Open-Source Components in Safety Critical Systems

S. Suomalainen 50575U, Mipro Oy, Kunnanmäki 9, 50100 Mikkeli

Abstract—Faults in control software of the airplanes have caused airplanes to crash and many people got killed. These vehicles are driven by systems that are called safety critical. Nowadays safety critical systems are done with programmable logic which requires software. It is easy to make software complex and hard to verify. Software might have features and functions that could lead to accident or incident. This is the reason why there are strict guidelines how to implement and verify safety critical software. The quality assurance is well defined. There are only few suppliers of the components of the safety critical system. The prices are high and systems are closed. Open-source software could replace some of these highly priced software or even components of them. Open-source software is said to be high quality and reliability is good. This paper explores requirements for safety critical software and quality assurance in open-source software. The paper presents examples of projects where open-source components are used in safety critical system. Linux was the only open source software that was found to be used as safety critical software. In conclusions IEC 61508 requirements are compared to quality assurance of open-source development and suggestions for using open-source components in safety critical system are given. Open source project do not have enough specified quality assurance practices to fulfill IEC 61508 requirements. The worst is the lack of documenting in open source development.

Index Terms—Safety critical software, open source, quality

1. INTRODUCTION

Safety critical system is an automation system which protects from loss of human lives, environmental disasters or economical losses. Safety critical systems are used for example in airplanes, train control and in traditional industry. The use of software-based systems is increasing dramatically (Storey 1996). The fail of safety critical system might cause the death of several people. The hardware of safety critical system is done redundantly and voting system is used. Failure in hardware causes system to go fail safe state. Fail safe state could be for example controlled shut down. Typically if accident or incident occurs it is due to software. There are standards and guidelines how safety critical software should be developed and how quality assurance should be done (IEC 1998; Leveson 1995).

Automation systems have been traditionally closed systems in a way that only one supplier can provide the whole system. There have been many contributions to make automation systems more open for example Olé for Process Control, OPC (OPC Foundation 2004). OPC is specified interface between automation systems. Still development environment for automation system is unique for every supplier. On the other hand software for automation system is quite simple and should not contain errors. The functional quality should be good and functional requirements are tested well. Input variables may cause unspecified combination and this combination results unspecified output which is not noticed in verification routines.

The rise of the open source development has given new competition for the conventional software industry. Open source software has been used in several applications in IT sector. There are some studies about quality in open source development and they have given promising results about the quality of the open source software (Zhao & Elbaum 2003).

1.1 Motivation

Components of automation systems are expensive because of the few suppliers. The operating systems are vendor specific software. The application software, that is used to build the applications, is unique for certain platform and it is almost impossible to add any features to them. The suppliers of the hardware and the software for the safety critical system are very limited. There is little competition and this makes prices very high and systems are closed.

The open source development could bring prices down and development environments could be more open.

1.2 Research problem and objectives

The objective of this research is to explore quality assurance in open source projects. This quality assurance is compared to the requirements of the IEC 61508. The examples of the projects where open source is used in the safety critical system are presented and lesson learned are collected.

The questions for this paper are related to the software of the safety critical system and the quality of the open source development. Can components of the safety critical system be open source code? How is quality assurance handled in the open source development? What are the quality requirements for the safety critical system? Examples of the use of an open source code in a safety critical application? How quality is handled in these projects?

1.3 Method

The methods used in this research are literature study to
other research papers that discusses open source development and especially the quality attributes. Quality assurance required for the safety critical software will be gathered from different standards, guidelines and books. The examples of open source components in safety critical system are collected, quality assurance in these projects is presented if it is available and author’s opinion of quality is presented.

The researches are collected from IEEE databases and INSPEC databases. Some examples are found through Internet.

1.4 Scope
There are also other aspects for safety critical software. Standards and guidelines define certain project management practices, tools, requirement specification, risk analysis and documents. These aspects would need more extensive research.

One important part of safety critical system is also underlying hardware. The paper will not describe hardware; it will concentrate on software.

This paper will only concentrate on quality assurance and if it is sufficient in open source. One part of the quality assurance is testing which combines both verification and validation. The paper concentrates on verification. In development demands for the quality assurance of the safety critical software will be focused on European standards (Storey 1996) and guidelines.

1.5 Structure of the paper
The paper begins from introducing quality requirements for the safety critical software according to the IEC 61508. Chapter 3 introduces quality assurance in open source development which is gathered from different researches. Then in chapter 4 examples of the safety critical systems, where open source components are used, are introduced. Finally conclusions summarizing results are presented.

2. QUALITY ASSURANCE REQUIREMENTS FOR SAFETY CRITICAL SOFTWARE

This chapter presents quality activities required by IEC 61508 using standard itself and also guidelines and books for interpreting them. There are other standards (IEC 61511, EN 50159) that have requirements for safety critical software but IEC 61508 is the main standard for safety related systems in Europe where other standards are based on.

