GENiC
Globally Optimised Energy Efficient Data Centres

GENiC

Deliverable D5.1
Development & Integration guidelines including integration environment & means

Dissemination Level: Public

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 608826

Project Number	:	608826
Project Title	:	Globally optimized ENergy efficient data Centres - GENiC
Deliverable Dissemination Level	:	Public

Deliverable Number	:	D5.1
Title of Deliverable	:	Development & Integration guidelines including integration environment & means
Nature of Deliverable	:	Report
Internal Document Number	:	GENiC_D51_WP5
Contractual Delivery Date	:	M9
Actual Delivery Date	:	M10
Work Package	:	WP5
Author(s)	:	All
Total number of pages (including cover)	:	28

Abstract

This document describes development and Integration guidelines, including integration environment & means.

Keyword list

Development, Continuous Integration, Testing, Revision Control System, Build Automation Tools, SW Packaging, SW Delivery.

Document History

Date	Revision	Comment	Author/Editor	Affiliation
Mar 31, 2014	1	Initial draft version	Enric Pages	ATOS
Apr 01, 2014	2	Tentative ToC	Enric Pages	ATOS
Apr 15, 2014	3	Install and user guides templates	Enric Pages	ATOS
May 07, 2014	4	Development guidelines and toolset	Enric Pages	ATOS
May 21, 2014	5	Component Survey	Enric Pages	ATOS
May 29, 2014	6	Section 6	Enric Pages	ATOS
June 03, 2014	7	Section 5	Enric Pages	ATOS
June 12, 2014	8	Section 6 update, section 7, section 8 draft	Enric Pages	ATOS
June 23, 2014	8	Compilation of pending inputs	Enric Pages	ATOS
June 27, 2014	9	Quality Review	John Bynum	TUe

Executive Summary

The intention of this deliverable D.5.1 is to present the set of tools and establish recommendations and guidelines for the development and integration of the GENiC platform. It describes both development and integration methodologies to manage and control the implementation, testing, deployment and delivery of components.

The guidelines include best practices for Source Control Management, documentation, testing, build automation, continuous delivery and software packaging.

Additionally this document provides information about GENiC collaborative environment set up by the consortium and introduces the continuous integration practices that will be included in the development process.

Table of Contents

Index of figures

Index of tables

1 Introduction

1.1 Overview

The aim of this deliverable, D.5.1 Development & Integration guidelines including integration environment & means, is to present a set tools, common practices and guidelines for the development and integration environment provided by the consortium. It will establish the guidelines and procedures for the GENiC components which will be integrated in the scope of WP5 to build the GENiC integrated platform (initial and final versions).

A Collaborative Development Environment (CDE) has been set up to facilitate the software development process of isolated and distributed teams of different organizations.

A Continuous Integration (CI) system has been introduced in this document, including various best practices for the project management and integration tools and software development tools.

1.2 Structure of the document

This section provides information about how the document is structured; the deliverable "*D.5.1 Development & Integration guidelines including integration environment & means*" is the first of five deliverables within the WP5.

The document establishes best practices guidelines and procedures for the software and integrations tasks in the project. It establishes the procedures for the SW components within the architecture which will be integrated in "*D.5.3 GENiC integrated platform (initial version)*" [2] and "*D.5.4 GENiC integrated platform (final version)*" [3].

The document is organized in 8 sections presented below:

- Section 1 introduces the document, its structure and content as well as provides a glossary of acronyms.
- Section 2 describes briefly the development methodologies adopted in GENiC to facilitate the collaboration of distributed development teams.
- Section 3 introduces the Continuous Integration approach in order to support the software packaging process.
- Section 4 collects the set of tools and guidelines for the development environment, including team collaboration, software documentation, source control management, bug reporting and SW delivery.
- Section 5 collect the set of tools and guidelines for the integration environment, including software management builds for testing, compilation and deployment of the components and continuous integrations procedures to support the platform deployment and delivery.
- Section 6 provides an overview of the development and integration environment and some of its functionalities.
- Section 7 includes the references used for the document.
- Section 8 provides a set of document templates, including user guide, installation guide and a readme file.
- Section 9 provides the template of the component survey used early on to capture the configuration and time plan for the architecture components.

