Quality Committee Handbook

openKONSEQUENZ

created by

Quality Committee

Revision History

Version	Date	Reviser	Description	Status
1.0	2016-06-20	M. Jung	Alignment in AC/QC conference call	Released
1.0.11	2016-08-12	M. Jung	Removed UML tool discussion, added UML tool decision in design documentation, Added configuration documentation, Removed stale “Related docs” appendix Removed “Meeting structure” appendix (structure already implemented). Added details for commit rules, library usage, and GUI styleguide application.	Draft for v1.1
1.1	2016-08-18	A. Goering	Alignment in AC/QC conference call	Released

Formal

According to a decision of the Quality Committee, this document is written in english.

Document control:

Author: Dr. Martin Jung, martin.jung@develop-group.de (representative of the quality committee)

Reviewed by: SC, PPC, and AC of openKONSEQUENZ

Released by: QC

Released versions will be made available via the openKONSEQUENZ web site.

Related documents

Document	Description
BDEW Whitepaper	Whitepaper on requirements for secure control and telecommunication systems by the german BDEW Bundesverband der Energie und Wasserwirtschaft e.V. (https://www.bdew.de/internet.nsf/id/232E01B4E0C52139C1257A5D00429968/$file/OE-BDEW-Whitepaper_Secure_Systems%20V1.1%202015.pdf)
BSI TR-02102	Technical Guideline according to encryption recommendations and key length by the german BSI - Bundesamt für Sicherheit in der Informationstechnik (https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr02102/index_htm.html)
oK-Charter	The openKONSEQUENZ charter (https://wiki.eclipse.org/images/f/f5/20150623a_openKonsequenz_V14-3_%283%29.pdf)
oK-GUI-Styleguide	Style guide for module developers of openKONSEQUENZ modules according to the graphical user interface. (http://wiki.openkonsequenz.de)
oK-Interface-Overview	AC handbook external but related AC document (appendix), where the Interfaces and the overall oK-CIM-Profile of oK-Modules are described in short as well as showing the Building Block View Level 1. (https://wiki.eclipse.org/OpenKONSEQUENZACQCRichtlinien)
oK-Module-Tender-Call	The openKONSEQUENZ project planning committee prepares a document which describes the requirements to the development for each module. With this document it calls for tenders at software developers (module individual)
oK-Module-Tender	The software developers answer to the oK-Module-Tender-Call (module & developer individual)
oK-Vision	The oK document “Vision/Mission/Roadmap” - it is currently not available online.
oK-Website	The website of openKONSEQUENZ (www.openkonsequenz.de)

Architecture Committee Handbook	Textural guideline for module developers of openKONSEQUENZ modules with respect to architectural design & documentation. (https://wiki.eclipse.org/OpenKONSEQUENZACQCRichtlinien)

The Architecture Committee Handbook also contains the **glossary** of terms!

Table of Contents

1. Introduction

This document characterizes the role of the quality committee (QC) in openKONSEQUENZ. It defines quality criteria for the projects to be developed in the context of openKONSEQUENZ. While QC acts as a control authority, the Architecture Committee (AC) creates and maintains the overall software architecture, and gives rules for the creation, documentation and implementation of architecture in the various projects. The rules for architecture are described in the oK-AC-Handbook (in the following: AC-Handbook).

1.1 Mission of QC

The mission of the quality committee (QC) is to define, maintain, and enforce the guidelines for development in the openKONSEQUENZ group.
The QC defines the quality standards for project work (process quality) and project outcome (product quality). It provides procedure and methods for projects to use, and selects tools to be used (e.g. for documentation).
The QC also defines the integration- and QA-environment (see chapter III below).
The QC also monitors and evaluates projects and gives feedback to the projects and to the AC, the project planning committee (PPC), and the steering committee (SC). The QC supports the other committees in their work.

1.2 Responsibility of QC

The quality of the software code base in openKONSEQUENZ is ultimately the responsibility of the QC. The AC will focus on the construction of the software and on enabling the quality goals; their check and control are responsibility of the QC. Providing infrastructure for continuous tracking and reporting of the quality indicators during project work is also responsibility of the QC. If projects experience difficulties when applying the quality rules and regulations, it is the QCs job to support the projects and to help find a viable solution for their work. Finally, the QC is responsible to analyze project outcome, and to give technically sound recommendation to the PPC and SC whether to accept or to refuse a project outcome.

