12-12-2012

GN3 Software Documentation Best Practice Guide

Last Updated	12-12-2012
Activity:	SA4
Task Item:	1
Nature of Deliverable:	R (Report)
Dissemination Level:	PU (Public)
Document Code:	GN3-12-330 v2
Authors:	Branko Marovic (AMRES), Marek Lewandowski (PSNC), Andrzej Bobak (PSNC), Ognjen Blagojevic (AMRES) Ljubomir Hrboka (CARNET), Tihana Žuljevic (CARNET), Paweł Kędziora (PSNC), Kostas Stamos (GRNET) Angel Panov (BREN), Tomasz Krysztofiak (PSNC), Jakub Szubarga (PSNC), Gina Kramer (DANTE)

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 2007–2013) under Grant Agreement No. 238875 (GÉANT).

Table of Contents

Table of Figures

Table of Tables

Introduction

This guide should be used by all roles involved in software development documentation (e.g. developers, managers, technical authors etc.) and has been written as part of Task 1, SA4. Note that the material presented here should be viewed as a guide, in practice, each document created will have very specific user needs and constraints that will determine its ultimate shape and form. In addition, this guide can be also used as basis for analysis of documentation aspect within QA audits performed by SA4, Task 2.

This GN3 Software Documentation Best Practice Guide provides recommendations on how to draft and maintain documentation associated with the use and development of software. This includes:

- An outline of typical document types.
- Internal documentation.
- Code-related documentation, including source code.
- Public-facing documentation.
- The structure of public-facing documentation, including material for administrators and end users.

There are a number of points to bear in mind when creating documentation:

- Gather notes about architectural decisions and design peculiarities while the software is being designed and developed.
- Document information thoroughly once the software is stable.
- Good documentation is also a great advertisement for quality code, self-documenting code can help with this (ref to section).
- A clear, concise approach is always preferred. Break instructions down into easy-to-follow steps, etc.
- Ensure your docs are up-to-date and accurate. If there are changes in the way that something works, make sure that this is reflected in documentation updates.
- Try to illustrate complex material as much as possible.
- Have a clear purpose or objective in mind when drafting a document.
- Try to imagine who the reader might be and match his/her level of technical expertise.

In terms of document maintenance and storage, The Y3 GN3 review recommended that all software documentation should be stored in the common GN3 Forge repository. Use of cloud infrastructure for future document management is also an area for future investigation.

Although drafting documentation to support a software or service release is different from that of a report presenting an outcome or process, guides and technical authors are available to help, [PMO] This is covered in more detail in Section 5, Public-Facing Documentation Structure.

1 Documentation Types

Documentation in GN3 is split into two categories:

- **Internal documentation** (see Section 2 and 3)

  Documentation that is created during the software-development lifecycle by software development teams but not distributed publicly, for example:

  - Requirements
  - Design documentation
  - Specifications of data model, formats and algorithms
  - Code documentation

  This type of documentation may be created by any member of a GN3 software developer team and can be more flexible in its format and content than public-facing documentation.

- **Public-facing documentation** (see Section 4 and 5)

  Documentation that is publicly circulated for released software, for example:

  - Deliverables
  - ReadMe
  - Release notes
  - User guides
  - Admin guides (e.g. installation, configuration and maintenance)
  - API guides
  - Help (e.g. online help, context-sensitive help)
  - Context-sensitive help
  - Web content
  - Marketing material (e.g. brochures, datasheets, white papers)

It is important that this documentation complies with the standards set by the GN3 project office [**PMO**].

All GN3 software documentation should be stored in a common GN3 Forge repository. In order to meet this requirement, a common template for GN3 Forge workspaces has been provided:

- Main menu
  - Home
  - Features
  - Screenshots
  - On-line Demo
  - Downloads
  - Documentation
  - Feedback Area
  - FAQ
  - Contact
  - Registration

- Project Lifecycle
  - Change Log
  - Release Plan
  - Bug Reporting

- Developers' Menu
  - Developers Home
  - Issue Tracker
  - Source Code Management
  - Maven Repository
  - Continuous Integration

Main menu and Project Lifecycle sections are publicly available. The Developers' Menu is available for the use of GN3 members. Administrators of particular workspaces can add new areas to this structure.

2 Internal Documentation

Internal documentation is created and used by software development teams and stored in the GN3 Forge repository. It is created and maintained throughout the development lifecycle and comprises:

- Software requirements specifications.
- Architecture / design specifications.
- Internal technical documentation.

2.1 Software Requirements Specifications

The requirements that need to be fulfilled by software are usually related to the complexity of the product, its impact and life expectancy. Most of the software used as part of the GN3 Project is of moderate complexity (software requirements may be modest, but are important). If the software is a release that is later built upon, requirements specifications are helpful in managing changes (and for verifying that nothing has broken during the modification). Usually, this type of the documentation is provided as a text-based document (e.g. scenarios, use cases, etc.).

The Software Requirements Specification (SRS) documentation should be stored on the internal portal for developers and maintainers (GN3 Forge Developers Menu -> Developers Home section), or on the website, if open source software.

The SRS document [**REQ1**] should contain the customer's requirements and dependencies identified at a particular point in time. It may identify system or sub-system software requirements. In order to make this document self-contained and facilitate the interpretation of individual requirements, it should start with an adequate explanation of the system's purpose, key functions, and users.

The use cases of the requirements specification should describe the software features, operating and user environment, external interfaces, and non-functional requirements. The individual requirements should cover the following areas:

- **Functional Capabilities** – Summary of the major functions that the software will perform, to be built around the use cases.
- **External Interfaces** – Descriptions of user, software, hardware and communication interfaces.

- **Performance Levels** – The speed and response time of particular software elements.
- **Data Structures and Elements** – Description of data structures (and their dependencies) that must be used in the implementation of the application.
- **Safety** – Considerations related to the safety of the software.
- **Reliability** – The recovery time and availability of particular software elements.
- **Security and Privacy** – Requirements regarding the security and privacy of data, includes proposed mechanisms and protocols providing both security and privacy.
- **Quality** – Mechanisms applied to guarantee desired functional and structural quality of the software.
- **Constraints and Limitations** – Hardware and software requirements or limitations, related operating environment or power consumption constrains, communication protocols, regulatory limitations, standards, limitations regarding the implementation environment or languages, real-time constraints (for real time applications), audit requirements, etc.

All requirements should be valid and clearly described to ensure clear interpretation. They should also be verifiable and able to be tested in a way that satisfies the unambiguous assessment criteria of the requirements [**REQ2**].

Requirements specifications should contain following:

- **Title page** – the title page should contain product and project logos, the name of the software and information about who collected the data.
- **Overview** – this section should provide information about the document's purpose, motivation and intended audience.
- **Overall description** – this section should provide general information about the project and the organisations involved.
- **Hardware requirements** – if hardware requirements are crucial (e.g. multi-processor environment), they should be described here.
- **Software requirements** – all required software (e.g. application server, database server) should be listed here. Requirements related to the licences of third-party software or reused components should also be described.
- **Other requirements** – other requirements that do not directly relate to hardware or software should be listed here (e.g. time of response, colours or fonts of the GUI).
- **Stakeholders** – in this section, all stakeholders have to be described in detail.
- **Use cases** – this is the most expanded part of the requirements documentation. Use cases present all usage scenarios in a detailed manner. These are created by clients (end-users) and software analysts. Use cases are written in a human language and should contain following sections:

    - Author of the requirement.
    - Stakeholders that take part in this scenario.
    - Description.
    - Step-by-step procedure.
    - Exceptions.

  - Priority.
  - Frequency of use.
  - Notes.

2.1.1 Examples

Software Requirements Specifications

- https://intranet.geant.net/sites/Services/SA2/T5/cNIS/Documents/design/cNIS_1.0-2.0_requirements_analysis-0.42.doc
- https://wiki.man.poznan.pl/perfsonar-mdm/images/perfsonar-mdm/a/ab/GN2-04-150v14.pdf
- https://intranet.geant.net/sites/Services/SA2/T5/iSHARe/Documents/I-SHARe-Specifications-v3.3.doc
- https://intranet.geant.net/sites/Services/SA2/T5/cNIS/Documents/design/cNIS_3.0_MPLS_functional_requirements.docx

2.2 Architecture / Design Specifications

Software architecture documentation should be located on an internal portal, for use by developers and maintainers (GN3 Forge Developers Menu -> Developers Home section), or on a website, if open source software.

Software architecture is a set of structures composed of software elements, properties of those elements and their relationship. Transformation of requirements into software architecture has a tremendous impact on software development.

Without proper architecture documentation, communication between various stakeholders (requirements engineers, software architects and designers, implementers, testers, integrators, etc.) would be impeded. The importance of this documentation is especially evident on long-term projects, when old participants leave and new participants become involved in the project. In this instance, architecture documentation serves as a form of training. Another aspect of architecture documentation is as a support tool for system analysis, as it should contain all necessary information for performance, usability and other types of testing.

While preparing architecture documentation, one must take into account following principles:

- Documentation should be easy to read.
- Documentation should be reviewed and updated at regular intervals.
- Documentation should be easy to find.
- Any notation used within the documentation should be explained.

Documentation should contain descriptions of layers, tiers, (and their interaction), protocols and interfaces used.

Layered architecture organises software in logical sub-level components (layers) built one upon another. Most common layers in multi-layered architecture are:

- The user-interface layer, for interaction of users and application.
- The application layer, which separates business logic from the UI layer.
- The business layer separates business logic from other layers.
- The infrastructure layer (or data-access layer) provides access to stored data or a network.

Layers can be presented as a stack of rectangles or set of concentric rings, such as those depicted in Figure 2.1.

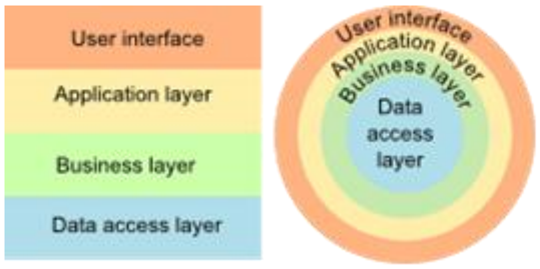


Figure 2.1: Example of layered architecture

Tiered architecture organises software into physically separated, client-server divisions. For example, the most common three-tier architecture consists of:

- The presentation tier, which displays information and interact with clients.
- The application tier, where processing is being performed.
- The data tier, which usually consists of database servers.

Tiers are commonly presented as boxes, and usually connected with two-way arrows expressing the relationship between them, as seen below in Figure 2.2.

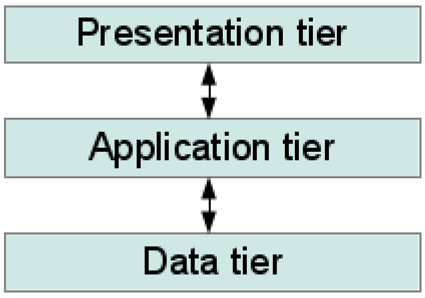


Figure 2.2: Example of three-tier architecture

Components of the system communicate with each other via interfaces. Documentation of the interfaces should contain:

- The component's or interface's name.
- Brief description.
- Resources (e.g. methods) they provide with syntax, semantics and usage restrictions.
- Data type definitions.
- Configuration parameters.
- Quality parameters, such as performance and reliability.
- Design issues.
- User guide.

Unified Modelling Language (UML) documentation should be used, where appropriate. The tendency to misuse UML is quite common. Diagrams must be carefully selected and stripped of irrelevant information.

Architecture specifications describe why specific routines should be designed in a specific way. Approaches for lower-level design may be included, but actual studies are usually detailed in other documents.

Architecture / design specifications should contain the following:

- **Title page** – the title page should contain product and project logos, software name, version number, release date and information about the main developers (including contact information).
- **Overview** – this section should give a brief overview of the software, the motivation for the software, what the software can be used for and its position in the literature. In particular, the features offered by the software package should be stated. The overview should not be written in a strictly technical language, since it can be addressed to non-technical people.
- **Design** – this section should present the overall design of the software at a high level. Used design and programming methodologies should also be stated. Not only overall architecture, but design patterns and frameworks should also be involved.
- **Packages** – in this section, any external packages that are used should be listed, including each package's purpose and functionalities used in the developed software. Modules that are required (and thus used) by the external packages should also be listed.
- **Description of modules and APIs** – this section should describe the available modules, executables and the API. Prototypes of all APIs should be given and where important the internal workings should be briefly described. The crucial aspect is to provide detailed description of interfaces of the modules.
- **Testing** – this section should document the tests that are used to test the software. Links or information about the tests' location should be given.

2.2.1 Examples

Architecture Specifications

- https://wiki.man.poznan.pl/perfsonar-mdm/images/perfsonar-mdm/9/95/GN2-05-057v5.pdf

- Architecture section of https://intranet.geant.net/sites/Services/SA2/T5/iSHARe/Documents/I-SHARe-Specifications-v3.3.doc

3 Code-Related Documentation

There are baseline requirements for the documentation of software specifications, software architecture, source-code documentation, data model, algorithms, structures and formats, which are detailed in the following sections. Note that only open source projects require code-related documentation to be made public.

Internal technical documentation explains the source code used in the software. When creating software, code alone is not enough, and some text-based information is necessary. As maintaining internal technical documentation is quite difficult, automation is essential. Tools such as Javadoc or Doxygen make it possible to write this documentation as source code comments. The output format is usually in a reference guide style. Internal technical documentation should be created using the same tool as for the regular source code, so that the programmer can directly refer to his/her code.

There are different forms of technical documentation. This is due to the different purposes of the documentation (description of the source code, database tables, etc.) and different tools used to create documentation.

- **Source code documentation** should be written in parallel to the code itself, so it is important to create this documentation using the same tool, to make this process as fast as possible. The output of the source code documentation should be generated automatically (with each release) and presented in an organised structure (with links and indexes), e.g. HTML format.

  The quality of the source code documentation depends on developer effort. Therefore, regular effort is crucial. The following source code elements should be documented:

  - Classes and interfaces (author, date, aim).
  - Fields.
  - Methods (documentation of getters and setters is not obligatory).