Quality management can be divided into quality assurance and quality control. In the quality assurance of the safety critical software main aim is to increase the quality of the product, provide a foundation for the safety justification and to improve development and production efficiency (Storey 1996). Quality control aims to define techniques and activities to fulfill requirements for quality (Storey 1996).

In safety critical software quality plays big role. It is not allowed to have serious quality problems that affect on safety. Then again quality problems that are important for other software like usability are allowed if they don’t affect on safety. In many books (Leveson 1995; Storey 1996) concerning safety critical software it is not mentioned quality or testing for quality. Testing and other quality management procedures are done for safety. Safety is one property of quality and it should be built on software (Leveson 1995). The core of the safety critical software is the safety functions.

Testing activities requires that software is tested against requirements, test plans and test procedures are documented, any function has to be tested and dead code must be removed (Locke 2003). After modifications all the tests must be done again for the part that is affected.

Important concept in safety critical system is safety integrity. Safety integrity is the likelihood of a safety related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time (Storey 1996). Safety integrity level depends on severity of hazardous event and frequency of hazardous event. These two attributes gives risk classification. Risk classification leads to safety integrity classification. Software safety integrity level can be achieved from safety integrity classification. In table 1 likelihoods for failure rates for each safety integrity level are presented. For every safety critical system in Europe safety integrity level has to be defined and every part of the system (hardware and software) has to fulfill this safety integrity level.

2.1 Quality management according to IEC 61508

IEC 61508 requires for any software that is used in safety critical software to have functional safety planning which defines strategy for software procurement (IEC 1998), Software lifecycle consists following parts and for every part procurement should be defined (IEC 61508):

- development
- integration
- verification
- validation
- modification

IEC 61508 defines also safety critical software configuration management procurements. Software configuration management should (IEC 1998)

- apply administrative and technical controls
2.2 Testing according to IEC 61508

Testing in IEC 61508 is divided into three; module testing, software integration testing and system integration testing for software and hardware. All these activities are verification activities (Järvi 2004). Testing includes both verification and validation and IEC 61508 gives requirements for both. This article concentrates only on verification.

Module testing is an evaluation of small, simple functions. Faults are easier to locate and much cheaper to correct than in later phases. Module testing requires (IEC 1998; Järvi 2004)

- tests shall be specified during design
- test shall show that module does intended functions and not any unintended ones
- results shall be documented
- corrective actions shall be specified

Software integration testing should investigate the characteristics of the whole software. Software integration testing requires (IEC 1998; Järvi 2004)

- test shall be specified during the design and development phase
- tests shall specify division of the software, test cases and data, types of tests to be performed, test environment, tools, configuration and programs, test criteria, procedures for corrective action
- software shall be tested according to test plans
- test results shall be documented
- any modifications or changes shall be defined where changes impact

Integration testing means system integration testing which evaluates whole system including software and hardware. Integration tests requires (IEC 1998; Järvi 2004)

- tests shall be specified during the design and development phase
- test shall specify division of system, test cases and data, types of tests to be performed, test environment including tools, support software and configuration description
- test criteria
- any modifications or changes shall be defined where changes impact
- tests shall be documented

Faults in software integration testing and system integration testing are very hard to identify and correct. It is very expensive to make modifications and changes in integration phase. Nevertheless it is better to find faults in integration phase than in operation phase. If fault occurs in operation phase, usually a deviation is done for the local administrator of accidents thought fault would have only caused incident.

If software is modified during operation or maintenance for one reason or another following actions need to be done (IEC 1998; Järvi 2004)

- analysis of the impacts of the changes
- safety plan for modifications and verification
- plans for modifications and verification
- environmental affects on modifications and verifications
- updates for documents
- all the details from systems shall be available

It should also be noted here that impacts of non-safety software for safety critical software should be defined and verified. Non-safety software shouldn’t impact on safety functions and if it does impact, protection against failure situations shall be specified (Järvi 2004).

After every development phase verification should be done. Verification should show that phase is concluded successfully.

2.3 Dynamic and static testing and modeling

Dynamic testing investigates system and its components by executing them. In dynamic testing several test cases are executed. A test case defines test data (inputs and outputs), pre-conditions and post-conditions. Dynamic testing is divided into three main categories (Storey 1996):

- functional testing investigates all the functions that are defined within its requirements
- structural testing uses detailed knowledge of its internal structure to investigate its characteristics
- random testing investigates randomly different inputs and occurring outputs

Static testing investigates system and its components without operating them for example by reviewing. Some of the testing is performed manually but there are also automated tools for doing static testing (static code analysis tool).

In modeling mathematical representation of the behavior of the system or its environment is used. Modeling is normally used in early stage of system design but it can also used in late phases. Modeling can also be graphical presentation of the system.

2.4 Testing methods and techniques

In black-box testing tester doesn’t have any idea of the functions inside program. The tester only tests application against test plans. This is also called requirements-based testing (Storey 1996). In dynamic testing functional and random testing are examples of black-box tests.