1.3 Competence of QC

The QC may reject a project outcome based on quality analysis. If either the source code itself, the software running on the QA environment, the documentation, or any other mandatory project result does not meet the quality requirements, the QC will advise the SC and the PPC to reject the project's outcome and to plan re-work. Such an advisory shall be

made in writing and published via the openKONSEQUENZ mailing list. The advisory may be challenged by the project and decision will then be escalated to the SC.
Whenever project proposals are prepared, the QC may alter the description of the project scope to include key quality requirements and the core quality rules, before a public call for proposal. Least of all, the QC will ensure that this document is a part of any official project proposal. Furthermore, the QC will alter the PPC's description of a project only, if specific quality aspects are especially important or have a high impact on the project scope.

1.4 Ownership of documents

To define the quality rules and to enforce them, the QC creates, maintains and releases documents, and defines the development environment.
The list of documents are:

- This document "openKONSEQUENZ: Quality Committee Handbook"
- A directory of coding guidelines (currently co-located in this document, see Appendix E)
- A list of acceptance criteria for project outcomes (to be defined)
- A repository containing standardized test data (to be initialized)

The build infrastructure, its technical specification, and a HOWTO for its usage are also owned by the QC.

2. Quality Rules and Guidelines

The following sections define the quality requirements on development in the openKONSEQUENZ group.

2.1 Project and module Classification

The project and its modules shall be classified using the following rules:

- Complexity on a scale of Small, Medium, Large
- Criticality on a scale of Normal, High, Very High

Criticality

This document classifies criticality as a vector in three dimensions, **availability**, **confidentiality**, and **integrity**. The Criticality values are related to the definitions in the International Standard ISO/IEC 27000. Whenever one dimensions is classified “high”, the project’s criticality is considered to be “high”, and whenever one dimensions is classified “very high”, the projects’s criticality is considered to be “very high”. The classification should be done according to the approach of Determining the protection requirements, described in the BSI-Standard 100-2 (see Appendix A, [1]) .

Complexity

Complexity is related to the size of a module, and can be coupled to typical sizing like story points or (in this case) Person Days (PD).

2.2 Review method

The overall classification shall be documented in each project proposal. It is required to determine the review methods as stated in the table below:

Complexity / Criticality	Small (< 120 PD)	Medium (120 - 240 PD)	Large (>240 PD)
Normal	No manual review required	Peer Review	Peer Review
High	Peer Review	Peer Review	Walkthrough
Very High	Peer review	Walkthrough	Deep inspection

The review methods apply to documentation, code, test specifications, and test protocols. “No manual review” means automated checks on the code are sufficient. “Peer review” means an offline check by AC or QC representatives. A representative is nominated by the committees and is not necessarily a member; e.g. he/she may be another committer or

developer, preferably from an other project. “Walkthrough” means an online check by an AC or QC representative, the author explains the solution. “Deep inspection” means an online check by an AC member and a QC member, the author and the project lead explain the solution. The author is responsible to organize the review.

2.3 Code quality

File system conventions

Uniform appearance of the project directory structure and consistent naming of files are mandatory in collaborative work.

- The standard maven project directory structure[1] shall be used.
- The project shall provide AsciiDoc files for all documentation purposes. The files are enumerated below, see appendix C “Project Directory Layout”.

Coding guidelines

Uniform appearance of source code, the readability, and the avoidance of error-prone statements are key to good code quality.

- The coding guidelines shall be adhered to. See appendix E “Coding Guidelines” below.
- Functions/Methods shall be documented using Javadoc, JSDoc, Doxygen, or equivalent source code markup systems.

The coding guidelines selected for openKONSEQUENZ will be implemented by rulesets of software tools (e.g. PMD, Checkstyle, FindBugs). They will then be enforced by the continuous integration procedure. The ruleset will be made available via the openKONSEQUENZ wiki or a source code repository.

Commit rules

To guarantee a up-to-date view on the codebase, and to allow all developers to use functionality as soon as possible, it is important to commit early and often. Functionality that is presented in a Sprint Review Meeting has to be pushed to the repository before the Sprint Review Meeting. Smaller sets of functionality (e.g. an interface without implementation, an abstract class, etc.) should be pushed as soon as they are testable. To guarantee a continuous flow of development result, a push is required every two weeks (mid-sprint push & sprint review push).

Unit tests

Unit-testing is a standardized method to assure quality on source-code level.

- Unit tests shall be written and cover 100% statements and 80% branches.
- At least one unit test case shall be written for each use case (or user story).
- Unit tests may use their own, individual test data. If possible, they shall use standardized test data, in order to avoid “wrong” data.