1.3 Glossary of Acronyms

Acronym	Definition
CDE	Collaborative Development Environment
CI	Continuous Integration
D	Deliverable
DRS	Document Review Sheet
EC	European Commission
GC	GENiC Component
GCG	GENiC Compoent Group
GENiC	Globally optimized Energy efficient data Centres
IDE	Integrated Development Environment
IDF	Invention Disclosure Form
MVN	Apache Maven
SCM	Source Code Management
SMARTER	Specific, Measurable, Achievable, Relevant, Time-bound, Evaluated, Reevaluated
SVN	Subversion
SW	Software
TDD	Test Driven Development
WP	Work Package
SDLC	Software Development life-cycle

Table 1 - Glossary of Acronyms

2 Development Methodology

Software development methodologies will be used to structure, plan and control the progress of each software component within the GENiC architecture. Each GENiC component group GCG or individual GENiC software component is allowed to use its own methodology, such as waterfall, prototyping, iterative, incremental, spiral or rapid development.

Each development team for a specific component will be in charge of the implementation, documentation and validation of the aforementioned piece of software as well as to integrate their functionalities within the architecture, the different development groups in the project are regulated by an Invention Disclosure Form (IDF) which will be used to assess ownership of intellectual property rights.

Common development activities and phases have been identified through the GCG to guide the development; these common activities in the scope of the platform will follow **agile methodologies** to support the developments during the project life-time.

Agile developments for software engineering promotes development, teamwork, collaboration and process adaptability and fits well for complex systems like the GENiC platform with nondeterministic, non-linear characteristics. In order to avoid ambitious plans or predictions, which are often hard to reach, our methodology will rely on an adaptive and iterative approach.

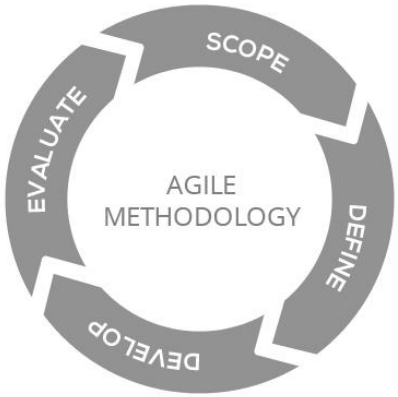

Figure 1 Agile Development life-cycle

Below are some principles that the agile philosophy can bring to the development process:

- Continuous delivery of valuable software;
- Adaptive; welcome changing requirement even late in the development;
- Iterative; deliver working software frequently;
- Team work collaboration as a key aspect;
- The environment should provide a friendly developer experience;
- Promote face-to-face communication;
- Use software features as a measure to check the progress;
- Sustainability of the development environment;
- Towards technical excellence and good design enhances agility;
- Simplicity;
- Self-organizing teams;
- Tune and adjust at regular intervals based on team continuous feedback.

Four main phases have been identified for the common development activities:

- **Design phase:** Requirements elicitation and architecture analysis.

- **Development phase:** Implementation, testing, documentation
- **Deployment phase:** Integration, deployment and platform maintenance
- **Delivery phase:** Packaging and software delivery

Design and architecture analysis is the initial step of any software development model, the elicitation of requirements cannot be fully covered at the beginning of the development cycle as suggested by traditional development approaches.

In research projects, requirements may need to be adapted (i.e.: after the refinement of our draft architecture). In that sense agile methodologies allow us to assess the direction of the development lifecycle through cadences, or iterations, of work where every aspect of development is continually integrated, tested and documented.

The **design** phase is led by WP1 Requirements, Architecture and Process Definition, where a set of requirements is identified trying to cover the whole set of desires for the GENiC platform. The requirements are intended to be as **SMART/ER** as possible, where SMARTER is a mnemonic acronym meaning **S**pecific, **M**easurable, **A**chievable, **R**elevant, **T**ime-bound, **E**valuated, and **R**eevaluated.

Implementation, testing and documentation activities within the development phase will be carried out by the development work packages, covered from WP2 to WP4. Each component or group of components within the architecture has a direct mapping with a project task, trying to archive the milestones defined per task. These components have been described in the first iteration of the GENiC architecture (D1.2. Requirements & Draft Architecture [1]).

The development work packages will lead the software design tasks that will be accomplished with the **implementation** of the components; at this stage this process could include a refinement of the architecture or its requirements.

Software **testing** and **documentation** are very important steps in the life-cycle as they help teams to recognize defects or inconsistencies in early stages of the development allowing for mitigation plans or refinements in future iterations.