- **Database documentation** can be limited to a legible EER (Enhanced Entity-Relationship) diagram and a description of tables, attributes and constraints. The database documentation creation process can be supported by open source software. Existing, free EER software tools include: fabFORCE DBDesigner [**DBD**], MySQL Workbench (both designed to work with MySQL RDBMS) and SQLPower Power Architect [**PARCH**] (designed to work with PostgreSQL and other RDBMSs). Aside from the ability to export EER diagrams, and DDL scripts for particular RDBMS, which are crucial to understand the data layer, it is also sometimes necessary to produce the documentation with detailed description of every table and field of the database. Some of the tools mentioned are also equipped with this ability.

- **Description of the algorithms and data structures** – all implemented algorithms and non-standard data structures should be documented in the source code and as an external text file. This enables the information to be retrieved and easily published when needed. For algorithms, mechanisms and computational complexity (time and space) should be provided.

- **Release notes** – when software is developed in an iterative approach, each release should be published with basic information about new features, removed bugs, etc. An Agile approach assumes that release notes are automatically generated. Automation can be easily achieved when using an advanced issue tracking system (e.g. JIRA).

3.1 Source Code Documentation

Source code documentation should be located on the Javadoc folder tree or Implementation file (if single) on the Internal portal for developers and maintainers (GN3 Forge Developers Menu -> Developers Home section). Open source software should be made available on the website.

High-quality source code documentation is crucial for long-term development and smooth software reuse, as it helps developers to understand what the code does and how to use it. The most efficient way to prepare documentation is to write code documentation in the source code, using conventions that are appropriate for the language (for example, Javadoc for Java). This approach brings two essential benefits. First, it is easy to maintain. Because the code and its documentation are in the same place, any changes in the code can be immediately reflected in the documentation. Second, such documentation can be automatically processed to generate standardised, easy-to-read text for humans and other applications (e.g. Integrated Development Environments – IDEs), source code documentation.

Inline source code documentation is also convenient for developers using “bleeding edge” code, or a specific snapshot of software that is not an official release. Developers can easily generate valid documentation for this snapshot from source code alone, using available (usually free) tools, without having to wait for an official release or update of online code documentation.

In open source projects, it is good practice to release a source code package along with binary release (as a separate package, or, less common, in the same package as binaries). Such a package can be used by IDEs to show developers both the source code and its documentation.

It is highly recommended to write code documentation and also provide the online documentation generated from it. In this way, publically available source code documentation may always be up to date with the latest software release. This task may be easily automated by building continuous integration tools.

Best practices on writing source code documentation can be found in Section 5.2 Coding Guidelines, of Software Developer Best Practices.[**Error! Reference source not found.**]

3.1.1 Examples

Source Code Documenation

- Source code of perfSONAR services: svn+ssh://svn@svn.geant.net/GEANT/SA2/ps-java-services/tags

3.1.2 Data Model Specification

The data model should be located on the internal portal for developers and maintainers (GN3 Forge Developers Menu -> Developers Home section), or the website (if open source software). This model provides a specification of the data structure, describing the type of data held, its organisation, and the way in which it will be accessed. It is an abstract model used as a plan for application development, and it informs developers about data structures used by system and their mapping within the business domain.

Data architecture describes the architecture of the data structures used by an application. For a given system, it describes how data is processed, stored and utilised. Providing a tool-generated chart extracted from a database or ORM objects is insufficient for a complete data model.

While models, architecture and charts are useful, they must include clear and concise descriptions of classes, entities and tables, as well as their attributes and internal and mutual constraints. The provided diagrams should not take the form of large sheets with dozens of objects, but instead, be split into partially overlapping sections. Data model and architecture descriptions need to document both data in storage and data in motion.

It is important to consider data model documentation from the following aspects:

- A well-documented data model can have a great and practical impact on simplifying programming.
- A data model can help a software developer to gain an in-depth understanding of business requirements.

A poorly documented model could result in erroneous interpretations from various developers, which can result in inconsistent and unreliable data.

Three different data models should be produced as part of data model and database specification:

- A **conceptual data model** (Figure 3.1) is usually presented as a diagram depicting the high-level relationship between all business entities. A conceptual data model is a set of technology independent specifications about the data. It is used for discussion of initial requirements and its creation is the first step in organising the data requirements. It abstracts implementation details and provides graphical notations that describe entities, their relationships and any constraints between them. The basic elements of a data model are boxes (entities) and arrows (relationships). The best way to document complex data entities are data structure diagrams. A conceptual model can be used as a basis for a logical data model.

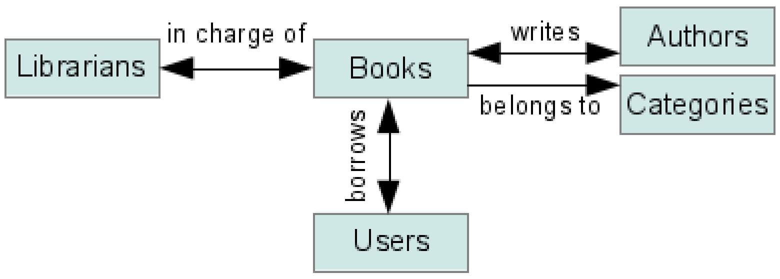


Figure 3.1: Conceptual data model example

- A **logical data model** (Figure 3.2) consists of descriptions of structures (tables, XML, object-oriented classes, etc.) and their relationships, which can be implemented by the means of a database management system.

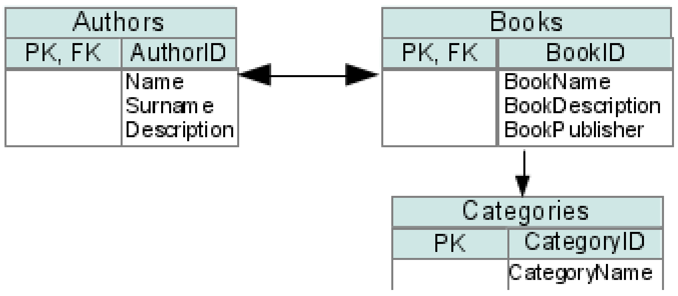


Figure 3.2: Logical data model example

- A **physical (implementation) data model** should describe the realisation of the data mechanisms, including physical storage, partitions, CPU, etc.

A data structure is an efficient way of storing data. It facilitates the organisation of logical and mathematical concepts of data. A carefully chosen data structure often facilitates the most efficient algorithm. Use of proper data structure also makes a significant difference to the running speed of program. An algorithm is usually documented with standard flowchart symbols. If a large system is to be documented, its subroutine is presented in a single block with a reference to the sheet where logic is detailed. It is not sufficient to fully characterise a large system. Data structures (database tables) and the flow of data should also have to be documented with flow diagrams. In this way, a system can be visualised as a tree structure, where the whole system is presented as the top node, the data structure, data flow and the logic are presented as tree nodes below the top, and more granular structures extend from each of these nodes. [**Merson**], [**Simsion and Witt**].

3.1.3 Algorithms, Structures, and Formats

Algorithms, structures and formats should be stored on the internal portal for use by developers and maintainers (GN3 Forge Developers Menu -> Developers Home section), or on the website, in the case of open source software.

The description of the algorithms, data structures or formats must be organised in a modular way, with clarification of each of the individual software modules in order to provide a description of the algorithms, structures or formats of each module.

The developers need to describe the algorithms and related complex data structures or formats that require a detailed explanation beyond the one found in the source code documentation, and only if there is some novel or crucial solution requiring detailed elaboration.

3.1.3.1 *Document structure*

The proposed grouping, if such documentation is deemed necessary, is as follows:

- Algorithms; subsection <Algorithm name>
- Data structures; subsection <Structure name>
- File/Message formats; subsection <Format name>

Conceptual and high-level descriptions should be included in the software architecture/design specifications, while low-level implementation descriptions belong to code-related documentation.

The most common tools used to describe an algorithm are pseudocode and flowcharts. Although flowcharts were the design tool widely used in the past, pseudocode has become the preferred tool, since it provides a better reflection of the structured programming concepts. Despite its proposed use, describing an algorithm with pseudocode is a process that requires compliance with certain rules, since a global set of syntax rules is not available. Before focusing on some general rules for writing pseudocode, it is important to summarise relevant issues:

- A reader should be aware of the programming language described by the pseudocode.
- Pseudocode should be simple and easy to follow, as well as be complete.
- Pseudocode should not follow details of the respective language syntax.
- If the author considers it necessary, external descriptions of possible complex structs etc. can be used.
- The author should explicitly state any possible syntax conventions.

There are some general rules an author should follow to ensure the correct pseudocode syntax: **[PSDRULES]**

- Include only one statement per line.
- Capitalise the initial keyword.
- Indent to show hierarchy.

- End multiline structures.
- Keep statements language independent.

If an author chooses to use flowcharts to describe an algorithm, the design process becomes much more specific than pseudocode, since there are rules that apply to flowchart design, especially for programming flowcharts. The ISO 5807:1987 standard describes the symbols and conventions for use with program flowcharts.

In general, good practice suggests the use of a pseudocode description followed by an equivalent program flow chart in order to provide a comprehensive visualised representation of an algorithm.

Within the data structures description, any data structure designed to organise and ensure data can be accessed and worked with, must be well defined.

Defining a data structure includes:

- A comprehensive description of the purposes of the data structure.
- A description of the properties of the data structure.
- A description of the related functions of the data structure.

All of these attributes could be organised in lists or tables, per software module.

Finally, the type, structure and the parameters of any file/message used in the software framework should also be detailed. An example of good practice on describing file/message formats would be to define the formats by each related interface in different subsections. It is preferable for the section describing file/message formats to be organised by interface, including a description of all the file/message formats, even if some of these are repeated.

Specific attention when describing file/message formats must be given to standards compliance. In order to conduct inter-platform communication between such systems, there is a need to agree on the types and contents of the file/message formats that are exchanged. This can be done by ensuring compatibility with existing or proposed standards, as a common reference point for implementation. This issue can be successfully addressed if the author uses current RFCs or other available standards (see [RFC]).

3.1.4 Examples

Algorithms, Structures and Formats

- perfSONAR LS developers have published a paper that describes in detail the main algorithms, structures and formats of the software [**perfSONAR**]

3.1.5 Tools for Producing Code-Related Documentation

Code-related documentation should be created with tools that allow easy creation, modification and maintenance of a document's structure and content. The following sections provide information about tools that are recommended for internal documentation.

3.1.5.1 *DocBook*

DocBook is an open source, semantic markup language tool for internal technical documentation that can easily be maintained, edited, converted and published. As a semantic language, DocBook enables its users to create document content in a presentation-neutral form that captures the logical structure of the content. The content can be published in a variety of formats, (e.g. HTML, XHTML, PDF) without any changes to the source. DocBook provides a vast number of semantic element tags which are divided into:

- **Structural** – specify broad characteristics of the contents (e.g. book, article, chapter, appendix). Structural elements can contain other structural elements.
- **Block-level** – not all of elements can contain actual text. Block-level tags are for example: paragraph, list, etc.
- **Inline-level** – wrap text within block-level elements. These elements do not cause the text to break when rendered in a paragraph format, but typically they cause the document processor to apply some kind of distinct typographical treatment to the enclosed text e.g. changing the font, size, etc. Inline-level tags are, for example: emphasis, hyperlinks, etc.

Rules defining the hierarchy of elements are formally defined in the DocBook schema, so correctness of the document can be easily verified. Since DocBook is pure XML, documents can be created and edited with any text editor. However, using a WYSIWYG (What You See Is What You Get) tool (e.g. XMLmind XML Editor) simplifies the documentation process.

DocBook files are used to generate the output in a wide number of formats (HTML, XSL-FO for later conversion into PDF), usually using DocBook XSL stylesheets. These stylesheets enable the generation of tables of contents, glossaries and indexes. Best practice reference relating to DocBook documentation creation may be found in Appendix A.

3.1.6 Javadoc, Doxygen, ROBODoc

Javadoc, Doxygen and ROBODoc are source code documentation tools that make it possible to create internal technical documentation from source code comments. These tools are very similar and automatically generate hierarchical, hyperlinked documents. Developers should provide annotations and comments (in a strictly defined format) before classes, methods and fields to be documented.

The resulting internal technical documentation can be rendered into several universal formats, e.g. XML, HTML and RTF.

ROBODoc example:

```
/****f* Robodoc/RB_Panic [2.0d]
  * SYNOPSIS
  */

  void RB_Panic (char* cause, char *add_info)

 /*
  * FUNCTION
  *   Prints an error message.
  *   Frees all resources used by robodoc.
  *   Terminates program.
  * INPUTS
  *   * cause    - pointer to a string which describes the
  *                cause of the error.
  *   * add_info - pointer to a string with additional information.
  * SEE ALSO
  *   RB_Close_The_Shop ()
  * SOURCE
  */
   {
     printf ("Robodoc: Error, %s\n",cause);
     printf ("         %s\n", add_info);
     printf ("Robodoc: Panic Fatal error, closing down..\n");
     RB_Close_The_Shop (); /* Free All Resources */
     exit(100);
   }

  /*******/
```

Javadoc example:

```
/**
 * Validates a chess move. Use {@link #doMove(int, int, int, int)} to move a
piece.
 *
 * @param theFromFile file from which a piece is being moved
 * @param theFromRank rank from which a piece is being moved
 * @param theToFile   file to which a piece is being moved
 * @param theToRank   rank to which a piece is being moved
 * @return            true if the chess move is valid, otherwise false
 */
boolean isValidMove(int theFromFile, int theFromRank, int theToFile, int
theToRank)
{
    ...
}

/**
 * Move a chess piece.
 *
 * @see java.math.RoundingMode
 */
boolean doMove(int theFromFile, int theFromRank, int theToFile, int theToRank)
{
    ...
}
```

Doxygen example:

```
/**
 * @file
 * @author  John Doe <jdoe@example.com>
 * @version 1.0
 *
 * @section LICENSE
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details at
 * http://www.gnu.org/copyleft/gpl.html
 *
 * @section DESCRIPTION
 *
 * The time class represents a moment of time.
 */

class Time {

    public:

       /**
        * Constructor that sets the time to a given value.
        *
        * @param timemillis Number of milliseconds
        *        passed since Jan 1, 1970.
```

```
         */
        Time (int timemillis) {
            // the code
        }

        /**
         * Get the current time.
         *
         * @return A time object set to the current time.
         */
        static Time now () {
            // the code
        }
};
```

3.1.6.1 *Natural Documents*

Natural documents (e.g. MS Word or Notepad format) should be used when providing semi-formal, internal requirements or documentation. Internal reports provided by developers may be related to case studies or proof of concept of some possible (or analysed) solutions. This type of code-related documentation can also be applied, when describing algorithms that have been implemented. In this case, not only the algorithm itself, but also preliminary assumptions, all known limitations and its computational complexity should be provided. A formal structure of internal reports can be defined by project managers, since it seems to be unnecessary to provide a unified structure for independent teams. It is a good practice to store the different versions of these files in a repository, so conceptions and applied changes can be tracked.

