space Computing can have a significant impact on the use of Semantic Web services for cross-enterprise communication. In order to achieve this goal, mechanisms are needed to ensure security in complex communication patterns (e.g. complex composition, orchestration, etc.) involving mediators and actors that may be previously unknown.
- **Security Management**: In TripCom's eHealth use case, the capability to *manage security*, in terms of authentication of users and related authorisations for accessing and modifying citizen health data stored in the European Patient Summary, provides a solution for the privacy and ownership issues.

4.1.5 Use Cases Solutions

The developed Triple Space technology must allow for efficient implementation of TripCom's use cases and should exhibit advantages over today's state-of-the-art implementation approaches.

- **Enterprise Application Integration**: For the realization for the Digital Content Management (DCM) solution, the Triple Space should handle *heterogeneous information with different data formats and concepts from different content providers.* The search for fine grained digital content ontology will is a a CSF.
- **Sharing Health and Medical Data Among Healthcare Organisations**: In order to guarantee instant access to medical information of any citizen across Europe in a cost-effective way, the solution must provide an *efficient, distributed. and scalable infrastructure* by distributing the demand of physical resources over all the healthcare organisations.

4.1.6 Ontological Infrastructure for Business Processes and Data

- **EDI-based messaging standard**: In order to provide a proper solution for the EAI use case, an EDI-based messaging standard must be found in order to provide negotiation messages between all roles present in a Digital Content Management scenario. This standard must be accurate enough to provide specific information regarding this business scenario.
- **Heterogeneity**: Using *ontologies* and *mediators*, the infrastructure developed for the eHealth use case must be a multi-lateral solution where any party involved (electronic health records, hospital information systems, ...) can interoperate with the European Patient Summary dealing with data and service heterogeneity issues.

4.2 Organisational Success Factors

The most part of the identified organizational CSFs are related to certain use case requirements. The rest of the CSFs relates to management tasks, dissemination, and exploitation.

4.2.1 Enterprise Application Integration

The follwoing factors are critical for the success of the EAI use case:

- **Improvement of existing EAI Infrastructures**: Existing EAI architectures must be improved by the use of TripCom-based system architecures. Provision of concrete TripCom-based architecture proposals for data integration, message integration and business processes integration between companies, in which asynchronous communication, service orientation and semantic mediation between agents are the key features for these architectures is considered a CSF.
- **Easily Adoptable System Architectures**: The TripCom-based EAI architecture should be easily accessible and understandable so that enterprises are not forced to radically rework the systems that they want to integrate via a Triple Space.

4.2.2 Sharing Health and Medical Data Among Healthcare Organisations

In the eHealth use case, the follwing CSFs were defined:

- **Solving Herogeneity Issues of Eurpean eHealth Care Data**: The main organisational critical success factor of the eHealth use case is the provision of an infrastructure capable of enabling instant access to medical information of any European citizen any time and anywhere across Europe. As a matter of fact, the medical information of European citizens may be stored and spread across several healthcare informative systems in different countries and, when coded, regional or national standards are followed. Therefore WP8B infrastructure

must be capable of *overcome heterogeneities among different eHealth standards* (messages, protocols and coding systems).

- **Subsidiarity Principle**: Another relevant organization critical success factor is the *respect of the subsidiarity principle*. The developed infrastructure must not interfere with the internal workings of healthcare organization administrations, leaving each healthcare organization the largest degree of freedom in making its own choices independently from the others organisations.

- **Data Ownership**: An additional critical factor is the capability of the developed infrastructure to *guarantee the privacy of citizens and to provide a solution for the problem of data ownership*. Each healthcare organization must keep control over its clinical data, by sharing information only with other authorised healthcare organisations. In this way, all EU citizens can feel free to move around Europe, since only authorised users will be able to easily and securely access their health data.

4.2.3 Standardisation, Dissemination and Exploitation

- **Visibility of TripCom Research Activities**: An important organisational CSF is *ensuring high visibility of TripCom*. This is achieved through various communication channels, e.g. announcement of the project progress on the project Web site or publications and presentations at conferences.

- **Standards**: To enable integration of TripCom with existing technologies, TripCom needs to *be aligned with existing standards* and to *take influence on existing and emerging Web-, Web service-, and Semantic Web-related standards* where necessary.

- **Input to other Research Activities**: Perceptible impact on other research areas, e.g. Grid technologies and WSMX, is also considered a success factor of TripCom. In the case of WSMX, it is important to provide input on the *applicability of TripCom for Semantic Web service invocation (both in combination and without WSMX*) and thereby allowing validation of the TripCom architecture. A necessary requirement for this step are clearly defined use cases that define the scenario and requirements for service invocation.