In order to facilitate the **integration**, deployment and delivery of the final prototype, WP5 Integration and Validation will be handling the validation and delivery of the development work performed in the previous phases.

3 Integration Methodology

Continuous integration (CI) is a software engineering practice in which new features and capabilities that were developed separate from the main large code base will be added to the stable code after an automated process of testing and automation of component builds. This means that each developer team keeps their work-in-progress continually integrated with every other developer team.

An article by Martin Fowler provides perhaps the most popular description of the continuous integration practice.

"*Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly."*

This methodology involves producing a clean build of the system iteratively using CI tools (further described in section 5); traditional software development methods don't dictate the frequency of integrations but in agile developments it is fundamental to integrate as often as possible rather than to integrate rarely. (See an example of CI cadence in Figure 2).

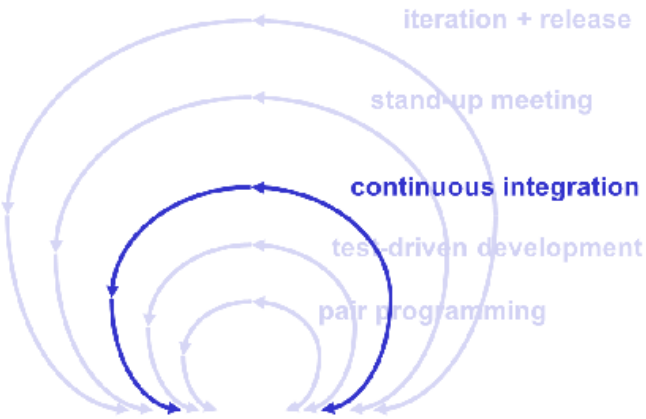

Figure 2 Continuous integration for agile developments

Here some principles and practices to support continuous integration and continuous deployment:

- Maintain a single source repository,
- Automate the build,
- Make the build self-testing,
- Every commit should reflect a new feature to integrate,
- Keep the build fast,
- Test in cloned environments,
- Make easy to get the latest stable executable,
- Give visibility to the development teams, and
- Automate deployment proving a continuous deployment releasing every successful build.

4 Development Guidelines and Toolset

4.1 Development Toolset

FusionForge is a free software fork based on GForge. It is a web-based collaboration software originally created by SourceForge [6], and the fork software is licensed under the GNU Public License. GForge provides project hosting, version control, bug-tracking and messaging though collaboration mailing lists.

```
http://it-forge.atosresearch.eu/projects/genic/
```

Source Control Management (SCM) refers to the use of a system to help developers keep track of the version history of their source codes, parallel versions and releases. **Apache Subversion [7],** called SVN, is the software versioning and revision control system used in our GENiC collaborative development environment.

```
http://it-forge.atosresearch.eu/svn/genic/trunk
```

Bugzilla is a bug-tracking system and testing tool licensed under the Mozilla Public License.. This tool will provide in the scope of GENiC a clear as well as centralized overview of development requests and their states. It also keeps track of software problems and resolutions. This allows developers, through the diligent use of the bug system, to detect and correct issues during the development cycles as well envisage possible inconsistencies, giving teams the ability to detect problems early and mitigate them in early stages.

```
http://itforgebugzilla.atosresearch.eu/bugzilla/
```

Mailman [12] is a free software for managing electronic mail discussions. It is integrated with a web dashboard to easily manage accounts and discussion lists. Mailman is written in the Python programming language and distributed under the GNU General Public License. The classifications of discussions in GENiC were divided by development Work Packages, management discussions, general technical discussions and test-bed discussions. See below the current mailing lists, private and public, used by the consortium within GENiC.

- General mailing lists:

```
genic-general@lists.it-forge.atosresearch.eu
-- Consortium mailing list (all GENiC partners)
genic-tech@lists.it-forge.atosresearch.eu
-- Technical mailing list (technical partners)
genic-testbeds@lists.it-forge.atosresearch.eu
-- Testbeds mailing list
genic-executive@lists.it-forge.atosresearch.eu
-- Executive Board mailing list
genic-advisory@lists.it-forge.atosresearch.eu
-- Advisory Board mailing list
```

