ClaRa⁺ Quality Management: Efforts for Certification According to ISO 9001:2008

Detailed Description of the Quality Management Systems at the ClaRa⁺ Development Team

Document version 1, Date: 01. June, 2018, linked to ClaRa⁺ release 1.1.0

Authors

Dr. Friedrich Gottelt, Timm Hoppe
XRG Simulation GmbH
Harburger Schlossstr. 6-12
21079 Hamburg, Germany

Dr. Lasse Nielsen
TLK Thermo GmbH
Hans-Sommer-Str. 5
38106 Bunswick, Germany

Contact

www.powerplantsimulation.com | info@claralib.com

ClaRa+ is an enhancement of ClaRa which was funded by the German Ministry of Economic Affairs and Energy (grants 03ET2009 and 03ET7060)

Supported by:
Federal Ministry for Economic Affairs and Energy
on the basis of a decision by the German Bundestag

TLK-Thermo GmbH XRG Simulation GmbH

Content

Scope of Document

XRG Simulation GmbH and TLK-Thermo GmbH as the main developers of the Power Plant Modelling Library ClaRa$^+$ aim to attain certification according to ISO 9001. This document gives a detailed explanation of the current Quality Management systems at XRG Simulation GmbH and TLK-Thermo GmbH with respect to ClaRa$^+$ development.

Summary

The software engineering processes of ClaRa$^+$ are designed to meet the following requirements

1. **Planning:**
   The extent of implemented features, improvements etc. is thoroughly planned for each release. Every team and its members continuously plan the precise implementation efforts.
2. **Tracking:**
   Every issue, design decision, implementation effort, support effort etc. is tracked through multiple cross-referenced instances: The version control of the codebase, the issue tracking system, the application lifecycle management and the internal integration test environment.
3. **Accountability:**
   Since every contribution and operation is tracked throughout the life cycle, it is possible to account for any change in the functionality and every contribution leading up to that change.
4. **Ability to react:**
   It is possible at any time to release a stable version or release a stable version containing specific hotfixes. The employment of a branching model allows engineers to quickly switch between different development efforts without comprising the codebase.
5. **Risk assessment:**
   With every new feature, interface, tool or general improvement, the risk is assessed. Especially redesign generally entail risky contributions to the codebase. For this reason, internal engineering processes protect certain development branches from such risky contributions. Features are generally implemented separately before being merged back into the main development as this often entails risky changes.
6. **Support:**
   The support process for ClaRa$^+$ provides a flat hierarchy in which a user is forwarded to a specialist directly after initial contact. Any issues reported from users are immediately fed back into the internal issue fixing process.

By meeting these simple requirements, The ClaRa⁺ development team employs a high level of quality control for its product ClaRa⁺.

Basic Release Schedule

The ClaRa release schedule defines two kinds of primary releases: Major product releases and minor maintenance releases. Major releases are scheduled once a year in winter (end of January) while minor releases are scheduled each summer (end of August).

The major releases include new features, changes to the user interface, performance improvements, introduction of new tools and interfaces, fixes of known bugs, and an updated documentation. Through the application of conversion scripts, a full backward compatibility will be granted in most cases.

Minor releases provide a fully backward compatible enhancement of the library. In this case only valuable improvements of the modelling physics and the user interface are published which leave the results of existing application models unchanged. Thus, minor releases may provide an enhancement of the documentation, new component models, new component parameters having a default value equivalent to the previous model version and (in exceptional cases) major bug fixes that correct erroneous results.

In addition to this release schedule, ClaRa⁺ is supported with hotfixes. Hotfixes provide immediate issue resolution for any kind of problem with the simulation functionality outside the usual release schedule.

This schedule enables short reaction times when ensuring optimal reliability of the modelling and simulation algorithms while enabling the software engineering to fundamentally improve the reliability of the algorithms and user experience.

Testing of the Simulation Functionality

Throughout its development ClaRa⁺ is tested in two phases:

1. During development: ClaRa⁺ development engineers apply unit tests, static code analysis and explorative testing wherever applicable in order to account for their respective contributions to the code base.

2. Product testing:
    i. Manual tests of new features by associated engineers in industry and research projects
    ii. Continuous integration tests ensure correct simulation and computation of results. The automated test environment was developed in-house and contains about 100 simulation models which span all supported engineering domains. Integration tests consist of models of differing complexity and different scope (some models implement very specific and potentially critical modelling tasks while other implement complex yet computationally uncritical models). The test models are tested in regards of their general ability to run in the simulator (symbolic pre-processing and numerical solving) and if applicable the correctness of the computed results. No release candidate can ever enter production without prominent improvement over prior releases.
    iii. Continuous integration tests during development of specific features allow engineers to implement a feature and test it within the test environment provided for the product itself without having to apply potentially risky changes to the codebase used by all engineers. This ensures new features are tested before contributing it to the product.

Issue Tracking and Continuous Integration

Issue Tracking

Whenever an integration test run has finished, all software engineers who have contributed to the version employed in the test run are notified about the test results. If issues are found, an internal bug fixing process is started by creating a ticket in the issue tracker and assigning it to the engineer responsible for the software component potentially responsible for the issue. The responsible engineer reviews the ticket and further isolates the issue at which point he or she will fix it if possible or

- Pass the ticket to another engineer if the issue does not fall within his or her responsibility or expertise or
- Employ pair programming or other techniques if the issue spans several components or responsibilities.

If an issue requires deeper redesign, it will become part of the product backlog (see “ALM”). Bug fixing retains precedence over all other development tasks.