4.2.4 Management

The main responsibility of the management task are measures concerned with ensuring the smooth execution of the project over the full course of its duration. There are several documents and procedures which are produced and implemented to this end and are considered critical success factors:

- **Following the detailed work plan**: It has to be made sure that all tasks carried out under the project are on time according to schedule. The deliverables should be completed at their respective deadlines to give enough time for quality assessment. This especially holds true for critical tasks which other tasks depend on.

- **Collaboration Model**: A *collaboration model* that fosters unhampered knowledge transfer and collaboration between the participating project partners has to be introduced. This includes exchange of pre-existing domain knowledge, documentation of new research results, immediate transfer of new insights, and a collaborative software development model for the implementation of the Triple Space prototype.

- **Quality Assurance Procedure**: All the results produced during the project should undergo a competent and extensive quality evaluation procedure. External quality assessors should also be invited to have a neutral evaluation of the project outputs.

- **Self-Assessment**: As the project progresses, an assessment of all project goals should be carried out in order to evaluate whether the project outputs are fulfilling the overall objectives. The overall project outputs should also be compared to related research communities and to their work and results.

- **Management Reports for the European Commission**: The management reports are intended to provide to the European Commission EC as well as to the consortium information about the exact scope of the work to be done, the goals to be reached, the completed/ongoing work and the efforts needed. They also work as control unit to raise a red flag on every possible upcoming risk. The financial consumption should justify the amount and quality of results produced in a particular period of the project.

- **Risk Management**:The risk management combines the identification of possible risks, the monitoring of these risks and the initiation of necessary counter actions in case of need. It is important that all consortium partners are aware of possible risks to the project and use the appropriate communication channels to inform the project management in case of any deviation.

- **Monitoring of the ESSI Cluster**: The monitoring of the ESSI[1] (European Semantic Systems initiative) membership as well as successful internal conflict management are CSF of the management of the project.

4.3 Economic Success Factors

The following economic CSFs have been identified.

4.3.1 Cost of Triple Space Technology Development

- **Additional Tools and Software Licences**: The extra costs caused by additional tools, software licences, etc. needed for the development of the Triple Space prototype must not exceed the allocated budget, e.g. tools for ontologizing EDIFACT subsets.

- **Controlling**: The project management must provide for a smooth project execution without creating management overhead for the consortium partners. If

[1] http://www.sdkcluster.org/

any changes/updates are required, there should be minimum possible changes adapted while consuming minimum possible resources.

4.3.2 Cost of Industrial Application

- **Technology Application**: TripCom-based applications must demonstrate economic benefits over existing ones.
- **Technology Adoption**: Adoption and further development of the TripCom infrastructure by the industry must be possible in a cost effective way.

5 Conclusion

In order to achieve the necessary expected results for the next implementation relevant project phases, a collaborative infrastructure will have to be set in place which allows each contributing party to work on a common shared and collaborative environment. Based on an analysis of state-of-the-art development models and collaboration processes, a concrete development model for research projects like TripCom has been proposed. This model is derived from a commercial software development process and common open source software development techniques and was tailored to best fit the characteristics of research projects like TripCom. As a consequence, a crucial task for the next project phase will be the refinement of the proposed model in terms of defining concrete sub-tasks and responsibilities.

The analysis of critical success factors showed that there are three main areas essential for a successful realisation of TripCom.

From a technical point of view, the success of TripCom highly depends on the uptake of other technologies like Semantic Web and Semantic Web Services. Furthermore, achievement of a scalable TripCom infrastructure, guaranteeing secure data exchange and seamless integration of established standards are the most important technical challenges of TripCom. For a scientific success of TripCom, it will also be important to disseminate the research results, to reach high visibility in the international research community and to contribute the found results to standardisation bodies.

The organisational success factors point out the need of clear management tasks like following a detailed work plan, quality assurance, self-assessment, reporting and risk management.

The main focus of economic success factors for TripCom is on the cost of implementation of a Triple Space based solution, which will be a deciding criteria for industrial adoption of the developed technology.

Finally, only the implementation of a well performing Triple Space prototype can serve as a proof of concept and motivate further developments in this area of research.

6 Acknowledgements

The section on critical success factors is based on responses to queries submitted to all contributing work packages. The authors wish to thank all contributors for the valuable input on this section.

References

[1] Anupriya Ankolekar, James D. Herbsleb, and Katia Sycara. Addressing challenges to open source collaboration with the semantic web. In *Proceedings of Taking Stock of the Bazaar: The 3rd Workshop on Open Source Software Engineering, the 25th International Conference on Software Engineering (ICSE)*, pages 9–13, Portland OR, USA, 2003.

[2] Matt Barney and Tom McCarty. *The New Six Sigma: A Leader's Guide to Achieving Rapid Business Improvement and Sustainable Results*. Pearson Education, 2002.