- WP mailing lists:

```
genic-wp01@lists.it-forge.atosresearch.eu -- WP1 mailing list
genic-wp02@lists.it-forge.atosresearch.eu -- WP2 mailing list
genic-wp03@lists.it-forge.atosresearch.eu -- WP3 mailing list
genic-wp04@lists.it-forge.atosresearch.eu -- WP4 mailing list
genic-wp05@lists.it-forge.atosresearch.eu -- WP5 mailing list
genic-wp06@lists.it-forge.atosresearch.eu -- WP6 mailing list
genic-wp07@lists.it-forge.atosresearch.eu -- WP7 mailing list
genic-wp08@lists.it-forge.atosresearch.eu -- WP8 mailing list
```

- Public mailing lists:

```
genic-public@lists.it-forge.atosresearch.eu
-- Users interest group
```

4.2 Source Control Guidelines

The source code management guidelines will describe the set of best practices used by software systems to support developers with keeping track of version history of source code as well releases, parallel versions, etc. The version control system selected offers a logical way to organize and control revisions. **SVN, Apache Subversion** is an open source version control system (VCS), which allows users to manage files and directories as well as the changes made to them over time.

Repository Layout

Due to the nature of the GENiC research project where developers are not working in the same location, but dispersed across many countries, the management of technical contributions and software components is an essential aspect. As such, the organization of code decoupled from a central repository is a key aspect.

There are many different ways to organize our source codes within the VCS; however, for a manageable repository layout, the Subversion project recommends the idea of a “project root”. For each component, as well as for the entire GENiC platform, there will be a project-wide root directory which will include all the developments of GENiC for a specific component or functionality.

These “project root” directories contain three common subdirectories:

- **/trunk:** contains the main and stable line of development,
- **/tags:** includes frozen copies of the trunk, software releases or notable revisions in the history of the repository,
- **/branches:** is the side-line of development, can be used to create development lines for multiple versions of the same product, after the merging of different development lines or the resolution of bugs into the stable trunk.

In our layout each sub-system (component) will have an independent sub-folder with its own *trunk, branches, and tags* folders.

Source Control updates

Updates will occur when a new functionality or new feature is implemented, an open bug is fixed or an old functionality is enhanced. The general rule while developing code is to update as early and as often as possible. With centralized repositories it is good to update often to avoid future merge conflicts.

Each developer should, at least, update **before modifying** the source code to ensure they are working with the latest version of the code and update **while modifying** to ensure seamless integration and to minimize merging problems. Updating the code **before and after committing** ensures that the main development line stays in good working condition and also prevents mixed revisions within the working copy.

Source Control commits

Each commit in the source code management tool creates a revision that allows developers to look back though each individual revision. The best practice when working with a common centralized repository is to perform one conceptual change at a time. It is important to avoid too many different modifications at the same time as it is difficult to keep track of chances if there are many conceptual implications for the same commit. In that sense, every commit should come with a message describing what has been modified from the previous version.

The most important rule to keep in mind is related to the stability of the code after your commits. When a change needs to be committed you should at the very least ensure that the new source code compiles to ensure that you are not blocking the progress of other developers.

Source Control branching / tagging / merging

To guarantee the stability of the /trunk while developing, the chosen strategy of using branches and tags, rather than prescribing a universal policy, depends on the approach chosen for each software component. The common approach that fits best with our GENiC components is the **Branch-When-Needed** system where developers work in a stable development line and then create a new branch when it is not possible to make a series of small commits without disrupting the stable development line. This approach guarantees stability at all times while adding only a small burden to developers' daily work as they must compile and test to ensure stability before every commit. For components that have been implemented by more than one development team, the "merge" command is a very useful action that applies the differences between two sources to a working copy path.

The stability of the source code is a key aspect when continuous integration needs to be applied. The stable line must at least compile and the continuous integration scripts will run over this stable directory.

For releases a copy of the stable /trunk will be tagged into /tags. The tag name needs to be as descriptive as possible as it will be used to keep track of the important milestones of the component. Spaces are not allowed and prefixes are recommended to maintain the historical information about releases.

4.3 Documentation Guidelines

A separate SVN has been provided for document control and management. This SVN is administered by the GENiC Coordinator and all partners are provided with full read and write access.