Example of internal report delivered in a natural document:

```
CNIS TOPOLOGY VISUALIZATION
===========================

Author:      Marek Lewandowski <marekl@man.poznan.pl>
Created:     2009/02/18
Last update: 2009/02/20


Introduction
-------------------
Visualization of the topology provides a cNIS operator with a general overview
of the network topology. Visualization, comparing to (...)

Goal
------------------
Simultaneous visualization of many network topology layers and relationships
between them.
```

```
Current state
------------------
Currently, topology can be visualized in two ways: using Google Map or using
generated graph.

Google Map visualization presents network devices (...)

Implementation
------------------
Google Map visualization has been implemented using Google Maps Api and
JavaScript. Geographical localization data is retrieved from a database. If a
node does not possess localization information, it will not be presented on a
visualization. The latter visualization has been implemented with Java (JUNG
framework) and Flash (...)

Interaction
----------------
There are several use cases possible:
- displaying whole IP topology (IP button pressed) - whole topology (...)

GRAPH PROCESSING
------------------
DAO methods (in VisualizationGraphDao) calculate the graph and return it in an
XML document (...)
```

- For further information, see [INTDOC].]

4 Public-Facing Documentation

Public-facing documentation is created by the GN3 Technical Authors and must comply with the standards set by the GN3 Project Office and be available through the GN3 Forge repository. Whenever a requirement for public-facing documentation arises within a GN3 Activity, the Activity Leader or Task Leader should contact the Technical Author assigned to their Activity or send an email detailing the request to the Project Office (gn3-po@geant.net), and brief them with the requirements.

4.1 Using Technical Authors

Pending deliverable workload in the PMO, software development teams can use Technical Authors for a number of tasks. For example:

- **Gathering and structuring information.**

  Activity or Task Leaders can ask a Technical Author to help structure the information needed to produce documentation. Usually, the Activity or Task Leader gives the Technical Author a list of contributors and indicates what information should be provided by whom. The Technical Author will then contact the content contributors with requests for information, track the progress of information gathering, and structure the information as it comes in, under the supervision of the Activity or Task Leader.
- **Content management.**

  Activity or Task Leaders can ask a Technical Author to manage content. This is usually needed for web, wiki or Knowledge Base content [**KnowledgeBase**]. The Technical Author identifies relevant themes and topics that are present in the content, and then develops a taxonomy, with the aim of enabling users to find content they are looking for quickly and easily. As new content for the taxonomy becomes available, the Technical Author can add it to the appropriate places in the taxonomy.
- **EC Deliverables.**

  Activity or Task Leaders can ask a Technical Author to help him/her develop a document. Once the scope is decided, the Technical Author can then be asked to gather and structure the information needed for the deliverable. When the deliverable is ready for the review process, the Technical Author sends it to the allocated reviewers (SME, Technical and QASPER) and coordinates any comments and change requests that are made.

- **Technical reference documents.**

  Activity or Task Leaders can ask a Technical Author to help him/her develop a document scope for technical reference documents (e.g. user guides, admin guides, installation guides, etc.). Once the scope is decided, the Technical Author can be asked to gather and structure the information needed for the document. When the document is ready for publication, the Technical Author can upload the master document and a PDF copy to the version control management system.

- **Website, intranet and wiki text.**

  Activity or Task Leaders can ask their Technical Author to help write, edit or upload content for the website, intranet or wiki.

- **PR material.**

  Activity or Task Leaders can ask their Technical Author to translate the achievements of software development teams into news items that PR can process. Alternatively, Activity or Task Leaders can submit newsworthy information to PR directly and ask their Technical Author to review the material PR produces.

- **Online help authoring.**

  Activity or Task Leaders can ask their Technical Author to assist with the creation of online help systems. The Technical Author can do this using any help authoring tool that the Activity provides for this, or by writing HTML and CSS in a text editor.

- **Technical diagrams.**

  Activity or Task Leaders can ask their Technical Author to create technical diagrams. The Technical Authors can also provide GN3 software development team members with a GN3 icon stencil.

- **Templates.**

  The Technical Authors provide a number of templates for GN3 documents (for example, a deliverable template, an admin guide template, etc.) but if a need for a new type of template arises, Activity or Task Leaders can ask their Technical Author to create the template and get it signed off by the Project Office.

- **Usability testing.**

  Developers can ask Technical Authors to test user-facing software from a user perspective and provide feedback (this is particularly useful for testing user interfaces).

- **Release notes.**

  Activity or Task Leaders can ask their Technical Author to edit release notes before publication.

4.2 Creating Public-Facing Documentation

The first phase of the documentation creation process (see Figure 4.1) is the same for both EC Deliverables and non-deliverable documents. It involves identifying content contributors, gathering and structuring content, and creating a first draft.

The following steps outline the process of creating a document:

1. **Identify a requirement.**

   If you think that new documentation is needed, you should provide your Activity or Task Leader with the following details:

   What kind of document is needed?

   ○ For example, an Installation and Configuration Guide.

   Why is it needed?

   ○ For example, because users would not be able to install and configure a specific application without instructions.

   Who is it for?

   ○ Who is the target audience? For example, a specific group of users.

   What should it achieve?

   ○ For example, it should provide users with step-by-step instruction of how to install the application on specific operating systems, and provide detailed explanations of all configurable parameters, including default settings and examples.

2. **Submit the requirement to your Activity or Task Leader.**

   The Activity or Task Leader will review the requirements and decide if it is justified.

3. **Contact a Technical Author.**

   Check which Technical Author is allocated to your Activity, and contact them with the requirement details the Activity or Task Leader signed off. If required, the Technical Author can help to review the scope of a document.

4. **Identify content providers.**

   Once the document scope is agreed, let the Technical Author know who will provide information for the different topics or sections.

The second phase of the documentation creation process (see Figure 4.2) applies only to EC Deliverables. In this phase, the Technical Author submits the deliverable to the Technical and the Policy Reviewers, and manages the resulting revision process with the document owner.

The third phase of the documentation creation process (see Figure 4.3) applies only to EC Deliverables. In this phase, the deliverable is submitted to the QASPER Reviewers. The Technical Author manages any change requests and finalises the deliverable. It is then published by the Project Office.

See the "Writing Process for GN3" document for further details [GN3-09-040].

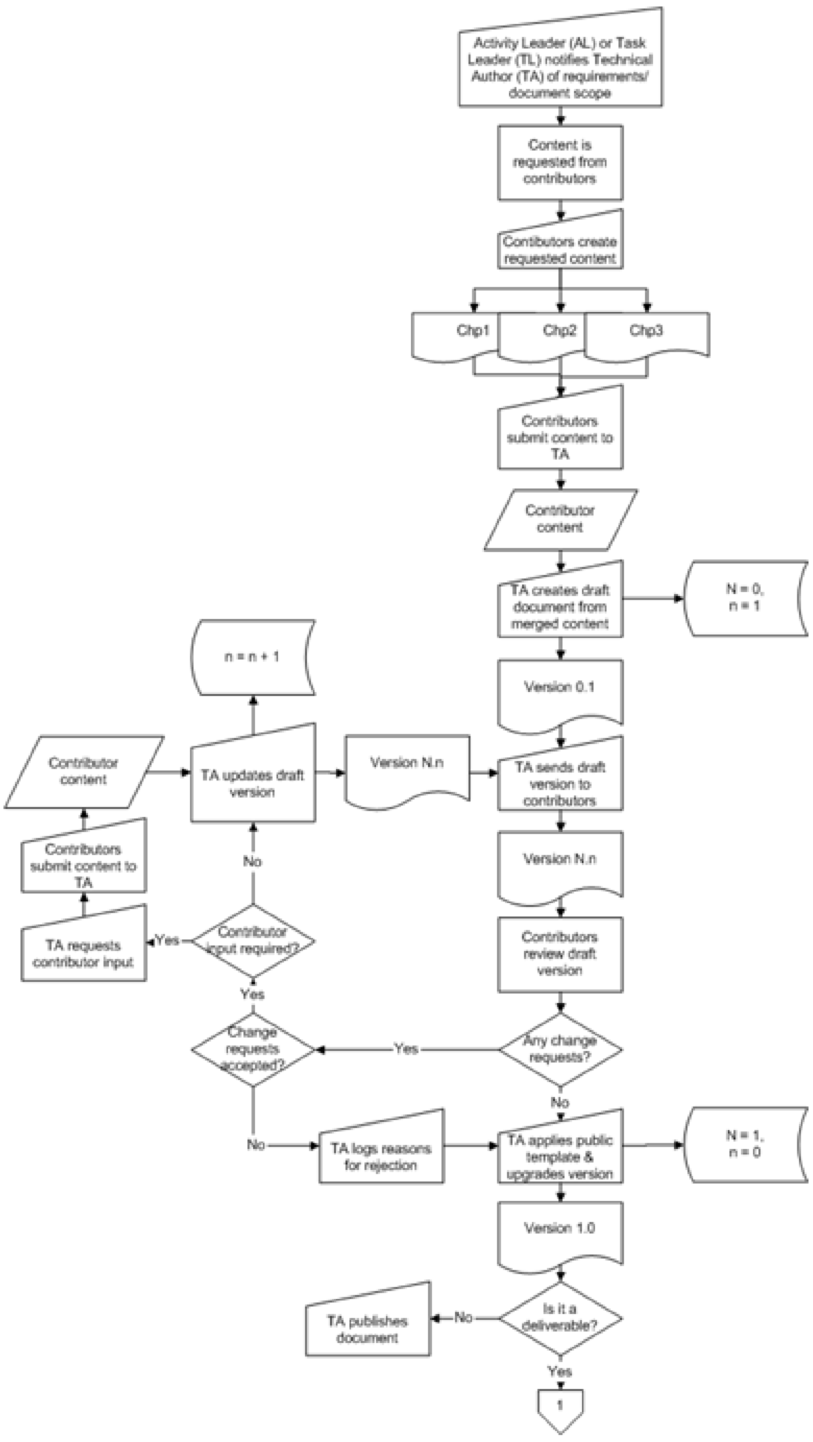

Activity Leader (AL) or Task Leader (TL) notifies Technical Author (TA) of requirements/ document scope
Content is requested from contributors
Contibutors create requested content
Chp1
Chp2
Chp3
Contributors submit content to TA
Contributor content
TA creates draft document from merged content
N = 0, n = 1
Version 0.1
n = n + 1
Contributor content
TA updates draft version
Version N.n
TA sends draft version to contributors
Version N.n
Contributors submit content to TA
No
TA requests contributor input
Yes
Contributor input required?
Contributors review draft version
Yes
Change requests accepted?
Yes
Any change requests?
No
No
TA logs reasons for rejection
TA applies public template & upgrades version
N = 1, n = 0
Version 1.0
TA publishes document
No
Is it a deliverable?
Yes
1

Figure 4.1: Documentation Creation Process – Phase 1.

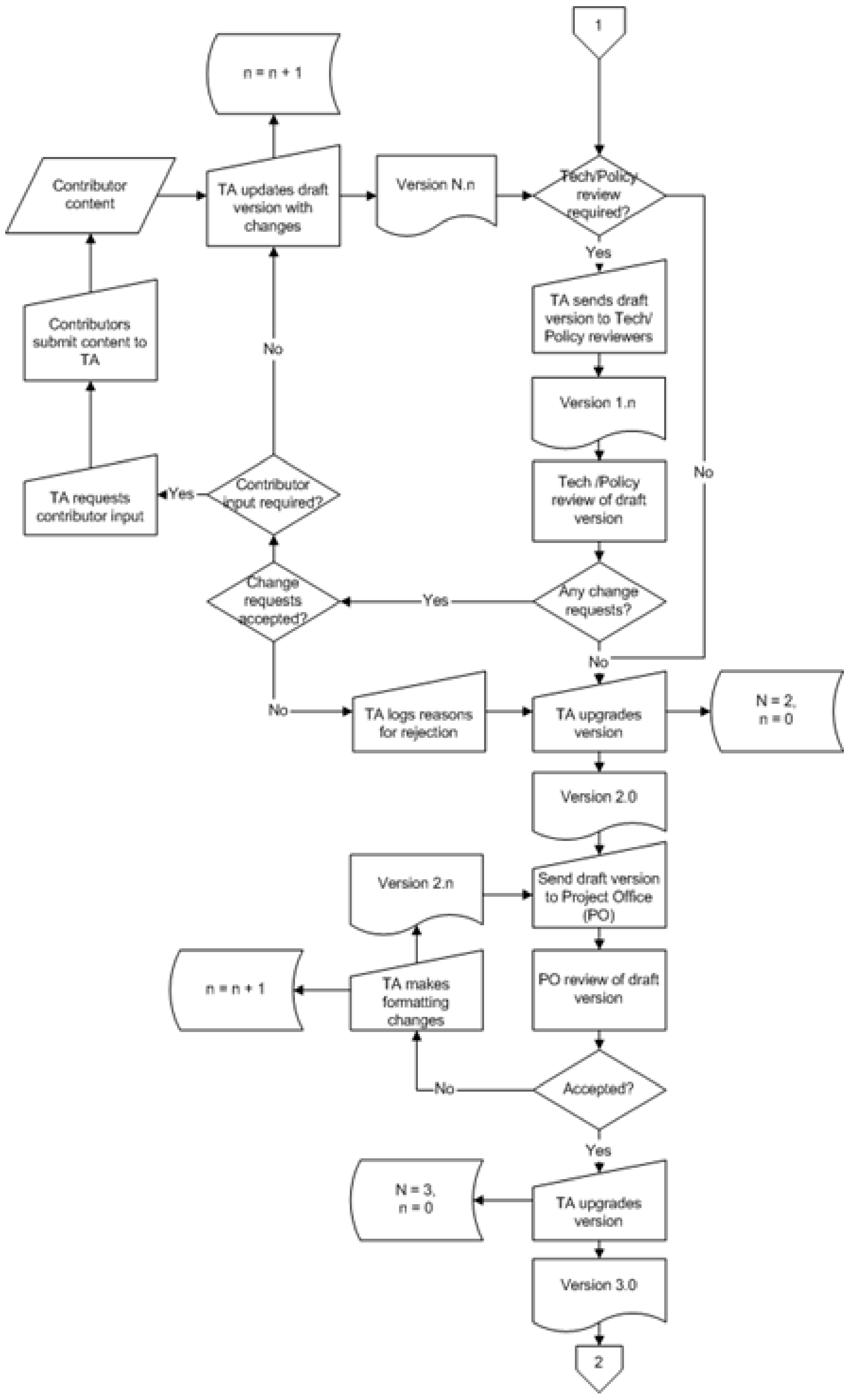

1
n = n + 1
Contributor content
TA updates draft version with changes
Version N.n
Tech/Policy review required?
Yes
No
TA sends draft version to Tech/ Policy reviewers
Contributors submit content to TA
No
Version 1.n
TA requests contributor input
Yes
Contributor input required?
Tech /Policy review of draft version
Change requests accepted?
Yes
Any change requests?
No
No
TA logs reasons for rejection
TA upgrades version
N = 2, n = 0
Version 2.0
Version 2.n
Send draft version to Project Office (PO)
n = n + 1
TA makes formatting changes
PO review of draft version
No
Accepted?
Yes
N = 3, n = 0
TA upgrades version
Version 3.0
2

Figure 4.2: Documentation Creation Process – Phase 2 (Deliverables Only).