[1] https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

- All unit tests shall be executed in the daily build of the integration environment.
- Coverage metrics shall be reported on a daily basis via the integration environment.

Configuration management

The configuration (Versions of different components, of external libraries, of configuration files) shall be described in the maven POM files.

- All code, configuration files, data, and documents shall reside in openKONSEQUENZ' git repositories.
- Dependencies among components of the openKONSEQUENZ software and to/from external software components shall be explicit. This shall be achieved by maintaining maven pom.xml files. No local libraries (implicit dependencies) are permitted.

To freeze versions for test and release, the versioning numbers shall conform to the pattern X.Y.Z.A, where:

- X: Major Release, incremented on major changes like interface definitions or end-user visible functionality
- Y: Minor Release, incremented on each delivery. Stable versions have even numbers, developer versions are odd numbers
- Z: developer tag, incremented on each code change
- A: Git-Hash, git-commit reference for easier handling

During development, feature branches shall be used, the name of the feature is added as part of the version string like this: X.Y.featurename.Z.A

All check-ins need to be commented. The comments shall be plain text and shall contain the following information:

- Topic (max. 50 characters)
- Task reference (to backlog item, requirement, defect, ...; always using the ID of the task)
- Reason: Why was this change necessary?
- Rationale: What has been done (on an abstract level)?
- Side effects: What else did change?

Build, package & test

Since the software shall be widely accessible, an automatic method to build, package and test it is required. The maven tool will be used to control these tasks.

- The maven files shall encompass functionality to build the software, create runnable packages, run the unit-tests, and create all documentation files.

Diagnosis, Exceptions and Errors

Each module shall offer interfaces to access diagnosis information such as operational state, errors or exceptions. All errors and exceptions shall be logged to a plain text file. The format of log messages shall comply to the following rules:

- Exactly one line of text for each incident

- Each line shall start with a timestamp, a module designation, a thread ID in a comma separated line header: “YYYY-MM-DD_HH:SS:ssss, moduleXYZ, TID4711823, ...”
- Other information relevant for the module may follow in a comma separated list
- Other information may contain json formatted object information. It is mandatory that json text is in one line and is correctly enclosed in curly braces
- If human readable text is part of the log messages, english language shall be used.
- Personalized Data and credential information must not be written to log files.

2.4 Design quality

Design Documentation

All projects shall document their technical solution concept. The design documentation has to conform to the architectural guidance specified by the AC-handbook. Technical decisions shall be explained, their alternatives and consequences of the decision documented. UML description shall be used as appropriate, according to the tooling decision, see Architecture Committee Handbook, section 2.1.
All libraries used by a project have to be listed in the design document, together with a rationale for their usage. Libraries that are already cleared are default, they have to be used unless there are valid reasons against their usage (a list of cleared libraries can be obtained from the eclipse foundation). If a project chooses to override this default, it needs approval of the architecture committee to do so. In that case the project is responsible to do the IP clearing. The clearing has to satisfy the publication license of openKONSEQUENZ. The clearing of the library has to be done as early and as fast as possible. If a clearing fails, the library must not be used, all development effort based on such a library is wasted.

Statement of Qualities

Each project shall document its quality requirements in the project’s architecture documentation (according to the AC handbook). Typically, not all quality attributes are relevant for a project or module. The quality dimensions as defined in ISO 25010 are a frame of reference. Whenever a quality is of special concern to the project, the impact on the solution has to be documented. The AC refines system requirements down to project or module requirements. For example, if a persistence module impacts the overall availability of the system, the AC breaks down the overall system availability requirements to this module. The module has to document these requirements, and has to state how they are implemented, verified and validated (see next sections).

Verification

One of the aspects of verification are the static analysis of source code and the design and code reviews. These need to be performed according the review method matrix, see table “Review Method” above.

The other aspects are tests. Unit tests are required to check basic code functionality, and to maintain code quality. Design tests check the technical solution against the functional and quality requirements.

- Module test specifications shall be defined for each module. Using black box test specifications, the test specification specifies how the requirements and quality attributes are checked.
- Acceptance criteria shall be defined for the module test specification.
- The test data required to run the module tests shall be specified, with respect to the global data model and the global test data set.
- Mock functionality (https://en.wikipedia.org/wiki/Mock_object), required for the execution of the module test, shall be specified.
- Test specifications, test data, and mock functionality shall be formalized in test code that can be run automatically. This might not be possible for all test specifications. Manual test have to specified as a working recipe for a tester.

Design Reviews

Design Reviews shall be carried out according criticality and complexity rating and the resulting review method. Review records shall be stored in the filesystem according to Chapter V “Project directory layout”.

