TAV - Report

Project: Luminosus

Johannes Schneider and Tim Henning

31. Oktober 2017

1 Milestone 0

1.1 Introduction

Extending the functionality of a lighting desk can be a difficult and time consuming task. To make it more easy to integrate and test new features the idea was to create a separate modular software platform that is connected via network to the lighting console. As a solution for this "Luminosus"was developed.

The main project development was done by Tim Henning while writing his bachelor thesis. He is the only developer of the project till now. For this V&V project Johannes Schneider joins the project as a developer and tester.

The programming language used is C++ 11 with syntax additions by Qt. The user interface part uses the descriptive language QML that includes function definitions in JavaScript.

The application is intended to be used by end users. It runs as a desktop application on Windows and MacOS and as an app on Android and iOS.

The existing code base consists of ca. 40k LOC.

Available artifacts are the source code, documentation, changelog, issue tracker, manual and in addition the Bachelor thesis *Entwicklung einer modularen Benutzeroberfläche als zusätzliche Bedieneinheit einer Lichtkonsole* including a requirement analysis and a discussion of the architectural decisions.

No specific development paradigm was used.

Manual tests are performed and static code analysis is provided by the IDE.

1.2 Learning from failures

Issue #2 Connection to Eos stopped working: After a while the network connection of the software to the lighting stopped working. The user was not able to continue using the application for the show even after restarting everything. While the fault is not yet known the issue is probably complexity related. Many different devices were involved and it is not clear if the problem was caused by this software a different failing part in the system.

Issue #3 Faders not updated: The user described an issue where the fader position were not updated correctly after switching the page. This issues can be categorized as preventable as the fault was a single missing line of code that could have been found by unit tests that cover all branches of the code.

Issue #4 Fullscreen on iOS not working: The issues states that the fullscreen button is not working correctly on the iOS platform. The failure is visible as too large UI elements that render the application unusable. On the one hand the fault can be categorized as preventable since it could be detected by a simple unit test that compares the scale property before and after the operation. On the other hand it is also a type of intelligent fault since the real problem was the pure existence of a fullscreen button on a mobile operating system.

Conclusion In general the introduction of unit tests would be a valuable addition to the existing code base. Furthermore integration tests for typical user scenarios would help to prevent some kind of faults. Last but not least for maintaining so many different platforms the user feedback is an important part in the process of V&V.

1.3 Five Basic V&V Questions

When do V&V start? When are they complete? Validation started right in the beginning. The project was validated on a daily basis due to a lot of contact to the supervisor. The specification was adjusted very frequently to ensure that the features that were developed were suitable for the projects goal.

Verification began early, too, in the form of static code analysis provided by the IDE (Qt Creator) and CLang. The development of tests did not start till the begin of this V&V project.

The process of V&V does not end till the end of the lifetime of the product. In this case as long as the software is maintained.

What particular techniques should be applied during development? Static code analysis is very good applicable for C++ due to strict type declarations.

Continuous Integration should be applied to the project since it will be used on various platforms. Furthermore automated unit testing will be applied during the semester.

Additionally profiling the memory consumption can help finding memory leaks and performance benchmarks make sure that the specification is met in terms of latency and responsibility.

In the end a formal verification of standardized components such as the network protocol implementations should be applied.

How can we asses the readiness of the product? The readiness is assessed by using the defined features of the specification and their current implementation status. Before a release is considered ready it should be at least tested manually in a typical use case scenario.

Furthermore open issues are a hint that the product is not yet ready.

How can we control the quality of successive releases? To ensure that successive releases do not introduce new faults a pre-release policy can be established to let a small group of beta users test the system before it is released to the public.

How can the development process itself be improved? Increasing emphasis on applying various V&V techniques during further development will help to improve overall code quality.

Furthermore the development process can be improved by introducing a second developer to the project. Thus pair programming and code reviews can be established to help finding flaws. In addition maintaining the documentation will lead to a better process.