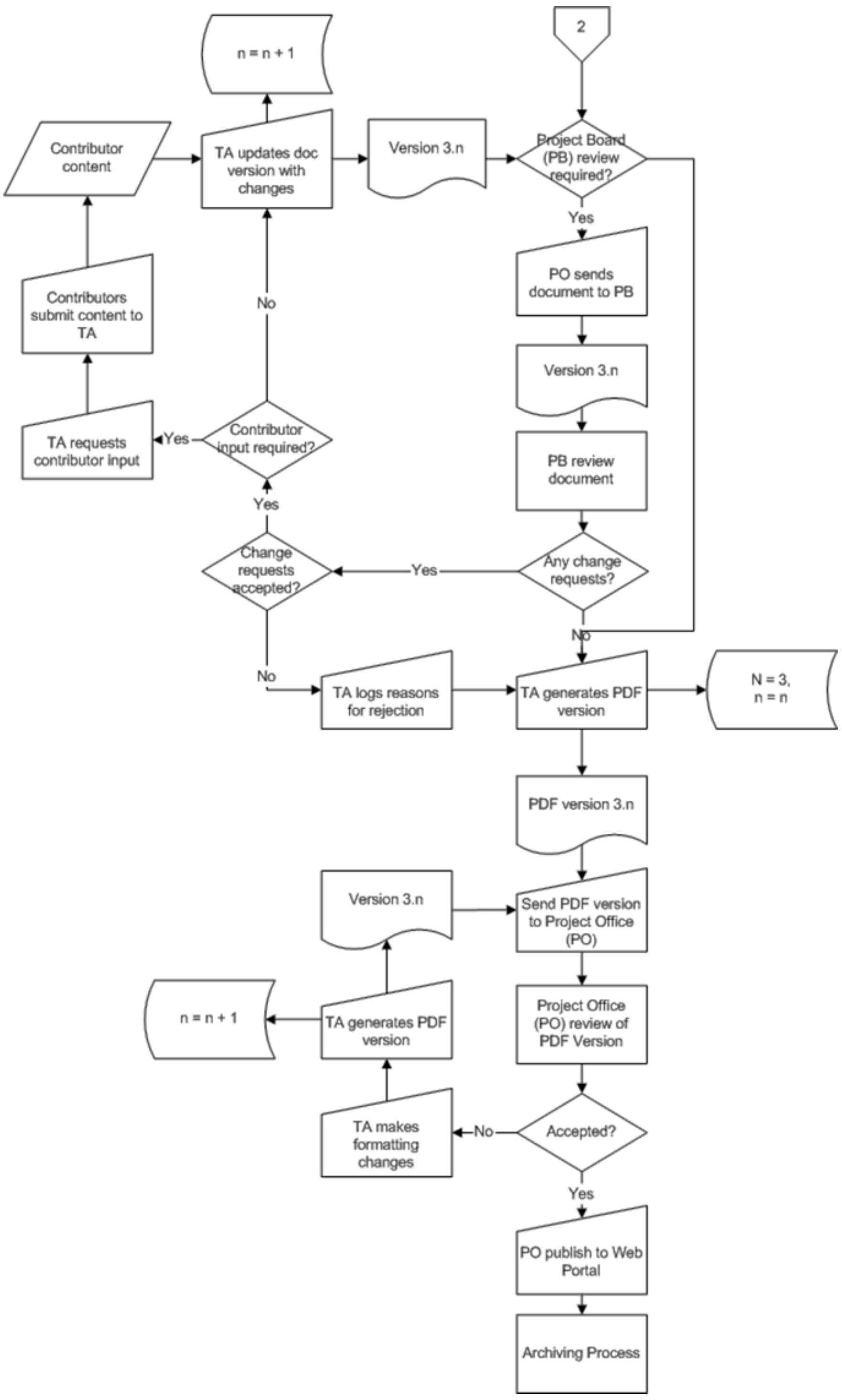

2
n = n + 1
Contributor content
TA updates doc version with changes
Version 3.n
Project Board (PB) review required?
Yes
PO sends document to PB
No
Contributors submit content to TA
Version 3.n
PB review document
TA requests contributor input
Yes
Contributor input required?
Yes
Change requests accepted?
Yes
Any change requests?
No
No
TA logs reasons for rejection
TA generates PDF version
N = 3, n = n
PDF version 3.n
Version 3.n
Send PDF version to Project Office (PO)
n = n + 1
TA generates PDF version
Project Office (PO) review of PDF Version
TA makes formatting changes
No
Accepted?
Yes
PO publish to Web Portal
Archiving Process

Figure 4.3: Documentation Creation Process – Phase 3 (Deliverables Only).

4.3 Changing Public-Facing Documentation

The process for changing public-facing documentation (see Figure 4.4) applies only to non-deliverable documents. (Once published, EC Deliverables are not changed). This process applies where:

- A bug has been identified in documentation and needs to be fixed.
- New content needs to be added to documentation (for example, if a new feature has been added to an application and needs to be documented).

A documentation change request may be issued by completing the following steps:

1. **Submit your change request to the Technical Author.**

   Note: You can submit change requests to the Technical Author via email, however, where the change requests applies to software for which a bug tracking tool is in place, it should always be assigned to the Technical Author via the bug tracking tool.

   Check which Technical Author is allocated to your Activity, and contact them with the following details.

   For bugs:

   Where is it?

   Specify exactly where the bug is located. For example, if it occurs in a manual, provide the manual's title and version number, the number and title of the section in which the bug occurs, and the paragraph number (you can also copy and paste the faulty text into your request).

   What is the issue?

   Explain what is wrong.

   How can it be fixed?

   If you know how the bug can be fixed, you should include this information (obviously, this is not necessary for typos). If you do not know how it can be fixed but you know who could provide information that is relevant for fixing it, you should include the name and email address of this person.

   For new content or amendments to existing text:

   What is it?

   Explain what the new content is about. For example, if a feature has been added to an application, explain what this feature does and how it is used.

   Where should it go?

   Suggest where the new content should be added. For example, if it should be added to a manual, provide the manual's title and suggest a suitable place for the addition (provide the section numbers and titles).

   Is more detail needed?

   Consult a subject-matter expert that might be able to provide more information.
2. **Review the change**

Once any changes have been carried out, the technical author will send the document owner a request to review it. If the change request is handled via a bug tracking system, the system will email you automatically once the Technical Author has marked the request as fixed. At this stage, you should review the change and do one of the following:

- If you are happy with the change, mark it as resolved and close it.
- If you are not happy with the change, reopen the bug and reassign it to the Technical Author.

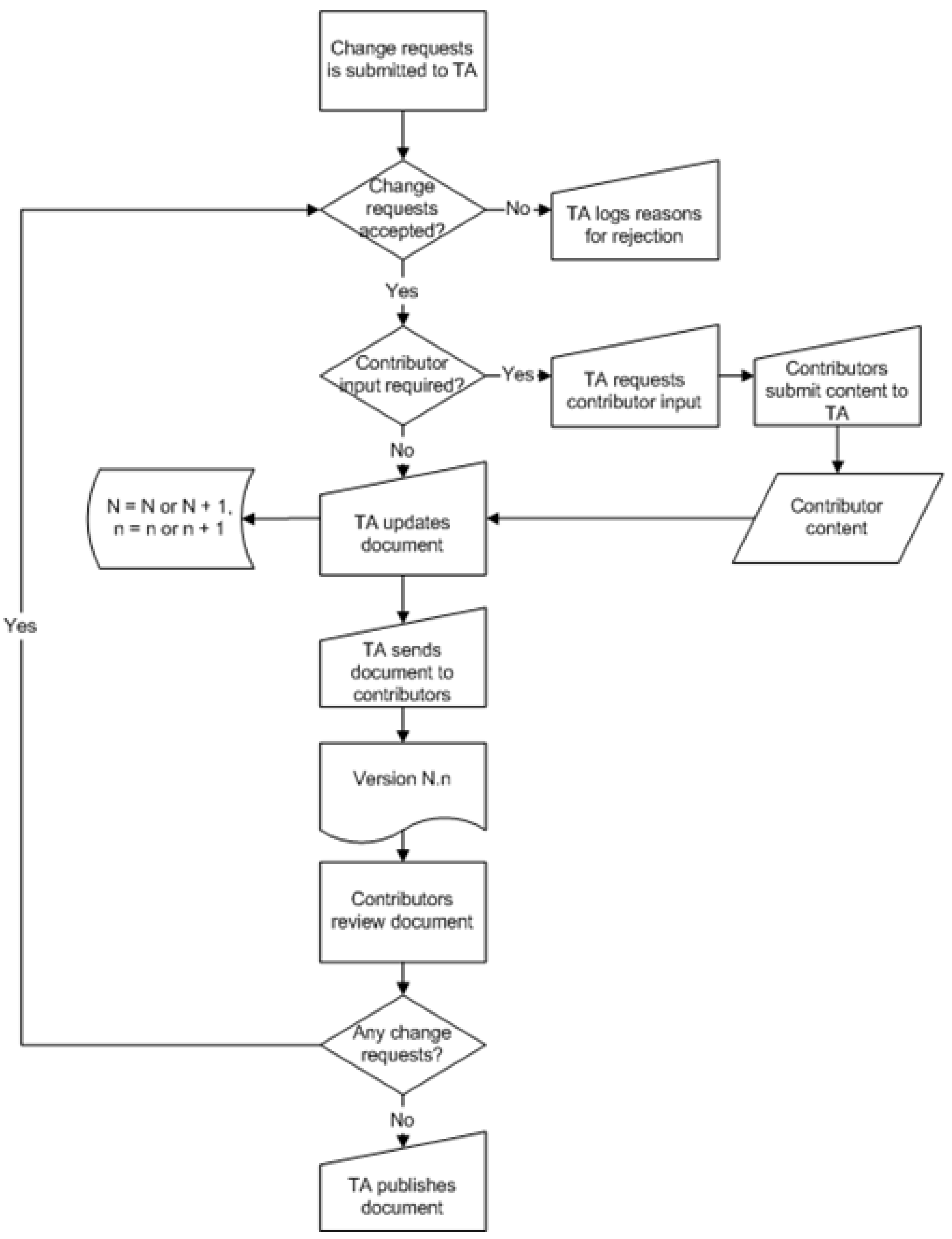

Change requests is submitted to TA
Change requests accepted?
No
TA logs reasons for rejection
Yes
Contributor input required?
Yes
TA requests contributor input
Contributors submit content to TA
No
N = N or N + 1, n = n or n + 1
TA updates document
Contributor content
Yes
TA sends document to contributors
Version N.n
Contributors review document
Any change requests?
No
TA publishes document

Figure 4.4: Documentation Change Process

5 Public-Facing Documentation Structure

Examples of existing documents of individual GN3 software developments have been collected through each project's QA, and may be found in the Document Evaluation summary spreadsheet (see [DocEval]). These examples can be used to build middle-line recommendations suitable for most projects, which are also relatively well-aligned with the existing documents. The existing examples should be also used to recognise practices that should be avoided or modified.

Templates for DocBook, Javadoc, Doxygen and ROBODoc may also be helpful. If used, these should be mentioned, referenced, and perhaps even refined to form part of the documentation. Some, primarily internal templates (but also Release Notes, Installation / Quick Start Guide, User FAQ, User Guide) are available [**ReadySET**]. Also, bear in mind the existing GN3 software documents, as mentioned in [**DocEval**].

Note that the purpose of this guide is to cover documentation that has the widest appeal/application to projects and is most useful to new joiners. As a result, details of code-related documentation (such as project planning, requirements, early specification, internal management, methodology and similar) have been kept to a minimum.

All public-facing documents should be stored in GN3 Forge Main Menu -> Documentation section, which is publically available.

Keep the distributed and web versions equivalent in content, using HTML as the preferred format of the web version. Generate distribution and web versions of documentation from the same source (e.g. in DocBook XML format) using documentation generators, such as Maven. If some pieces of text are needed in several places (such as the User Guide, which typically contains all information that is available in README and other places), please try to capture these relationships within each item's description.[1] Since several people may be describing different documentation artifacts, it is important to consistently cross-reference them within descriptions, e.g. **to establish relationships between similar/related sections of various documents**. Therefore, all authors involved will need to have at least two iterations of their initial documentation descriptions, and include cross-references to corresponding sections.