[3] Kent Beck. *Extreme Programming Explained–Embrace Change*. Addison-Wesley Professional, 1999.

[4] Barry Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, 1986.

[5] Barry W. Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, Winsor A. Brown, Sunita Chulani, and Chris Abts. *Software cost estimation with Cocomo II*. Prentice Hall PTR, 2000.

[6] John Casey, Vincent Massol, Brett Porter, Carlos Sanchez, and Jason van Zyl. *Better Builds with Maven*. Mergere Library Press, 2006.

[7] Dario Cerizza, Emanuele Della Valle, doug foxvog, Reto Krummenacher, and Martin Murth. Towards european patient summaries based on triple space computing. In *Proceeding of 1st European Conference on eHealth (ECEH)*, 2006.

[8] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, 2002.

[9] Peter Coad, Eric Lefebvre, and Jeff De Luca. *Java Modeling in Color With UML: Enterprise Components and Process*. Prentice Hall International, 1999.

[10] Alistair Cockburn. *Crystal Clear : A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, 2004.

[11] Christoph Bussler et al. Web service execution environment (WSMX). *W3C Member Submission*, June 2005. Available at: `http://www.w3.org/Submission/WSMX`.

[12] Adele Goldberg. Collaborative software engineering. *Journal of Object Technologie*, 1(1):1–19, 2002.

[13] The Standish Group. The chaos chronicles vers. 3.0. Available via: `http://www.standishgroup.com`, 2004.

[14] Patrick Hayes and Brian McBride. RDF Semantics. W3c recommendation, W3C, 2004. Available at `http://www.w3.org/TR/rdf-mt/`.

[15] Watts S. Humphrey. *Managing the Software Process*. Addison-Wesley Professional, 1989.

[16] Watts S. Humphrey. Using a defined and measured personal software process. *IEEE Software*, pages 77–88, May 1996.

[17] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.

[18] Capers Jones. Software project management practices: Failure versus success. *CROSSTALK The Journal of Defense Software Engineering*, 2004.

[19] Robert S. Kaplan and David P. Norton. The balanced scorecard: measures that drive performance. *Harvard Business Review, Jan – Feb*, pages 71–80, 1992.

[20] Craig Larman. *Applying UML and Patterns – An introduction to Object-Oriented Analysis and Design*. Prentice Hall PTR, 1997.

[21] Donald R. McAndrews. The team software process TSP: An overview and preliminary results of using disciplined practices. Technical Report Technical Report CMU/SEI-2000-TR-015, ESC-TR-2000-015, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA 15213-3890, 2000.

[22] Päivi Ovaska, Matti Rossi, and Pentti Marttiin. Architecture as a coordination tool in multi-site software development. *Software Process: Improvement and Practice*, 8(4):233–247, 2003.

[23] Stephen R. Palmer and John M. Felsing. *A Practical Guide to Feature-Driven Development*. Prentice Hall, 2002.

[24] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. W3c recommendation, W3C, 2004. Available at `http://www.w3.org/TR/owl-absyn/`.

[25] Lisa Pracchia and Bill Hefley. Accelerating sw-cmm progress using the tsp. In *Assuring Stability in a Global Enterprise (SEPG)*, Pittsburgh, Boston, 2003. PA: Software Engineering Institute, Carnegie Mellon University.

[26] John F. Rockart. Chief executives define their own data needs. Technical report, Harvard Business Review, 1997.

[27] Winston. W. Royce. Managing the development of large software systems. In *Proceedings of IEEE WESCON*, pages 1–9, 1970.

[28] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.

[29] Kendall Scott. *The Unified Process Explained*. Addison-Wesley Professional, 2001.

[30] Carnegie Mellon Software Engineering Institute (SEI). The product line practice initiative. Available at: `http://www.sei.cmu.edu/productlines/plp_init.html`, 2006.

[31] Sadat S. Shami, Nathan Bos, Zach Wright, Susannah Hoch, Kam Y. Kuan, Judy Olson, and Gary Olson. An experimental simulation of multi-site software development. In *Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 255–266. IBM Press, 2004.

[32] Robert Tolksdorf, Lyndon Nixon, Elena Paslaru, and Francisco Martin-Recuerda. A conceptual architecture for the triple space kernel. Technical report, Tripcom, FP6 - 027324, FU Berlin, LFU Innsbruck, 2006.

[33] Tripcom. Triple space communication, Annex I. – Description of Work. FP6 - 027324, November 2005.

[34] Dave Zubrow and Gary Chastek. Measures for software product lines. Technical Report Technical Note CMU/SEI-2003-TN-031, Carnegie Mellon University, Software Engineering Insitute, Pittsburgh, PA 15213-3890, 2003.