If an issue can be fixed while a redesign in a future release could potentially eliminate the possibility of such issues occurring in the first place, an engineer will provide a fix for the bug at hand while developing a potentially more time and resource critical redesign for a future release (see Version Control).

Throughout all phases of development, issues are always fully accounted for, are traceable throughout their bug fixing process and are fully connected to contributions to the codebase.

If issues are reported by users by mail, those issues are reviewed and undergo the same processes as issues reported internally.

Application Lifecycle Management

The ClaRa$^+$ development team employs a SCRUM-based [1] framework to perform agile software engineering. Issues which require redesign, potential new features and user stories are logged in the product backlog. Twice per year, a cross-departmental committee holds a release-planning meeting during which the backlog items to be implemented in the scheduled major release are selected. In SCRUM teams, the software engineers implement those backlog items for the upcoming release. Each sprint implements a predefined set of backlog items with the goal of implementing all assigned backlog items while being accountable for their progress throughout the duration of the sprint. At any time, the progress of each team can be traced throughout its repeated sprints. This provides the basis for requirements and project management.

While bug fixing processes may be part of this process (see Issue Tracking – issues which warrant extensive redesign), the fixing of issues takes precedence and is done predominantly in maintenance releases and hot fixes. It is the declared goal that a user does not have to wait for a major release to see an issue fixed. Steady improvement of computational correctness and performance takes absolute precedence.

Software engineers usually work on different versions: the next scheduled major release, the next scheduled maintenance release, the currently developed actual release candidate and potentially on new features. In order to provide an environment in which this can be done conflict free, the ClaRa$^+$ development team applies a branching model to its codebase in the version control (see Version Control).

Version Control

The ClaRa$^+$ codebase is managed in the version control system (VCS) Git [2]. There are several main requirements which have to be met by software engineering:

1. Every single contribution has to be accounted for (author, date, codebase prior to the contribution)
2. Every single contribution has to be traceable
3. Every single contribution has to be retractable
4. Contributions made to the codebase during bug fixing processes have to be linked to the ticketed issue
5. Feature development must not interfere with general development
6. If a feature cannot be finished in time, the general development still has to be able to go into production if it is so decided
7. Development of the next major release must not interfere with development of the next maintenance release and vice versa
8. Every release candidate has to undergo intense testing and exclusive issue fixing. This pre-release cycle must not interfere with general development or development of the next maintenance release and vice versa
9. At any time, it has to be possible to ship a stable version of ClaRa$^+$
10. At any time, it has to be possible to develop and apply hot fixes to a stable version of ClaRa$^+$
11. At any time, it has to be possible to merge or otherwise apply contributions which target several working versions of ClaRa$^+$ between versions
12. Every merge or other cross-version contribution has to be accounted for
13. There always has to be a running backup of the ClaRa$^+$ codebase

In order to meet the above requirements, the ClaRa$^+$ development team utilizes Git as VCS and a branching model to implement the multiple versions of ClaRa$^+$ which from now on will be referred to as branches to keep with VCS nomenclature.

Git in and of itself provides the necessary framework to meet the most basic of the above requirements. Every engineer always has a full backup of the ClaRa$^+$ repository on his or her machine

and would be able to effectively restore the central repository if it was in some way corrupted. Furthermore, every contribution to the codebase is done in atomic commits of changes which identify the author, the date and the parent commit (i.e. the parent version). Every commit can always be retracted and traced throughout the history of the codebase.

Commits done by engineers in order to fix issues reference tickets in the issue tracker and are automatically attached to the ticket and therefore traceable and reviewable from the issue tracker.

In order to meet the demands regarding the parallel development of different releases, a branching model is used. The branching model is loosely based on the GitFlow [3] model. This allows for independent development of major and maintenance releases. In order to minimize risk while enabling innovation, new features are also implemented separately. It also allows for immediate reaction to issue reports. Throughout the development of major and maintenance releases, risk assessment and risk control processes are continuously applied and enabled through this model.

This model allows engineers to meet all the above requirements regarding the development process. The process is monitored in the repository.

The main development branches are continuously integrated with the test environment. Feature branches are individually integrated with the test environment by the respective engineers.

Customer Support

ClaRa$^+$ licenses are usually sold with an appropriate support agreement. XRG Simulation GmbH and TLK Thermo GmbH employ support managers as first contact to customers. The support is available through the Dassault Systèmes support center or via E-Mail to LTX Simulation GmbH.

Support requests may trigger internal issue fixing processes. If a customer reports a malfunction, the created ticket is then transferred and properly cross-referenced into the internal issue tracker. While the current supporter provides the customer with a workaround, he or she will also trigger the bug fixing process. Depending on the severity of the issue and the risk and cost associated with providing a fix, the process may result in a hotfix, a fix in the next maintenance release (the most common case) or a redesign effort undertaken in the next major release. As described in Issue Tracking and Continuous Integration, a combination of those three outcomes might also apply.

[1] Wikipedia: “Scrum (software development)” [online]. Available: https://en.wikipedia.org/wiki/Scrum_(software_development), [Accessed June,1st 2018].

[2] L. Torvalds, J. Hamano and others: “Git” [online]. Available: https://git-scm.com/. [Accessed August, 16th 2016].

[3] V. Driessen: “A successful Git branching model”, January, 5th 2010. [online]. Available: http://nvie.com/posts/a-successful-git-branching-model/. [Accessed June, 1st 2016].