If some excerpts are needed, e.g. in the Software overview, a brief segment of bespoke text should be provided there instead of copying and cropping from a longer version. Avoid redundancy, but providing some overview (using separate documents to provide further details, if required), is also an option. It is not necessary to overload the overview text with refs to the detailed body if it can be easily located on a documentation home

[1] e.g. item X from README should have a detailed description in the user guide, while the content of the item Y can be directly copied.

page or README. Similarly, do not overload the overview with jargon or technical details that are elaborated elsewhere. If multiple-author text is merged within documentation of larger software, it should be divided into subsections for each participating component – e.g. by providing corresponding "RRD MA" subsection in the perfSONAR overview document, or in the configuration section of the installation guide. Direct links should also be provided to the latest version of artifacts published on the web. These links should be immutable, and point to the same content as the links (to the explicitly named latest version of the software among those that are published). Older versions can be made available with corresponding release descriptions (if shown on the web), and within release distributions (where included).

File names can end with an extension associated with specific file format. Whether ".txt" or ".TXT" extensions are used is left to the project's discretion, but this should be applied consistently throughout the project. This may also be preferred if the primary execution platform is MS Windows.

Paths within a software repository are relative to the corresponding a tag/branch/trunk.

The version control system used for source code is the natural place for keeping documentation artifacts that are packaged and distributed with applications. It can be used for other documents, particularly if they are in a mergeable text format (such as less-structured formats such as .csv or .txt, but not a text with heavy markup, such as PostScript), and if an alternative solution (internal portal, shared filesystem) does not provide support for versioning of documents.

5.1 General Informative Materials

A typical step before starting the production of a guide is called **task analysis**. Different types of users need to carry out different tasks. A task is a set of operations that is used to achieve a goal. A procedure is an ordered list of instructions that tells one person how to do a low-level task. A hierarchy of tasks exists, of which procedures form the lowest level. Useful actions are listed below to find the tasks and procedures that users should follow:

- Observe users and talk to them about their experience.
- Get information from existing documentation and from functional specifications.
- Match the tasks to known practice. For example, if one task is to create a database record, other tasks are necessary to select, change, and delete (or archive) database records.

The next step is to choose the appropriate **structure and content**. When creating a document, one or more of the following actions are suggested:

- Divide the document into sections based on user roles.
- Match the procedures to tasks. Group similar tasks into the same section.
- Organise sections so that frequent tasks come before infrequent tasks.

If both task-based instructions and reference material are needed, it is best to divide the document into two sections: guide and reference manual. The guide contains short procedural information. It does not explain each field in each dialog box. Instead, it contains cross-references to the reference section.

Finally, it is important to highlight that an administrator's documentation is likely to have some overlap with user documentation, since it covers similar configuration tasks in many parts. Therefore, it is important to pay attention to the contents of each document and try to keep any content overlap to a minimum.

5.1.1 Overview

Location: The overview should be placed at the beginning of the README and/or User Guide and also the product's web page (either at the home page or a dedicated overview page). In the case of GN3 Forge, it should be Main Menu -> Home.

One of the most important parts of documentation that is often neglected is an overview description of the software product. Its main objective is to introduce the reader to the purpose, key features, context, and the scope of the product. This overview provides crucial information to non-insiders, such as potential, but uninformed users, or casual or accidental visitors. Therefore, it must be clearly written using the simplest possible language in order to allow a moderately informed person to decide whether s/he should invest more in the study of the product, site, or the rest of the document. At the same time, the overview should communicate the key message to 'enlightened' (but non-expert) readers with minimal use of jargon used in the target domain of the product. Writing an overview can be difficult, as it is important to avoid jargon as much as possible. However, if some feature is strictly related to a specific business domain idiom, standard or regulatory requirement, there is no point in trying to avoid using the term or acronym.

All this information should be communicated within the first introductory paragraph, while additional assumptions, constrains, or boundaries of applicability may be elaborated in few additional paragraphs, if space permits.

A typical overview has three parts:

- The introduction.
- The hook for the rest of the document, or if standalone, a link to a more detailed document.
- A succinct description of the content (if the overview is actually an introduction to detailed document).

The overview in the README may be shorter than the one found in the User Guide and on the website. The overview at the site may be limited to a couple of sentences and a few key bullets leading to a separate, more detailed overview page that may provide a more detailed list of features.

When the overview is used to describe a larger text or document that follows, its introduction should describe the basic purpose and audience of the document and then summarise the topics covered. A good overview should describe the whole document in a few sentences. It is important to avoid excessive detail, since this is an overview and only the main idea should be included.

Writing a summary is not a trivial task. The big picture should be presented in a way that helps create a context for the document through the use of simple sentences.

Examples

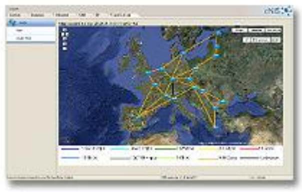
- The README file of the I-SHARe tool provides an example of a well-written overview of the software. [**I-SHARe**]

5.1.2 Screenshots

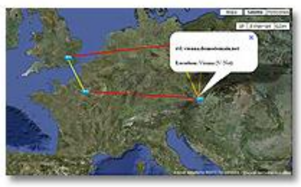
Location: Screenshots should be presented on product's web page (In case of GN3 Forge, it should be Main Menu -> Screenshots) and accompanied by short but meaningful description (e.g. "administration panel", "configuration form", etc.).

Screenshots should provide a general visual overview of the software, and reflect its functionality and application. Pictures should be carefully selected and attractively presented, so that the potential user is encouraged to download and use the application and is not overwhelmed by the materials. If extensive description cannot be avoided, the author should consider presenting pictures with the textual description, so that the reader can easily match the explanation with the graphics.

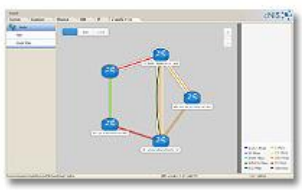
Screenshots displayed on the product page cannot be treated as part of the installation guide, administration guide, user guide or manual. Their main purpose of these pictures is to invite the user to install and use the application and convenience him or her about attractiveness of the product.



GÉANT network visualisation: Different details of topology data can be displayed by geographic location. Typical link speeds are distinguished by different colours.



Node details on google maps visualisation: Node details and link information can be displayed directly below the map.



Flash visualisation: Network structure can be displayed on an interactive diagram, highlighting interconnections between network layers.

IP discovery control pane: On-demand network discovery can be started on the fly from a control panel. It is also possible to schedule the discovery for specified periods of time.

Ethernet discovery results: The user can view and revise topology discovery results before saving them to the database.

Router list: Topology data can be edited and explored through a set of tables and panels. An intuitive structure helps finding appropriate data.

Figure 5.1: Screenshots with explanation of the presented functionality for cNIS (See [**cNIS**] for further indicative examples.)

Examples

- http://www.geant.net/service/cnis/About_cNIS/Using_cNIS/Pages/Using_cNIS.aspx
- https://forge.geant.net/forge/display/ishare/Screenshots

5.1.3 README

Location: README file should be available in GN3 Forge Main Menu -> Documentation section.

The README file, the name of which is traditionally written in upper case in order to be seen more easily, provides the basic information about the packaged software, library or archived set of files. It is a starting point for many users, and as such, its content should be simple and short. A well-written README can save time in understanding the scope and purpose of the package, its usage and key assumptions and limitations. It also serves as a catalogue for accessing additional information.

5.1.3.1 *Document structure*

Below is the recommended structure of a README file (see [**README**]). However, if any part of this document tends to become too long (out of necessity), it should be extracted to a suitable separate document (file, web page or the printed manual), and only a short summary with the key information and pointer to the extended version of the section should remain in README. As a guide, the length of the README should be between 3000 and 10,000 characters.

- The very beginning of README file should state the product name and its version code or number. This information may be embedded into a one-sentence description of the software's purpose, enabling the reader to unequivocally distinguish the products from anything else, and provide a "hook" for further reading.

- It is also useful to include the copyright statement and release date of the README file at the very beginning of README.

- The information about the producer, owner, or maintainer of the package should follow the copyright statement and release date. This may simply refer to the umbrella project or its website or, provide the complete contact information that includes:
  - Organisation or company name.
  - Product or organisation site.
  - The email address of a contact for administrative, commercial or distribution aspects of the product.
  - Email, phone, or immediate messaging contacts for technical support, questions, and bug reporting.

- Information about the possible costs or commercial conditions associated with the usage of the product. Since these conditions are prone to change and are often negotiable, this paragraph can point to a more detailed file or web address with more detailed options, prices or additional contacts.

- Licensing info, with pointers to corresponding licence files.
- Optional credits to the original creator of the product, but in a way which does not leave space for confusion regarding current owner and contacts. This information may also establish a stronger context or reference for the product, or act as an additional teaser for further reading about the purpose and features. This may also be a suitable place to point at the AUTHORS file (see Section 5.1.4, Authors and Maintainers).
- List of files included with distribution or online resources that provide additional details or deserve additional reading, if not included/referenced from sections of the README.
- Functional overview describing the purpose of the product or package in about one paragraph that may be accompanied by the list of its key features in short and easy to understand bullet points (see Section 5.1.1 Overview).
- The Hardware and Software Requirements sections provide an overview of requirements (see Section 5.3.1 and 5.3.2).
- Minimal installation and usage instructions, getting started tips, and pointers to more detailed installation, configuration and usage guides. An explanation of the purpose of each of the first-level subdirectories in the package, if any, may be provided.
- Important, known problems and brief instructions for reporting problems or getting support, with instructions on how to submit bugs, feature requests, patches, get announcements (see Section 5.2.2 Bugs and Support), or join a mailing list and user or development community.
- Short release notes describing the news and specifics of the current release. The information provided should be sufficient to allow the user to decide whether to upgrade the already-installed software or not. Everything beyond that point is reserved for separate Release Notes (see Section 5.1.7) and Change Log/Version History (see Section 5.1.9), which also needs to be referenced from this section.
- Optional legal notices including disclaimers, privacy statements, terms of use, etc. If this section is present, copyright and licence details, particularly in relation to used external components may be placed here. However, details related to licences are often stored in separate Licences and NOTICE files (See Section 5.1.5 Licences).

The README file should be self-contained, and include all the parts detailed above (that should not be replaced with links). The preferred format is plain text, with up to 80 characters per line, empty lines between paragraphs, dashes under the section titles, and without tab characters. The use of HTML is justified only if the document is heavily dependent on information provided through links to extended information.

Optional translations of README to other languages should be suffixed with a two letter ISO 639-1 language code (or an additional ISO 3166-1 country code, which is useful if there are some tailored contact points or links to custom contents for specific locales). It is better to use the ISO codes and avoid the translation of README file name into names like LEEME, LISEZMOI, CZYTAJTO, as it makes it more difficult for users to see in which languages README is available and to understand the purpose of files with unfamiliar names.

Examples

- http://openwebmail.org/openwebmail/doc/readme.txt
- http://gd.tuwien.ac.at/infosys/mail/pine/README
- http://cnmdev.lrz-muenchen.de/e2e/lhc/dist/read_me.txt
- https://intranet.geant.net/sites/Services/SA2/T5/AutoBAHN/Pages/AutoBAHNUserGuideWorkArea.aspx

5.1.4 Authors and Maintainers

Location: The AUTHORS file should be included in the documentation folder of distribution package, Maven's pom.xml (which may be used to extract information from different places, such as a website). This should be available in GN3 Forge Main Menu -> Documentation section.

The purpose of naming the authors of the project is to credit the work. Is should be independent from licensing info, overview, or README.

The structure is rather simple. For each author, her/his name, surname, and possibly home institution (especially if the institution is officially involved in the project) and country should be stated. The document may include a brief description of personal roles or period of involvement. It is generally a good idea not to leave the author's contact information, in order to keep the communication about the project using official channels (e.g. mailing lists, instead of private emails).

If there are significant differences in the level of contributions, author lists may be split into "developers" (higher level of contribution) and "contributors" (lower level of contribution). The list of authors may also be sorted according to their involvement.

Information about authors and maintainers, and other credits may be skipped only if it is necessary to preserve an authors' anonymity (imposed by an engagement contract or for security-related reasons). When authors cannot be named, e.g. in case of some legacy code, institutions or originating software should be mentioned instead. However, authors of the separate, third-party software components (e.g. libraries) should be mentioned elsewhere, e.g. in the attached NOTICE file, with licensing information.

Since this document should provide an accurate snapshot for each release, providing a link to a live list of project participants, instead of the above, is not sufficient.

Examples

- https://intranet.geant.net/sites/Services/SA2/T3/Pages/T3ServiceImplementationArea.aspx

5.1.5 Licences

Location: The software LICENCE (COPYING if a GPL is used), should be placed at the top of the binary and source distribution archives and source repository. It should also be available in Main Menu -> Licence section.

The Licence file is important in order for both users and automated scripts to understand licensing consequences of using software. These are related to business-friendliness, hidden costs, redistribution, re-use, modification, and other aspects that licences typically cover.

If standard licences such as GPL or ASL are being used, ensure that the latest version of the licence is used, and carefully read the instructions what should be changed in a licence (e.g. company and year-range of information). If there is a copyright notice as part of the licence, note that year span should not go into the future. The ending year is the last year of development, not the year that the copyright ends.

The Licence file should contain only the licence of the main software. If third-party software or separate software components are used, their licensing should be placed in the separate NOTICE file, together with the corresponding information about their authors.

Check each licence's rules about writing licensing info into the distributed files (particularly regarding placement of this information in source files). Maintaining them could be tedious (a bit easier with automated scripts), so do not add them if not necessary. This is usually only required for legal reasons.

Also, do not forget to mention and reference the licence, as well as all used components and their licences, in the software GUI, documentation, distribution, source code, or on the web, as required by the corresponding licences.

Examples

- See Appendices of GÉANT Intellectual Property Policy Document from http://www.geant.net/About_GEANT/Intellectual_Property/Pages/home.aspx
- http://openjpa.apache.org/license.html
- http://netbeans.org/about/legal/product-licences.html
- https://intranet.geant.net/sites/Services/SA2/T5/AMPS/Pages/ExternalLibrariesListandLicencesUsedinAMPS.aspx.

5.1.6 News

Location: The News section should be included as part of the GN3 Forge Main Menu -> Home page.