Find bellow the high level directory structure:

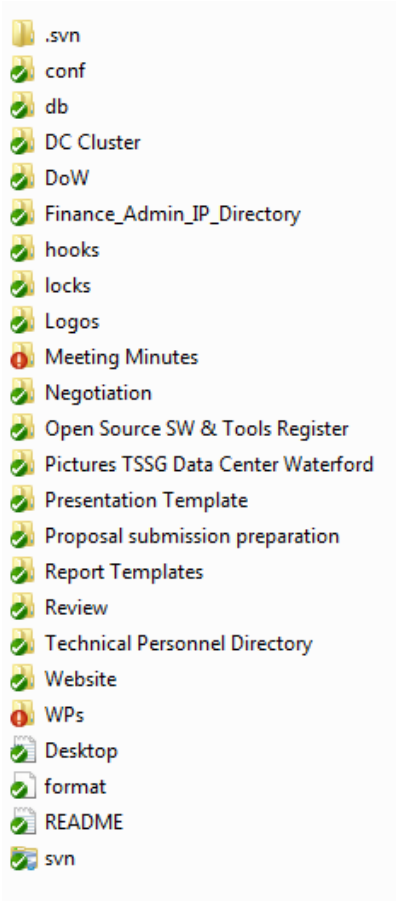

Figure 3 Documentation SVN high level structure.

Further information about the GENiC directory structure and its content can be found in deliverable D.8.2 GENiC Project Management Portal [5].

In addition to our common documentation repository a set of document templates have been prepared in order to gather a detailed view of the GENiC components accompanying the official deliverables. At the end of this deliverable, you will find in the annex (Annex A: Document Templates) the following documents:

- [GENiC_Component] User Guide
- [GENiC_Component] Installation Guide
- [GENiC_Component] Readme file

4.4 SW Delivery Guidelines

Each development team will work independently on their specific component or on an enhancement of a previous component functionality. This work will be regulated by an Invention Disclosure Form (IDF) which will be used to assess ownership of intellectual property rights.

Some of the components cannot deliver the sources due to proprietary software dependences. In these cases the development teams need to provide component releases together with documentation that describes the installation, the configuration as well as the usage of the component.

You can find in Annex B: Configuration & Usability Survey a component survey. The purpose of this survey is to capture GCG and GC installation / configuration and usability requirements early on for each of the components within the GENiC architecture.

5 Integration Guidelines and Toolset

5.1 Integration Toolset

Source Control Management (SCM) refers to the use of a system to help developers keep track of the version history of their source codes, parallel versions and releases. **Apache Subversion,** called SVN, is the software versioning and revision control system used in our GENiC collaborative development environment.

```
http://it-forge.atosresearch.eu/svn/genic/trunk
```

Bugzilla is a bug-tracking system and testing tool licensed under the Mozilla Public License. This tool will provide in the scope of GENiC a clear as well as centralized overview of development requests and their states. It also keeps track of software problems and resolutions. This allows developers, through the diligent use of the bug system, to detect and correct issues during the development cycles as well envisage possible inconsistencies, giving teams the ability to detect problems early and mitigate them in early stages.

```
http://itforgebugzilla.atosresearch.eu/bugzilla/
```

Maven [13] is a software management tool than can manage the build cycle of SW components. Maven attempts to simplify the software build process allowing developers to comprehend the complete state of a development effort in a short period of time. It is based on the concept of a project object model (POM) and licensed as open-source under the Apache License Version 2.0.

```
http://maven.apache.org/download.cgi?
```

Jenkins provides an easy-to use continuous integration system allowing developers to build/test their software projects continuously as well as monitor the executions of externally-run jobs. The CI server is written in Java and licensed under the MIT license. It has an enormous community that contributes plugins to the CI system.

5.2 Build Automation Guidelines

This section is based in the Apache Maven management software tool and Java-based components to give a realistic example of how the component build process is performed. Therefore, we introduce here tools and recommendations for testing, building and deploying components in order to facilitate the continuous integration process also introduced in this section.

The Apache Maven tool allows the automation and execution of a build every time that a developer triggers the build or even when changes in the repository are produced. Developers can include their test in this automated build in order to minimize bugs when new changes are deployed in order be able to verify the correct functionality of the change made. Ideally, if developers keep the build as fast as possible it helps ensure a smooth, timely and good integration of the components making it easier for anyone to get the latest stable version of the component. At the deployment level, Maven also includes a set of plugins to automate deployment/undeployment of components or applications which helps to avoid human errors when there are upgrades or releases of new versions of SW components.