Document list

- Project architecture concept. Contains the mapping to overall architecture and the break-down into modules. Also, it contains the rationale for technical decisions and alternatives, the statement on qualities and the criticality/complexity rating. This document shall conform to the guidelines by the AC.
- How-to information for developers and integrators. Contains information on building, running, coding, and testing in this project
- Review records. Contains a list of participants, findings, and resolutions
- Test specifications for module integration test. Contains the strategy for integrating this projects modules, and a concept to test the integration.
- Test protocols. Contain the test setup used and a verdict. If appropriate, execution logs are also included.
- For each module:
    - Design concept. Contains technical solution concept and coding instructions
    - Review records. Contains a list of participants, findings, and resolutions
    - Test specifications for module test. Contains the concept of testing, automated and manual parts
    - Test protocols. Contain the test setup used and a verdict. If appropriate, execution logs are also included.

2.5 Product quality

Automation in development

In order to maintain a high overall product quality, a high degree of automation is a key to handling this job efficient. Automation refers to the following steps:

- Immediate Verification of coding results: This is typically implemented in the IDEs running automated unit test code and static analyzers on the source code. If all tests on this level are passed, the software gets forwarded to the next stage.
- Continuous Integration: All development results are transferred to a dedicated build and test environment. On a daily basis, the development results are compiled, unit tests, integration tests and overall software tests are run automatically. Metrics and other analysis reports are gathered and provided as a feedback to the developers. If all tests on this level are passed, the software gets forwarded to the next stage.
- Continuous Deployment: All development results are transferred to a "QA stage", a computing environment as close to the final production environment as possible. The installation of the software in the QA stage is automated and triggered by successful integration builds. The software tests are run again, and additional (typically: manual) test steps are performed.
- Continuous Delivery: If the quality assurance on the QA stage has run successfully, the software is automatically distributed to end-users.

The goal of this automation is to erase as much manual effort of testing as possible. Tests, refactorings, integration of additional functional, all will be rather painless, because the robust testing environment protects the existing functionality.
openKONSEQUENZ will employ the first three steps, unit testing, continuous integration and continuous deployment.

Continuous Integration

Projects shall create test specifications which are run during continuous integration. The test cases shall determine, whether the integrated software's basic functionality is correct.
The continuous integration environment will calculate metrics, perform static and dynamic analysis, and will generate a feedback website showing the results of the tests and the metrics.

Continuous Deployment

After successful run of the continuous integration tests, the software build can automatically be forwarded to the QA environment. While continuous integration runs at least once per day, QA testing after deployment runs at least once per sprint.
Projects shall create test specifications for software/system test on the QA stage. This includes automated tests, automated UI-Tests, and descriptions for manual test steps.

Validation

Validation - the check whether or not the project delivered software like the PPC envisioned, and like the users require it - is performed on the QA environment. This means, that representatives from PPC or end users perform the validation. Projects shall create a validation specification that states, for each use case, the scenarios a end user should execute in order to validate the system behavior. Minimum required validation is a 3 months test operation on the QA environment. Additional validation measures shall be documented in the Validation concept.
Validation will also determine whether or not the project's outcome satisfies all regulations, e.g. the GUI-styleguide (see also the Architecture Committee Handbook, section 8.4, for usage of the styleguide document).

Document list

- Project test specification. Contains the strategy for testing the complete project outcome. This document also contains the description of manual tests. If the mock-up specifications (see 3., sect. “Verification”) do not cover all aspects needed for validation purposes, they have to be supplemented here.
- Validation concept. This documents contains a description of all manual tests to be executed by an end user in order to validate project outcome.
- The project shall deliver a “User documentation” (according to App. A, [3] “BDEW Whitepaper”, sect. 2.1.2.2)
- The project shall deliver a “Administration documentation” (according to App. A, [3] “BDEW Whitepaper”, sect. 2.1.2.2)

3. Development Setup

3.1 Environments

Development Environment

The environment for development is not regulated. It is *recommended*, that development environments use syntax checkers and static analyzers (e.g. PMD, Checkstykle, JSLint, ...), but it is neither prescribed nor will it be checked. Development Environments shall use and run the Unit Test frameworks. Development Environments shall use git and Maven.

Integration Environment

The integration environment uses git and Maven to access the code and to build/test/deploy the software. It runs gerrit as a tool for reviews, Hudson as a continuous integration platform, SonarQube for metrics, and Nexus for software distribution. The integration environment also runs the Bugtracker (Bugzilla).