The News section is intended for users, so its content should not be overwhelming with too much technical detail or excessive use of jargon. News should include or provide information to the user about:

- Simple description of current and upcoming releases.
- New, important software functionality.
- Current project activities.
- Any observed and unresolved problems that a user may encounter.
- Forthcoming events like promotions, trainings, etc.
- Service unavailability, due to planned maintenance.

- Service failure, if it is unavailable, when it should operate.

In order to improve the flow of news to users, the project participants should establish and maintain a mailing list that could be joined by users.

Examples

- https://www.edupki.org/news/
- http://tomcat.apache.org/ home page.

5.1.7 Release Notes

Location: Release Notes should be made available in GN3 Forge Project Life Cycle -> Change Log section.

It is good practice for each release to have some kind of the announcement, either on the news section of the website, on the official mailing list, or both. It is also good practice for each release to have a complete reference list of changes (see Section 5.1.9 Change Log). However, news should be short (able to be read in about one minute), and a ChangeLog is tiresome to read, so for major (and sometimes minor) releases there should be something to fill the gap: Release Notes.

Release Notes should be easy to find and read, informative enough to persuade the user to upgrade, and noticeably different from the ChangeLog. The notes should be limited to the changes since the previous major version, but should also cover the important changes and fixes since the last major version and the current minor release. The release notes should list:

- Important or attractive new features, in a compact and user oriented form.
- Changes in existing features affecting the usage, appearance or performance of the system – other internal changes in implementation should not be mentioned here.
- Key migration issues from the previous major version, pointing to the installation/migration guide if any details are needed.
- Changes in software and hardware dependencies.
- Known issues.

The primary source for Release Notes is the change log. Release Notes should point at installation instructions, ChangeLog, and, if some issues are reported, Bugs and Support. The major changes described in Release Notes may deserve separate News items, or even changes in overview description.

Examples

- http://www.mozilla.org/en-US/firefox/11.0/releasenotes/

5.1.8 Release Plan / Roadmap

Location: GN3 Forge Project Life Cycle -> Release Plan section.

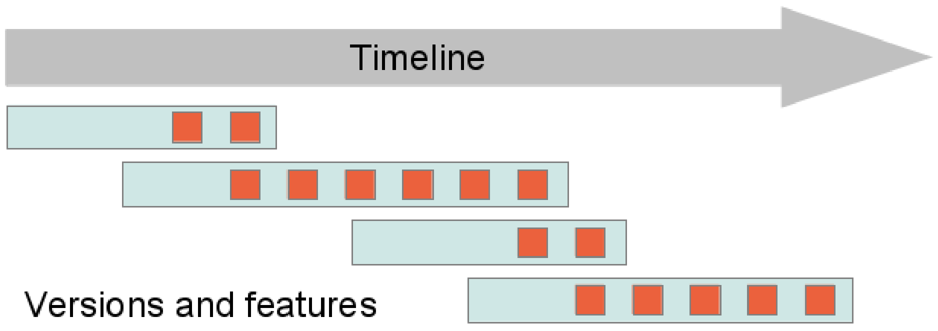
With the variety of rapidly evolving applications and technologies, it is generally difficult for potential users to choose the product upon which they will base their business strategy. Roadmaps are one of the factors users take into consideration when making decisions. Therefore, it is recommended that once product has defined a roadmap that matches the software required to reach a short- or long-term goal, it should be placed on the public website of the project. Even if it is not a properly declared roadmap, but rather just a vague vision or rough plan, it is beneficial to have it published.

Strategic planning is concerned with long-term goals, such as maximising business value and satisfying stakeholders, offering feasibility in terms of resource capacities, and reflecting dependencies between features. Operational planning is concerned with each subsequent release and allocation of resources to feature development within release.

The roadmapping process can be divided into three phases:

- The preliminary phase, in which essential conditions, leadership and scope should be defined.
- The development phase, in which product, requirements, technology and alternatives are defined.
- The follow-up, iterative phase, in which roadmap is constantly validated, reviewed and updated.

The software roadmapping process typically results in a layered-bar diagram such as the one depicted in Figure 5.2.



.

Figure 5.2: Typical representation of the project roadmap

.An alternative to visual representation could be a simple list of planned releases, with dates and a brief description of features. JIRA provides a roadmap gadget for each project, along with issues scheduled for the next ten releases. There is also an overview of progress made towards the release of a new version.

Examples

- https://forge.geant.net/forge/display/ishare/Release+Plan
- https://forge.geant.net/forge/display/autobahn/Release+Plan
- http://trac.edgewall.org/roadmap

5.1.9 ChangeLog/Version History

Location: GN3 Forge Project Life Cycle -> Change Log section.

This reference material provides a complete list of released software versions, and for each version a list of enhancements, changes, bug fixes and so on, is represented as a bulleted list. The newest release should be placed at the top. If the list is too long, it could be divided into software components.

The ChangeLog is often used by advanced users when considering an upgrade, and it is essential help to the users when upgrade fails, to find the source of the failure. Therefore, it should be linked from the release announcement email or news item. It should also be referenced from README and release notes, for which the ChangeLog forms the base.

Listed changes should reference the corresponding issues, if they are publicly available. The important sources for change descriptions are descriptions of resolved issues, commit comments, and build comments. Most issue trackers, including JIRA, allow easy creation of release notes in both plain text and HTML. However, this document should be written with the advanced user in mind and thus should intentionally reduce the amount of developer-related details present in issue trackers.

In order to put the ChangeLog in perspective with related artifacts, they are listed here in decreasing order in terms of level of details:

- Commit comments.
- Build comments.
- ChangeLog.
- Release Notes.
- News.
- README release notes.
- Overview.

However, the changes described in the ChangeLog may also affect the website, User Guide, and administrator documentation.

Examples

- https://forge.geant.net/forge/display/autobahn/Change+Log
- https://forge.geant.net/forge/display/cNIS/Change+Log
- https://forge.geant.net/forge/display/ishare/Change+Log

- https://forge.geant.net/forge/display/perfsonar/Change+Log
- http://cnmdev.lrz-muenchen.de/e2e/lhc/dist/release_notes.txt
- http://tomcat.apache.org/tomcat-6.0-doc/changelog.html
- http://www.mozilla.org/en-US/firefox/11.0/releasenotes/buglist.html

5.2 End User Documentation

5.2.1 User Guide[2]

Location: GN3 Forge Main Menu -> Documentation section.

The User Guide is a technical document intended to assist users in operating given software. It often provides a user-based overview of the software and its usage through the use of screenshots as illustrations of software's user interface, features, usage scenarios, or the way the information is presented. It is advised to address one type of audience in a given user guide at a time (in some instances, providing different user guides for IT managers, project managers, application administrators, users, developers, etc.). All user guides should contain:

- Preface

  The Preface is the part containing the information on how to navigate through the document. It often provides basic information about the software and describes related documents. If the document is complex, it may be advisable to add a "How to use this document" section.
- Table of Contents

  The Table of Contents provides a clear map of the location of the information provided in the document.
- Prerequisites section

  Prerequisites inform the Guide's readers about software and hardware which user must fulfil for proper use of given software and guide. In most cases, this section contains descriptions of:
  - Server/target machine hardware specifications.
  - Additional software which must be installed on target machine with detailed instructions on how to do this.
  - Initial configuration.
- Instead of detailed description of all necessary prerequisites, this section may provide references to the Hardware requirements specification and Software requirements specification documents.
- Typical Use Cases section

  The Typical Use Cases section is a guide on how to use the core functions of the system.
- Troubleshooting section

[2] Note this section have been based on http://www.klariti.com/technical-writing/User-Guides-Tutorial.shtml.

The Troubleshooting section provides information on how to deal with common issues while installing or operating the software. It provides information about known errors and workarounds.

- FAQ (Frequently Asked Questions) section

  The FAQ section provides answers for the questions most frequently asked by users.

- Contact Details

  Contact Details should provide comprehensive information about points of contact regarding technical support, feature requests, error reports, etc.

- Glossary

  The Glossary is the section describing any acronyms or uncommon terms used in the documentation.

The following steps should be taken in order to create a user guide:

1. Define the audience of the document.
2. Define subjects that will be covered in the document.
3. Create front page, cover pages, table of contents and (optional) copyright of the document.
4. Provide content of the document.
5. Publish.

After defining the audience of the document it is advised to define audience's needs. It requires answering the following questions:

- Where is this document supposed to be used (e.g. in the office, in the server room, at home)?
- What kind of experience is needed?
- Should readers already be familiar with the application (e.g. is this a User Guide for an upgrade of the application or for something new)?
- Are the users required to install the software or will this process be launched automatically in some other way, e.g. by a system admin?
- Will screenshots and diagrams help users to understand how to use the software?

Topics required for different groups of users may vary (e.g. topics covered in an Administrator's Guide will differ from those in an End User's guide). When topics are defined, these should be grouped in sections. Sections may refer to modules of the software (e.g. http://downloads.geant2.net/repository/cNIS/current-release/doc/maven-site/docs/docs/CMAUserGuide.html, Section 3 to 8), use cases or actions. Actions required to complete the use case should be described with procedural approach: divided into atomic steps and accompanied by appropriate screenshots, if necessary. Optional steps should be described in an IF-THEN manner.

Publication of the document is the final step of user guide creation. Its format should be considered regarding platform the software will be used on (e.g. it is not recommended to use MS Word format document for Linux-based applications – PDF is usually an adequate default).

The User Guide should be distributed with the application, thus it is guaranteed that its content is in line with the application version. In the case of web applications, the user guide may be available online. However, additional attention must be paid to provide consistency between the online documentation and individual features of the particular version of the application.

Examples

- http://forge.geant.net/docs/cnis/current/docs/docs/CMAUserGuide.html
- https://intranet.geant.net/sites/Services/SA2/T5/iSHARe/Documents/I-SHARe-OnLineDoc.doc
- http://cbt.geant.net/repository/gn3/i-share/I-SHARE_overview.htm
- http://www.geant2.net/upload/pdf/AMPS_2.0_UserGuide_1.0.pdf

5.2.2 Bugs and Support

Location: GN3 Forge Project Life Cycle -> Bugs Reporting section.

The purpose of this document is to explain a bug reporting procedure to a user, including a step-by-step guide of what to do if a user finds a bug.

Details about bugs can come from the database, but must be free from internal data relating to developers and testers. If known bugs are present in the published version of the software, details should be contained in the end-user document. Bugs should be described with as much detail as is necessary to be useful to users.

Description of the bugs should be in a form that is correct and easy for users to understand.

The user should be able to reference a list of known bugs contained in a document, check the website and then, if this is a new bug, follow the bug reporting instructions to report it by mail, in the issue tracking system, helpdesk, or a via a developer bug tracker, (depending on the specific profile of the organisation, application and available user support).

On the basis of these instructions, a user should also be able to determine whether the problem is already known, and should be able to get recommendations on how to avoid duplicate problem reporting.

The bug reporting instructions should be also incorporated in the document and provide the following instructions:

- Brief (one sentence), descriptive bug summary.
- Short description of what was expected to happen and what actually did happen.
- Detailed explanation how to reproduce bug, specific steps needed, etc.
- List environment information (platform, OS, version number, build number, browser).
- Priority/severity of the bug (priority – importance and urgency of resolving the error, severity – impact or consequence of the bug, frequency – probability of the bug occurring or manifesting itself to the end user, organisation or service). Priority would best be defined as a result of understanding the combination of both severity and frequency of a bug.

- Screenshots, a picture is worth a thousand words. They should not be too heavy in terms of size. Use .jpg or .gif formats (not .bmp).
- Logs. If exist, include it as attachments.
- Contact information.

Examples

- https://forge.geant.net/forge/display/perfsonar/Contact
- https://forge.geant.net/forge/display/ishare/Bug+Reporting
- http://httpd.apache.org/bug_report.htm

5.2.3 Online Demo

Location: GN3 Forge Main Menu -> Online Demo section

The Online Demo should be publicly available, and include required credentials. It will help users become familiar with functionality and layout of the product before they decide to install or use (as Software as a Service) it. The on-line demo may also be used for training purposes or as a sandbox for verification of new features. It is recommended (however, of course not always possible) to provide credentials for the most privileged type of user (admin), in order to allow potential users to verify as many functionalities as possible.

If an Online Demo cannot be provided (e.g. because of desktop nature of User Interface) it is recommended to prepare comprehensive instructional films.

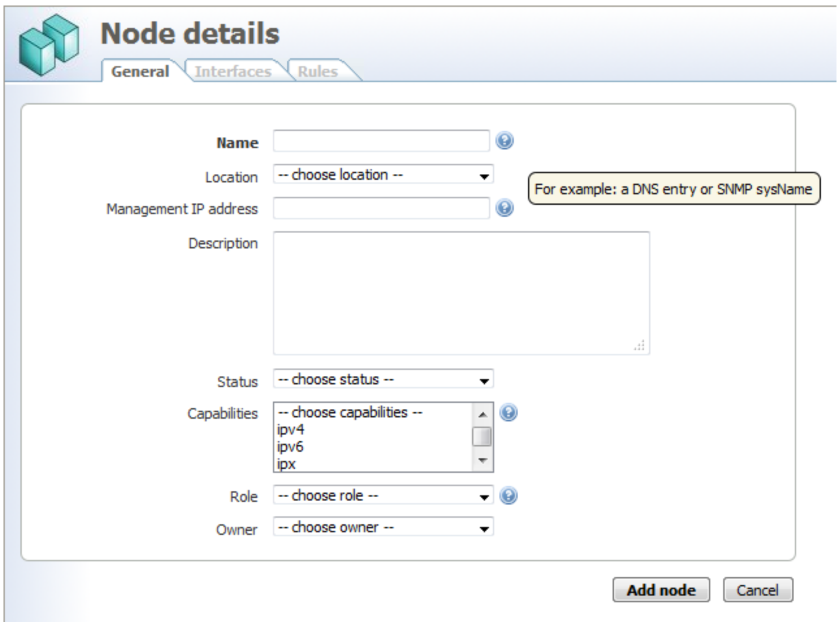
Examples

- https://forge.geant.net/forge/display/autobahn/On-line+Demo
- https://forge.geant.net/forge/display/cNIS/On-line+Demo

5.2.4 Integrated Help

Location: The help files should be integrated into software.