Jenkins CI server offers the mechanisms to automate the build management, release management, deployment automation and test orchestration of a SW project. Initially Jenkins was called Hudson, developed by Sun Microsystems [14], and afterwards the code base was inherited by Oracle [14] who tried to change the way the project was managed. Tensions between Oracle and most of the main developers as well the community lead to a project fork called Jenkins. Jenkins is managed by most of the original developers including the founder. It provides a large variety of plugins to extend the base functionalities creating a versatile tool to support the communities' needs for continuous integration.

Below are some of the most common practices for continuous integration:

- Continuous integration is a matter of attitude and collaboration rather than tools. With the right attitude and tools, CI can minimize the duration and effort required for each SW iteration within the agile SDLC.
- The use of control version tools such as CVS, SVN, and/or Git is strongly recommended to create revision checkpoints.
- The ability of the teams to create and automate builds for their SW components in addition to the capability to automate the release process helps to improve the components integration process and the time required for delivery.
- The build process can be instrumented to deal with unit, acceptance and integration tests towards a test driven deployment, allowing for the early detection of bugs or inconsistencies in the SDLC.
- Alerting teams of test failures or broken builds accelerates their resolution in order to reach a stable baseline in a short period.
- The use of a CI server which automates and triggers the testing, building and integration process definitively facilitates the management and automation of the aforementioned best common practices in integration environments.

5.3 Testing Guidelines

The objective of testing is the early detection of problems and verification of software functionalities under specific conditions. The Test Driven Development (TDD) technique repeatedly drives the development process through three steps. Developers must write test for each bit of functionality and modify the functional code until the test is passed while the old and new code is refactored continuously to make it well structured.

The test and validation of the GENiC platform will be reported in D.5.5 Verification and validation [3] of the GENiC platform after the release of the GENiC integrated platform. Despite this, continuous integration and testing are continuous activities during the entire project life-time.

The validation can be divided in two aspects:

- **Functional validation:** to demonstrate the expected software functionalities work under different conditions and configurations.
- **Non-functional validation:** to validate the quality of the solutions, including the robustness, efficiency and extensibility.

To cover the validation, four levels of testing will be implemented:

- **Unit test:** verifies the functionality of each GC in isolation.
- **Integration test:** verifies the interaction between different GCs.
- **System test:** evaluates the behavior of all components within the architecture at the same time.
- **Deployment test:** evaluates the functionalities of the platform when all the components are deployed in the GENiC test facilities.

Further details about the testing procedures and tools will be included in the subsequent deliverables included in this work package.

6 Collaborative Development and Continuous Integration Environment FAQs

Requesting access to the GENiC Collaborative Development Environment

First of all you need to register as a user of the GForge system.

(http://it-forge.atosresearch.eu/account/register.php)

After approval, you can request access to the GENiC Collaborative Development Environment:

(http://it-forge.atosresearch.eu/project/request.php?group_id=[projectID])

Checking out a working copy of the SCM system

Only GENiC developers can access the SVN repository. You will be asked for your username and password. Type them when prompted.

```
svn checkout --username developerName http://Subversion_URL
```

However, the recommended practice is to integrate the SVN with your IDE. Other clients are available, and some examples of both type of tools are listed below:

- Subclipse for Eclipse IDE (http://subclipse.tigris.org)
- RapidSVN (http://rapidsvn.tigris.org/) – Multi-Platform
- TortoiseSVN (http://tortoisesvn.tigris.org/) – Windows only

SCM Year 1 development line

Below is the main development line for the GENiC platform during Y1.

(http://Subversion_URL/svn/genic/branches/GENiCY1)

Updating your local working copy

```
svn update --username developerName Local Working Copy
```

SCM commit changes

```
svn commit http://Subversion_URL –m "descriptive message"
```

SCM branching / tagging

One of the features of the VCS is the ability to isolate changes as a separate line of development.

How to branch:

```
svn copy http://Subversion_URL/trunk
http://Subversion_URL/branch –m "descriptive message"
```

How to tag:

```
svn copy http://Subversion_URL/trunk
http://Subversion_URL/tags/ReleaseName –m "descriptive message"
```

Subscribing to a project mailing list

To receive messages from the collaboration mailing lists, you have to request access for the mailing lists of interest.