Quality Assurance Environment

The Quality Assurance (QA) environment uses the Nexus of the Integration Environment to get access the software. It uses Maven to run the automated software tests. Other than this, the QA Environment is similar to the production environments.

3.2 Development Tools

This section lists a set of development tools approved for use. The actual use of tools in a project is highly dependent on the technology of the project. Tools can be added in agreement with the AC.

Build automation

- Maven: https://maven.apache.org/

Source code management

- Git: https://git-scm.com/

Source code conventions and static analysis

Used for source code checks on development and integration environment.

- PMD: https://pmd.github.io/
- Checkstyle: http://checkstyle.sourceforge.net/
- Findbugs: http://findbugs.sourceforge.net/
- JSLint: http://www.jslint.com/

Automated code review tracking

- Gerrit: https://www.gerritcodereview.com/

Issue tracking

- Bugzilla: https://www.bugzilla.org/

Continuous integration

- Hudson: http://hudson-ci.org/

Source code and test coverage analysis

- SonarQube: http://www.sonarqube.org/

Software distribution and dependency management

- Nexus: http://www.sonatype.com/download-oss-sonatype

Appendix

A. Project directory layout

The following directory structure and file name conventions shall be used (*.adoc are asciidoc files). The files mentioned explicitly contain the documentation according to chapter II “Quality Rules and Guidelines”. Other files (e.g. code and parameter files) shall be stored according to the maven standard directory layout.

```
/get_started.adoc (how to get started with the project, where to look, what to do)
/howto/build.adoc (how to build the software)
/howto/run.adoc (how to run a demo or the application)
/howto/code.adoc (how to set up for coding)
/howto/test.adoc (how to set up and execute tests)
/howto/config.adoc (how to set config parameters, semantics of parameters)
/documentation/user.adoc (user documentation according to 4. Product Quality)
/documentation/admin.adoc (admin documentation according to 4. Product Quality)
/arch/architecture.adoc (architecture concept and mapping to overall architecture)
/arch/model/* (UML tool files)
/arch/images/*.png (UML diagram exports and other image files)
/configuration/* (config files, esp. A configuration file for QA-environment)
/reviews/YYYYMMDD-scope-type-author.adoc (review records)
                    (e.g. 20160429-ControllerComponent-Peer-ScroogeMcDuck.adoc)
/test/integrationtest.adoc (integration test concept for the project)
/test/test.adoc (software test concept for the project)
/test/validation.adoc (validation concept for the project)
/test/protocols/YYYYMMDD-type-tester.adoc  (test execution records)
                    (e.g. 20160501-Integration-DonaldDuck.adoc)

/dirX (directory for module X)
/dirX/* (follow maven conventions from here)
/dirX/src/site/resources/arch/design.adoc (design and mapping to project
architecture)
/dirX/src/site/resources/arch/model/* (UML tool files)
/dirX/src/site/resources/arch/images/* (UML diagram exports and other image files)
/dirX/src/site/resources/reviews/YYYYMMDD-scope-type-author.adoc (review records)
                    (e.g. 20160429-ControllerComponent-Peer-ScroogeMcDuck.adoc)
/dirX/src/test/resources/specs/moduletest.adoc (test concept for the module)
/dirX/src/test/technology/* (test code, e.g. JUnit code in /dirX/src/test/java/*)
/dirX/src/test/resources/protocols/YYYYMMDD-scope-type-tester.adoc (test exec
records)
                    (e.g. 20160430-ControllerComponent-Module-DonaldDuck.adoc)
```

The names of the modules (here: “dirX”) shall be taken from the overall architecture definitions.

B. Coding Guidelines

The overall technology stack of openKonsequenz is not yet defined. The following guidelines are subject to change. (In the future, they may be put in another document for easier version control.)

The following list of coding guidelines shall be adhered to:

Java

- https://google.github.io/styleguide/javaguide.html
- www.securecoding.cert.org/confluence/display/java
- http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html
- https://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html

JavaScript

- https://google.github.io/styleguide/javascriptguide.xml
- http://www.w3schools.com/js/js_conventions.asp

SQL

- http://www.sqlstyle.guide/

XML

- https://google.github.io/styleguide/xmlstyle.html

JSON

- https://google.github.io/styleguide/jsoncstyleguide.xml

The following list of metrics shall be calculated during the development:

- Size of code base
- Comment ratio
- Cyclomatic complexity
- Test coverage (line and branch)

Additional pointers to good practice:

- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013
- https://cwe.mitre.org/top25/index.html