Integrated help significantly improves software usability. It provides the user with the information that is needed in the most convenient way, as it is not necessary to explore external documentation. However, additional effort is required to maintain this additional functionality. Integrated help may be available “on demand” (when help message is displayed after user’s action, e.g. in a form of tooltips) or permanently (e.g. descriptions of fields in a form). Both of these approaches are convenient from the user’s perspective. The decision about the type of integrated help is to be taken by the GUI designer. It is important to apply a consistent approach, because using both “permanent” and “on demand’ integration help would be extremely confusing to the user. Additionally, information that is displayed must be presented in a simple and straightforward way. Over-wordy sentences should be avoided, in favour of short statements followed by examples of correct values.



Node details

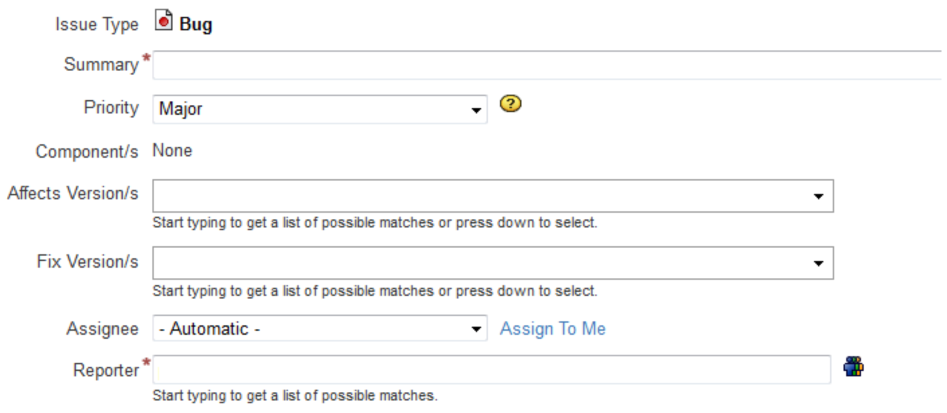
General | Interfaces | Rules

Name	
Location	-- choose location --
Management IP address	
Description	
Status	-- choose status --
Capabilities	-- choose capabilities -- ipv4 ipv6 ipx
Role	-- choose role --
Owner	-- choose owner --

For example: a DNS entry or SNMP sysName

Add node | Cancel

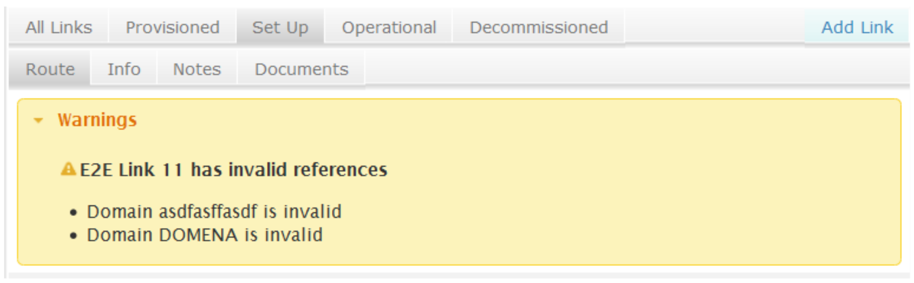
Figure 5.3: Integrated help that is available “on demand” (when the question mark icon is selected)



Issue Type	**Bug**
Summary*	
Priority	Major
Component/s	None
Affects Version/s	Start typing to get a list of possible matches or press down to select.
Fix Version/s	Start typing to get a list of possible matches or press down to select.
Assignee	- Automatic - Assign To Me
Reporter*	Start typing to get a list of possible matches.

Figure 5.4: Integrated help that is always available to the user

Error and warning messages can be part of integrated help functionality if these provide not only alert code but also additional information (e.g. explanation or example of correct value). If such feedback is provided, it is important to validate the inserted data as soon as possible, so that it could be easily corrected by the user. Error messages should be displayed with a red typeface (or background). Yellow is preferred for use in warning messages. In case of validation messages, an example of the correct value should also be displayed.



Warnings

E2E Link 11 has invalid references

- Domain asdfasffasdf is invalid
- Domain DOMENA is invalid

Figure 5.5: Example of warning message. (As recommended, a yellow background has been used)

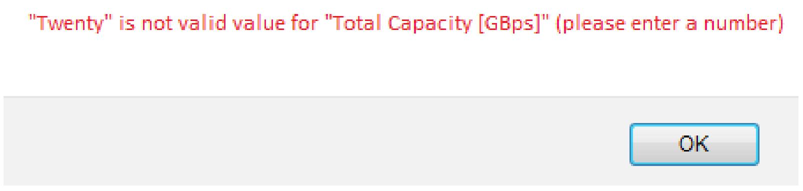

"Twenty" is not valid value for "Total Capacity [GBps]" (please enter a number)

Figure 5.6: Example of validation message. (As recommended, a red typeface has been used and an example of correct value is provided)

Examples

- http://issues.geant.net

5.3 Administrator Documentation

Typical reasons for producing administrator documentation include helping the administrators of the system and decreasing support costs, and using the documentation as a marketing tool for demonstration of completeness, configurability or openness, which gives the control to software users, etc. However, before starting to produce the documentation, identifying the reasons for producing the guide is also a crucial process. This will provide the guidance for the level of administrator documentation required, and its possible coexistence with other forms of administrator support. For example, the effectiveness of supplying administrator documentation in comparison with other types of administrator support should be seriously considered. At this stage, the level of access to the administrator documentation has to be decided. In general, it is best to have all documentation freely available. However, the materials may be kept internal or available on demand, only if the only expected scenario is that product maintenance will be taken over by the support team.

5.3.1 Software Requirements

Location: GN3 Forge Main Menu -> Documentation section.

In the literature [WIKISRS] a Software Requirements Specification (SRS) is a complete description of the behaviour of the system to be installed. The IEEE (www.ieee.org) is an excellent source for definitions of System and Software Specifications, with the IEEE 830 standard defining the benefits of a good SRS. For example, IEEE STD 830-1998 can be used as the basis for Software Specifications. In general, good software requirements should exhibit some basic characteristics **[SRSCHAR]**:

- **Correct**.

  Each requirement must accurately describe the functionality to be delivered.
- **Feasible**.

  It must be possible to implement each requirement within the known capabilities and limitations of the system and its environment.
- **Necessary**.

  Each requirement should document something the user really needs or something that is required for conformance to an external requirement, an external interface, or a standard.
- **Unambiguous**.

  The reader of a requirement statement should be able to draw only one interpretation of it. Also, multiple readers of a requirement should arrive at the same interpretation. Natural language is highly prone to ambiguity. Avoid subjective words such as: user-friendly, easy, simple, rapid, efficient, several, state-of-the-art, improved, maximise, and minimise.
- **Verifiable**.

  If a requirement is not verifiable, determining whether it was correctly described is a matter of opinion. Requirements that are not consistent, feasible, or unambiguous also are not verifiable.

  The Section of Software Requirements must include comprehensive information on the prerequisite software to enable the proper functioning of the system, together with a detailed description of how to obtain and install it. A thorough description of the Software Requirements Specifications is particularly significant, since it forces the concerned administrator to rigorously consider all of the determined requirements early in the installation procedure, thereby reducing the effort required to install the software product.

5.3.1.1 *Document structure*

In order to meet the requirement of completeness, it is recommended to use a specific Software Specifications template. In general, a brief list summarising the prerequisite software should be present, before proceeding to the detailed description. This list can summarise the title and the version of each software item, the direct URL to the software and its relationship to each part of the software suite installation (e.g. Required, Not needed, etc.). In addition, it would be desirable to describe each software item’s product dependencies. An indicative instance of the structure of such a list is illustrated below (taken from AutoBAHN documentation).

Prerequisite	AutoBAHN software Suite	Lookup Service	WebGUI
Java 1.6 VM	Required	Required	Required
PostgreSQL 8.x or higher	Required	Not needed	Not needed
Quagga 0.99.6 or higher	Optional	Not needed	Not needed
Tomcat 6 or 7	Required if cNIS is used, otherwise not needed	Required	Required

Table 5.1: Example of Software Specifications template

At this point, it is important to highlight that an author must be aware of the need to accurately state the supported operation system (OS) versions, since the installation process of each individual software item may vary by different versions of OS. Moreover, further software compatibility issues (e.g. supported and prerequisite version) must be checked and declared in detail.

At this point, there is usually a decision to be made about how to depict installation procedures for different platforms. The main criterion here is how different the procedures are – if the steps are drastically different, the procedure will need to be explained separately for each platform. But if the steps are not very different, you could choose the most common platform as your base, and wherever the steps are different, indicate the steps for the different platforms as indented text. However, it is recommended to use different sections during the comprehensive description of the software individual software installation steps, even though they have minor differences. It is possible to describe the installation process on different OS in parallel for each individual software component, but it is preferable, that the whole installation process is described in sections named according to the OS installation procedure that is described.

The overview of the required software must follow a detailed description of the installation steps of each individual software component, considering in the description order its potential dependencies. Obviously, the Software Requirements section must include analytical instructions describing the installation procedure of each required (or optional) software component step-by-step. The subsections containing this information should be written taking into account that everyone can follow the guide without prior knowledge of the described procedure. To accomplish this objective, the author should provide analytical instruction on each step, including issues that might be considered naive. Furthermore, even if a component could be installed using a wizard, it is recommended an alternative (manual) installation procedure to be available as an appendix, if possible.

Finally, it is helpful for administrators dealing with multi version environments or migrating/upgrading software versions if a separate subsection is devoted on software version consistency issues, considering cases associated with the system or individual software component updates.

Examples

- The AutoBAHN installation guide contains a software requirements section that is structured according to the provided guidelines: https://forge.geant.net/forge/display/autobahn/Downloads

- cNIS software requirements: http://forge.geant.net/docs/cnis/current/docs/docs/InstallGuideBook-main.html#sec_software_reqs
- https://forge.geant.net/forge/display/ishare/Installing+I-SHARe#InstallingI-SHARe-Softwarerequirements
- https://forge.geant.net/forge/display/perfsonar/MDM+Administrator%27s+Guide+-+Getting+Started
- Prerequisite Software and Requirements sections of subpages of https://forge.geant.net/forge/display/perfsonar/perfSONAR+MDM+Administrator%27s+Guide

5.3.2 Hardware Requirements

Location: GN3 Forge Main Menu -> Documentation section.

All software products have some minimal hardware requirements that need to be met to ensure stable work and acceptable performance. Usually, software also depends on other components, e.g. specific runtime environment, database, operating system, or is a part of larger software component. These components usually have hardware requirements of their own.

In all cases, authors must clearly state all hardware requirements that need to be met to run the software or that have impact on its performance. Such a statement is needed, even if the requirements are comparable with hardware configuration of standard computers. If software depends on other components, authors should point users to these requirements.

Meeting the hardware requirements of the software is crucial to the software's stability and reliability. In the worst case, not satisfying requirements might be a reason for software instability (if there are more tasks in a unit of time than system can process), lead to false results, or may prevent an application from providing them at all, due to some unexpected behaviour.

It is also important to take into consideration the specific nature of the environment where the software is supposed to be run. Although the list of factors that need to be taken into consideration when calculating hardware requirements depends on the nature of software, a few common factors may be identified. These include:

- Expected number of users.
- Expected number of requests per unit of time.
- Size of network (for network measurement tools) and its load.
- Network throughput (for network measurement tools).
- Number of measured metrics.

For example, systems calculating network statistics, running in a small network with low traffic, will have insignificant processing power requirements, require minimal amount of memory and store unnoticeable amount of data over time. But the same systems running in a large network with heavy traffic may use full power of a multi-core processor, gigabytes of RAM, and need to store hundreds of megabytes of data a day.

Authors of hardware requirement documentation should consider the following:

- **Architecture**

  If software requires specific hardware architecture (processor type, chipsets, dedicated device/architecture) to ensure stability, it must be clearly stated. High -level programming languages are usually architecture independent, and provide an abstraction layer over underlying physical architecture. Some languages (like Java or C#) require dedicated runtime environment. In these cases, it is enough to point at respective runtime environment and its list of supported architectures.

- **Processing power**

  CPU power requirement is probably the most difficult to define due to numerous factors influencing it, like architecture, number of processors, size of cache, communication subsystems, etc. Furthermore, users may have different definitions of “good performance”. What one user may see as “good enough” performance, others will describe as “unacceptable” due to his or her subjective impression. Therefore, rather than providing specific processor architecture, number of cores/processors and clock speed, it is better to provide a real-time examples of software performance on a selected platform under specific circumstances, for example:

  - It takes XX time to process a task of size YY on system ZZ.
  - System processes XX requests per <unit> on system ZZ.
  - On system ZZ this software uses XX% of processor power under YY conditions.

  This way, the user may predict how the software will behave in the specified system. It is also a good idea to test software on different platforms. Especially useful is comparison of performance of application on different number of processors/cores. This comparison will help users to decide if they should invest in system with faster processor, or in many less powerful (but also less expensive) processors.

- **Memory**

An insufficient amount of system memory (RAM) may have major impact on software’s performance. If there is not enough RAM memory, operating system starts to use slower (tens of thousands times slower) swap file as substitute. This seriously affects system’s efficiency.

- **Secondary storage**

Hard disk requirements cannot be limited to estimation the space needed to install the software, but should also include calculations on the space needed over a time (for log files, historical data storage, backups, etc.). The nature of the storage requirements should be described, along with factors influencing it.

- **Display adapter**

If an application uses only command-line interface or standard desktop environment, this requirement can be omitted. Otherwise detailed specification of the display adapter (manufacturer, list of drivers, etc.) should be provided.

- **Peripherals, connection**
- Any additional, critical requirements must be mentioned. Common requirements are:
  - Network devices and their speed/throughput
  - CD/DVD drives
  - USB ports

Examples

- http://forge.geant.net/docs/cnis/current/docs/docs/InstallGuideBook-main.html#d214e63
- https://intranet.geant.net/sites/Services/SA2/T5/AutoBAHN/Documents/Autobahn-install.doc

5.3.3 Installation Guide

Location: GN3 Forge Main Menu -> Documentation section.

An Installation Guide is written to describe the installation of a software framework. In most cases, an administrator installation guide overlaps with user installation guide since it covers similar configuration tasks. This is why the clear lines drawn must be drawn to divide the content of these two guides.