(http://lists.it-forge.atosresearch.eu/mailman/listinfo/[LIST_NAME])

Requesting access to the bug tracker system

First of all you need to register as a user in the bug tracker system.

http://itforgebugzilla.atosresearch.eu/bugzilla/createaccount.cgi

When your access is granted you will be able to fill bugs for GENiC components. (See section 6)

How to fill a bug

In order to fill a bug you have to sign into the bug tracker system.

http://itforgebugzilla.atosresearch.eu/bugzilla/

The next step is to fill the form for the GENiC component you want to notify of an open issue or bug. Each component within the architecture will have a default assignee who takes care of the requests for the component. It is possible to extend the list of receivers to make other people aware of the open discussions during the bug resolution.

7 References

[1]. D1.2. Requirements & Draft Architecture http://www.projectgenic.eu,

[2]. D.5.3 GENiC integrated platform (initial version) http://www.projectgenic.eu,

[3]. D.5.4 GENiC integrated platform (final version)http://www.projectgenic.eu,

[4]. D.5.5 Verification and validation http://www.projectgenic.eu,

[5]. D.8.2 GENiC Project Management Portal http://www.projectgenic.eu,

[6]. SourceForge http://www.sourceforge.net,

[7]. Apache Subversion http://www.subversion.apache.org,

[8]. GForge AS User Manual http://read.pudn.com/downloads91/doc/347382/gforge_user_guide.Eng.pdf,

[9]. "Agile Software Development, Principles, Patters , and Practices", Robert C. Martin

[10]. "Continuous Integration", Martin Fowler, http://martinfowler.com/articles/continuousIntegration.html.

[11]. "Continuous Integration", Paul Duvall, Steve Matyas, Andrew Glover http://www.martinfowler.com/books/duvall.html,

[12]. Mailman User Guide http://www.gnu.org/software/mailman/users.html,

[13]. Maven guides http://maven.apache.org/guides/,

[14]. Oracle and Sun Microsystems http://www.oracle.com/us/sun/,

[15]. Continuous Integration with Jenkins http://www.vogella.com/tutorials/Jenkins/article.html,

[16]. Bugzilla http://www.bugzilla.org,

[17]. Maven release plugin. http://maven.apache.org/guides/mini/guide-releasing.html

8 Annex A: Document Templates

A.1 User Guides

The user guide template is described below. The component owners may include additional information that they consider relevant to enable the use of the component by end-users or developers.

<Component Name> User guide

Release Information

Component Name	Release Number	Release Date	License

Introduction

- ✓ *Brief description of the component. What does this component do and who does it serve?*
- ✓ *Brief explanation of the structure of the document.*

Configuration

If the component needs any configuration, include in this section what has to be configured, where and how.

Feature 1 to N

List for each of the features provided by the component include a step-by-step describing how to use it.

- *Interfaces provided*
- *Which are the minimum parameters required*
- *List the return value types.*
- *Usage of the feature*
- *Testing of the feature*

Known limitations

This section will include known limitations of the component as well as open bugs that have been identified.

If it has been identified describe how to solve or workaround the limitation or bug.

A.2 Installation Guides

The installation guide template is described below. The component owners may include additional information that they consider relevant to enable the installation of the component by end-users or developers.

<Component Name> Installation guide

Release Information

Component Name	Release Number	Release Date	License

Minimal System Requirements

List hardware requirements to use the component such as: processor, memory, free disk space, OS, network bandwith.

Supported platforms

List in this section the operating systems (OS) tested.

Software pre-requisites and dependencies

List here the minimum set of software to use the component. In this section will be described the software dependences with other components, other products, or 3rd party libraries.

Installation

Describe here the step by step instructions to install and configure the component.

Getting started

Define examples of usage or reference the user guide if it exists.

Known limitations

This section will include known limitations of the component as well as open bugs that have been identified.

If it has been identified how to solve or workaround the limitation or bug, please provide this information in this section.

FAQ

This section will include known issues during the installation process identified in different platforms. This section will also provide solutions to the known issues.

A.3 Readme Files

The GENiC components must be delivered together with a readme file containing useful information about how to install and configure the component, as well as how to get started with the component. In addition, the readme file has been used as release changelog, keeping track of the changes on every new release of the components.

See below the README file template proposed:

```
#####################################################################
#                            GENiC                                  #
#-------------------------------------------------------------------#
#      ( Globally optimized ENergy efficient data Centres )         #
#####################################################################

=====================================================================
<Component Name> <Version Number> README
=====================================================================
```

```
// Please use this Readme template as a Readme and Release Note
// for all new GENiC software you are releasing.

Component  Name
Software Release Number (X.X.X)
Release Date of the Software

=========================================================================
DESCRIPTION
=========================================================================
// Provide a description of the component:
//  - Functionality.
//  - Features.
//  - Intended Applications.
//  - Compliance standards.

=========================================================================
RELEASE CHANGELOG
=========================================================================
// List features and functions that are new in the release.
// Feature name
// Description

=========================================================================
BUG FIXES
=========================================================================
// List important fixed bugs in the release.
// Bug #
// Problem Description

=========================================================================
<Component Name> <Version Number> SETUP
=========================================================================

HARDWARE REQUIREMENTS
-------------------------------------------------------------------------
// List hardware requirements for the component.


PLATFORM SUPPORTED
-------------------------------------------------------------------------
// List the operating systems (OS) supported.

SOFTWARE DEPENDENCIES
-------------------------------------------------------------------------
// List software dependences with other components,
// products, libraries


COMPILATION INSTRUCTIONS
-------------------------------------------------------------------------
// List compile instructions for the different platforms.


INSTALLATION INSTRUCTIONS
-------------------------------------------------------------------------
// List installation steps for every supported platform.
// Refer to the installation manual if there are any available.

DEPLOYMENT INSTRUCTIONS
-------------------------------------------------------------------------
// Describe the deployment steps on how an end-user or developer
// can use the software.

TESTING INSTRUCTIONS
-------------------------------------------------------------------------
// Describe how to run automated tests
// Describe Unit tests for the component
// Describe Integration tests for the component


=========================================================================
CONTRIBUTORS
=========================================================================
// List main developer and contributors names.

=========================================================================
```

```
KNOWN LIMITATIONS
=================================================================
// List known limitations.

Limitation #
Workaround

=================================================================
LICENSE AND COPYRIGHT
=================================================================
//Please place a copy of the license here
```

9 Annex B: Configuration & Usability Survey

The purpose of this survey is to early capture GCG and GC installation / configuration and usability requirements for each of the components within the GENiC architecture, in order to produce development and integration guidelines in the scope of D5.1 Development & Integration guidelines including integration environment & means.

The component owners need to fill the survey as detailed as possible per component, including the envisaged interactions with other GENiC components as well layers within the architecture reference.

The Configuration & Usability survey is a life document that will be refined during the project life-time at least one time per component release.

1. NAME AND BRIEF DESCRIPTION OF THE COMPONENT (GC).	
2. HOW YOUR COMPONENT WILL BE DELIVERED? (JAR, WAR, TARBALL, RPM-BASED, etc)	
3. WHAT IS CURRENTLY CONFIGURABLE FOR YOUR COMPONENT (First Release)?	
4. WHERE AND HOW YOUR COMPONENTS NEED TO BE INSTALLED / DEPLOYED? **(WEB SERVER, PUBLIC REPOSITORY, DECOMPRESSED IN THE FILE SYSTEM)** **(DESCRIBE as much as possible STEP BY STEP GUIDE)**	
5. MAIN PROGRAMMING LAUNGUAGE OF YOUR COMPONENT.	
6. LIST THE SW DEPENDENCIES OF YOUR COMPONENT(3RD PARTY DEPENDENCES, GENIC DEPENDENCES)	

7. LIST THE INTERACTIONS WITH OTHER GC OR GCG.	
8. BASED ON 3), HOW CAN THE CONFIGURATION BE IMPROVED? WHAT ELSE CAN BE MADE CONFIGURABLE IN YOUR COMPONENT AT THE END OF THE PROJECT?	
9. CAN YOU THINK OF ANY OTHER USABILITY & CONFIGURABILITY REQUIREMENTS FOR YOUR COMPONENT?	
10. CAN YOU THINK OF ANY USABILITY & CONFIGURABILITY REQUIREMENTS FOR THE OTHER COMPONENTS, OR FOR THE OVERALL SYSTEM?	
11. LICENSE TERMS OF YOUR COMPONENT	
12. ESTIMATE as detailed as possible A RELEASE TIMEPLAN FOR YOUR COMPONENT	
13. DESCRIBE THE CHANGELOG WITH THE PREVIOUS RELEASE (if it apply)	