To find the correct level of detail for the Installation Guide, it is important for the author of the guide to put him/herself in the position of the users (system administrators in this case), and try to imagine and answer their questions. Focus should be placed on the primary users. Great care must be given to the fact that guides are written by one or more experts who know the terms, the assumptions, and the shortcuts in the subject area. However, non-experts are not expected to be aware of this information, due to their lesser experience with the subject matter, and the experts writing the guides should always be aware of this fact. Sometimes it is necessary to state things in a more obvious way because non-expert users do not know the subject. However, information that is not necessary under the expected software usage should not be added.

While producing the installation guide, some of the information can be included in a User or System Administration Guide as well. The decision regarding what information should go where is a high-level decision that should be made early in the development cycle, as it may affect software design. However, it is much better to repeat crucial information in more than one place since there is usually no control over the way technical manuals are used. Duplication of information, however, requires greater vigilance to keep everything up-to-date and consistent.

When in doubt, it is preferable to include important information in a number of guides, despite the risk of repetition.

Some hints that an author should always keep in mind when producing an Installation Guide are:

- Use clear headings that help to summarise the task.
- Make individual responsibilities clear. If a process involves more than one person, ensure each one is included when writing a procedure.
- Start each instruction step with a verb that tells the reader to do something, such as: "Open a new window…", "Press the button…"
- Use a numbered list when the order of tasks is important. Use a bulleted list when the order is not important (for example, when the reader can choose between different options).
- Place notes and warnings at the start of the instructions, or before the list item to which they refer.

- Specify conditions before the primary part of the instructions so that the reader may perform the installation steps as s/he reads them. For example, do not write, "Before you start the installation, make sure that…" at Step 5 of installation instructions.
- Do not mix instructions with conceptual information. Summarise the necessary background information before the instructions.
- Write for your audience and use a level of detail that is suitable to their skill level.
- Avoid lists of more than approximately ten steps. If possible, divide a long list of instructions into two or more different tasks.
- Specify what the reader does when the task is complete. If a reader is unsure, or asks “Now what do I do?” the instructions are not complete.

Once the installation guide is complete, double-check the guide’s accuracy. This is not a trivial procedure and the best way to accomplish it is trying to follow the guide in order to verify its effectiveness. It is also recommended to provide the Guide to a tester and let him/her follow the instructions without any interference from the software developers. They can only monitor the procedure, and note the steps where incomplete or poorly understood information led to installation errors.

Document structure

The following lists the main components of a typical software installation guide:

- Overview of System Features

  The overview should include the basic features/characteristics of the software as a list, or as a table.

- Minimal Quick Start Configuration

  This section can be published as a separate “Quick Start” or “Quick Start-Up” guide.

- Set-Up Configurations

  Usually a software will have a “typical” as well as an “exceptional” installation. This installation will also sometimes be referred as a “deployment”. The configuration description must be present if any configuration needs to be performed by the admin, even if it is trivial or the software is reasonably configured by an installer program or script.

- Maintenance, Error Messages and Troubleshooting

  All software systems display messages to give feedback about system status and to signal if there are problems with the system. This section lists such messages along with explanations and instructions on how to resolve the issue.

- Manual Installation Appendix

  If the installation process is performed through an installation wizard, a detailed guide to manual installation must be provided as an appendix.

- Migration Appendix

A “Migration” section may be either provided as an appendix or even as a standalone document, because it is not relevant for most administrators, e.g., it may only be needed for legacy versions of the software.

Examples

- The AutoBAHN installation guide: https://forge.geant.net/forge/display/autobahn/Downloads
- cNIS installation guide: http://forge.geant.net/docs/cnis/current/docs/docs/InstallGuideBook-main.html
- https://forge.geant.net/forge/display/perfsonar/perfSONAR+MDM+Administrator%27s+Guide
- https://forge.geant.net/forge/display/ishare/Installing+I-SHARe

6 Conclusions

Best practices described in this document, together with the ones provided in [**Error! Reference source not found.**], [**Error! Reference source not found.**] and [**Error! Reference source not found.**], will be treated as references during periodical audits conducted as part of SA4 Task 2 Quality Assurance. Similarly to SABP, SDP and QABP guides, this document may be changed/updated when requested by developers or when such a need arises (e.g. if GN3 Forge workspace template of workspaces is modified).

Appendix A DocBook Reference Material

Best practices related to DocBook documentation creation include:

Document structure:

"long" documents, like guides, books:

"book" form.

Document begins with "bookinfo" and it is divided into "chapters" and "sections" "BookInfo" contains "authorgroup" citing book authors.

"Date", "ReleaseInfo", "RevHistory" and optionally "legalnotice" and "copyright", see the example: book-cnis.xml.

"short" documents, like manuals, notes, articles:

"article" form.

Document begins with "articleinfo" and it is divided into "sections" "ArticleInfo" "authorgroup" citing book authors, "Date", "ReleaseInfo", "RevHistory" and optionally "legalnotice" and "copyright".

"Article" form is recommended option, while "book" should be used mainly for modular documents, i.e. those which consist of external independent documents

Required elements – you must use an element from the list below, if a part of text conforms to the meaning of this element:

- orderedlist – A list in which each entry is marked with a sequentially incremented label.
- itemizedlist – A list in which each entry is marked with a bullet or other dingbat.
- caution – A note of caution.
- important – An admonition set off from the text.
- appendix – An appendix in a book or article.
- author – The name of an individual author.
- bookinfo – Meta-information for a book.
- caution – A note of caution.
- chapter – A chapter, as of a book.
- classname – The name of a class, in the object-oriented programming sense.
- cmdsynopsis – A syntax summary for a software command.

- command – The name of an executable program or other software command.
- computeroutput – Data, generally text, displayed or presented by a computer.
- constant – A programming or system constant.
- copyright – Copyright information about a document.
- envar – A software environment variable.
- errorcode – An error code.
- errorname – An error name.
- errortext – An error message.
- example – A formal example, with a title.
- exceptionname – The name of an exception.
- filename – The name of a file.
- figure – A formal figure, generally an illustration, with a title.
- footnote – A footnote.
- funcdef – A function (subroutine) name and its return type.
- funcparams – Parameters for a function referenced through a function.
- pointer in a synopsis.
- funcprototype – The prototype of a function.
- funcsynopsis – The syntax summary for a function definition.
- function – The name of a function or subroutine, as in a programming.
- language.
- glossary – A glossary.
- interfacename – The name of an interface.
- methodname – The name of a method.
- methodparam – Parameters to a method.
- methodsynopsis – A syntax summary for a method.
- package – A package.
- para – A paragraph.
- procedure – A list of operations to be performed in a well-defined sequence.
- programlisting – A literal listing of all or part of a program, used additionally to list a part of configuration file in the text.
- property – A unit of data associated with some part of a computer system.
- replaceable – Content that may or must be replaced by the user.
- returnvalue – The value returned by a function.
- screen – Text that a user sees or might see on a computer screen.
- section – A recursive section.
- step – A unit of action in a procedure.
- structfield – A field in a structure (in the programming language sense).
- structname – The name of a structure (in the programming language sense).

- table – A formal table in a document.
- tip – A suggestion to the user, set off from the text.
- type – The classification of a value.
- ulink – A link that addresses its target by means of a URL (Uniform Resource Locator). Remember that this tag requires setting URL attribute value.
- uri – Universal Resource Locator, used for example to describe the address syntax (http://${tomcat.hostname}:${tomcat.port}/manager).
- userinput – Data entered by the user.
    - xref – A cross reference to another part of the document.

Recommended elements:

- acronym – An often pronounceable word made from the initial (or selected) letters of a name or phrase.
- action – A response to a user event.
- application – The name of a software program.
- beginpage – The location of a page break in a print version of the document.
- email – An email address.
- emphasis – Emphasised text.
- formalpara – A paragraph with a title.
- guibutton – The text on a button in a GUI.
- guiicon – Graphic and/or text appearing as an icon in a GUI.
- guilabel – The text of a label in a GUI.
- guimenu – The name of a menu in a GUI.
- guimenuitem – The name of a terminal menu item in a GUI.
- guisubmenu – The name of a submenu in a GUI.
- prompt – A character or string indicating the start of an input field in a computer display.
- quote – An inline quotation.
- variablelist – A list in which each entry is composed of a set of one or more terms and an associated description.
    - segmentedlist – A segmented list, a list of sets of elements.

Additional elements, worth using:

- database – The name of a database, or part of a database.
- menuchoice – A selection or series of selections from a menu.
- option – An option for a software command.
- optional – Optional information.
- parameter – A value or a symbolic reference to a value.
- sbr – An explicit line break in a command synopsis.
- subtitle – The subtitle of a document.

- symbol – A name that is replaced by a value before processing.
- programlistingco – A program listing with associated areas used in callouts.
    - screenco – A screen with associated areas used in callouts.

FAQ and other issues:

How to describe a host or a user system item – a system-related item or term – its attribute class can be set to the one of the following values:

- Enumeration:
- Constant
- Daemon
- Domainname
- Etheraddress
- Event
- Eventhandler
- Filesystem
- Fqdomainname
- Groupname
- Ipaddress
- Library
- Macro
- Netmask
- Newsgroup
- Osname
- Process
- Protocol
- Resource
- Server
- Service
- Systemname
- Username

How to describe a database schema:

- database: The name of a database, or part of a database.
- class: constraint, datatype, field, foreignkey, group, index, table, user, view, e.g:

```
<para>For many years, O'Reilly's primary web server, <ulink
url="http://www.oreilly.com/">http://www.oreilly.com/</ulink>, was
hosted by <application>WN</application> on <systemitem
class="systemname">helio.oreilly.com</systemitem>.</para>
```

"cmdsynopsis" is used to describe a command with parameters. Brackets are used to distinguish between optional, required, or plain arguments. Usually square brackets are placed around optional arguments, [-g], and curly brackets are placed around required arguments, {-g}. Plain arguments are required, but are not encased with brackets. For the sake of commonly used conventions of describing a command, which states that obligatory parameters are not present in any brackets, a command will be described as follows:

```
cnis-client [-s profile] operation [argument].
```

instead of:

```
cnis-client [-s profile] {operation} [argument].
```

Sometimes the XLST engine does not highlight some tags in the output document, which looks inappropriate. It is a render engine-specific issue and should not be treated as a high-priority problem. Moreover, it is highly undesirable to mark the text with a tag, which does not conform to its semantics to obtain a properly formatted output (e.g. use the 'filename' tag instead of 'property' to bold the text, when the present XLST engine omits the 'property' tag). There are two possible ways to handle this issue: change the XSLT engine to another or accommodate the existing XSLT style sheet to the one's needs.

References

[cNIS]	http://www.geant.net/service/cnis/About_cNIS/Using_cNIS/Pages/Using_cNIS.aspx
[DBD]	http://www.fabforce.net/dbdesigner4/
[DocEval]	https://intranet.geant.net/sites/Services/SA4/T1/Documents/Best%20Practices%204.0%20changes/Documentation%20evaluation%20summary_2012.xls
[Forge]	https://forge.geant.net/forge/display/ishare/Screenshots
[KnowledgeBase]	https://kb.mdsd.geant.net/
[I-SHARe]	https://svn.geant.net/fisheye/browse/SA2T5-ISHARe/trunk/central/ishare-central-server/src/main/doc/readme.txt?hb=true (link requires GN3 login) https://intranet.geant.net/sites/Services/SA2/T5/iSHARe/pages/home.aspx#Introduction
[**Merson**]	Merson, P, *Data Model as an Architectural View*, Software Engineering Institute, Carnegie Mellon, October 2009
[PARCH]	http://www.sqlpower.ca/page/architect
[perfSONAR]	http://wiki.perfsonar.net/jra1-wiki/images/8/89/PerfSONAR-mLS-paper.pdf
[PMO]	The Programme Management Office, is home to the GÉANT technical authors, who can also assist with your documentation, see: https://intranet.geant.net/sites/Management/PMO/TA/Pages/home.aspx
[PSDRULES]	faculty.ccri.edu/mkelly/COMI1150/PseudocodeBasics.pdf
[QABP]	The most recent version of the *GN3 Quality Assurance Best Practice Guide* can be accessed from the GÉANT Intranet at: https://intranet.geant.net/sites/Services/SA4/T1/Documents/Forms/AllItems.aspx
[README]	http://www.mactech.com/articles/mactech/Vol.14/14.10/WritingAReadMeFile/index.html
[ReadySET]	Ready-to-use software engineering templates: http://readyset.tigris.org/
[REQ1]	http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=720574&contentType=Standards&queryText%3Drecommended+practice+for+software+requirements+specifications+830+1998
[REQ2]	http://techwhirl.com/skills/tech-docs/writing-software-requirements-specs/ – Writing Software Requirements Specifications
[RFC]	http://www.rfc-editor.org/rfcxx00.html
[SABP]	The most recent version of the *GN3 Software Architecture Strategy Guide* can be accessed from the GÉANT Intranet at: https://intranet.geant.net/sites/Services/SA4/T1/Documents/Forms/AllItems.aspx
[SDP]	The most recent version of the *GN3 Software Developer Guide* can be accessed from the GÉANT Intranet at: https://intranet.geant.net/sites/Services/SA4/T1/Documents/Forms/AllItems.aspx
[**Simsion and Witt**]	Simsion, G.,and Witt, G.C., *Data Modelling Essentials*, Elsevier 2004
[SRSCHAR]	http://standards.ieee.org/findstds/standard/830-1998.html

Glossary

API	Application Programming Interface
DDL	Data Definition Language
CPU	Central Processing Unit
EER	Enhanced Entity-Relationship
GN3	The GN3 Project (3rd GÉANT project year)
GPL	General Public Licence
GUI	Graphical User Interface
IDE	Integrated Development Environment
ORM	Object-Relational Mapping
OS	Operating System
PMO	Programme Management Office
PR	Public Relations
QASPER	Quality Assurance and Public and External Relations Committee
RDBMS	Relational Database Management System
RFC	Request For Comments
SA	Service Activity
SME	Subject Matter Expert
SRS	Software Requirements Specifications
UML	Unified Modeling Language
XML	Extensible Markup Language
XLST	eXtensible Stylesheet Language Transformations
Y3	Year 3