In white-box testing tester has access to source code and tester understands its behavior. This testing method is combination of dynamic and static testing. Structural testing in dynamic testing is a white-box technique.

IEC 61508 defines testing techniques for dynamic (both black-box and white-box) and black-box testing that are ____________
recommended in each safety integrity level in table II and table III (IEC 1998). The abbreviations NR, R and HR reference to Not Recommended, Recommended and Highly Recommended. If some technique is highly recommended then it has to be used in verification.

Storey (Storey 1996) gives a table V of which technique to use in each development stage. Table divides development into requirements analysis and functional specification, top-level design, design reviews, implementation, integration testing and validation. As can be seen verification should be started from the requirements but most activities are done after top-level design. Implementation and integration testing has same techniques which are also mentioned in table II and table III.

2.5 Other quality assurance aspects

Important aspects are also traceability, test adequacy and coverage. There are some guidelines which strategies to use but not so clear as different testing techniques. Traceability adds probability that software contains code that will be executed under unusual conditions and boundary conditions are examined (Locke 2003).

It is argued (McDermid 2001) how does the IEC 61508 prove that software is safe. The IEC 61508 gives different techniques that should be done for each safety integrity level. Safety integrity level defines how reliable software is but it doesn’t directly measure software safety or give any evidence that software is safe. McDermid (McDermid 2000) suggests evidence based approach for testing.

Organizational aspect in testing is testing organization. Table IV shows how testing should be handled in different safety integrity level software (Storey 1996). As can be seen independent department or organization is recommended almost for any safety integrity level software.

2.6 Example of safety critical software verification

Meyer (Meyer 1996) gives an example of safety critical software and what techniques are used. Article argues that formal configuration management, static code analysis and formal walk-through were not very useful and didn’t add any quality or safety. Good practices had been iterative process model with bench-marking, extensive black-box testing, scenario and limited integration testing. Important practice is giving plans and prototypes to customer as soon as possible. This way software is tested in early stage in real environment and improvements are given in development phase.

2.7 Summary of IEC 61508 requirements

IEC 61508 gives lot of requirements for software development. Here are summarized main points from the previous chapter.

Quality management requires safety plan for the whole lifecycle of the software. Plan must define configuration tools, change control methods, configuration and release status and how safety integrity level is demonstrated. Plan should also define impacts of non-safety functions to safety functions. Tests for the each part of the software must be defined in early phases in design. Tests must show that functions do what they are supposed to do. All the results must be documented and corrective actions must be defined. Test plans must contain test cases, test data and test pass/fail criteria. Test tools and environment must be defined and impacts of changes must be defined.

IEC 61508 and guidelines gives lot of techniques that should be used. Techniques are divided to dynamic testing, static analysis and modeling. Depending on safety integrity level different techniques are recommended to use. Most mentioned techniques are boundary value analysis,
performance modeling, functional testing, walk-through, design reviews and environmental modeling.
Depending on safety integrity level different independences on testing are required. Higher safety integrity level is more independent tester must be from the development organization.

As can be seen from chapter 2.6 not all the techniques and plans are very useful. More agile methods are preferred and more customer participation is suggested.

### 3. QUALITY ASSURANCE IN OPEN SOURCE SOFTWARE

The chapter is gathered from the several researches of open source development. The chapter will represent the most used techniques and basic principles of the quality assurance in open source development.

Quality assurance techniques in open source development are not consistent, different projects use different techniques. Other surveys shows that open source software have higher quality and better functionality than commercial software (Stark 2000; Graham 2000). Another research claims that

<p>| TABLE V | TESTING TECHNIQUES IN SAFETY CRITICAL SOFTWARE DEVELOPMENT STAGES |</p>
<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
<th>Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirements analysis and functional specification</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walkthroughs</td>
<td>Formal methods</td>
<td></td>
</tr>
<tr>
<td>Design reviews</td>
<td>Software prototyping / animation</td>
<td></td>
</tr>
<tr>
<td>Checklists</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Top-level design</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walkthroughs</td>
<td>Performance modeling</td>
<td></td>
</tr>
<tr>
<td>Design reviews</td>
<td>State transition diagrams</td>
<td></td>
</tr>
<tr>
<td>Checklists</td>
<td>Time Petri nets</td>
<td></td>
</tr>
<tr>
<td>Formal proofs</td>
<td>Data flow diagrams</td>
<td></td>
</tr>
<tr>
<td>Fagan inspection</td>
<td>Structure diagrams</td>
<td></td>
</tr>
<tr>
<td>Detailed design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walkthroughs</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Design reviews</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control flow analysis</td>
<td>Formal methods</td>
<td></td>
</tr>
<tr>
<td>Data flow analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbolic execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checklists</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fagan inspection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metrics</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static analysis</td>
<td>Functional testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Boundary value testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structure-based testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Probabilistic testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error guessing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process simulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error seeding</td>
<td></td>
</tr>
<tr>
<td><strong>Integration testing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walkthroughs</td>
<td>Functional testing</td>
<td>Enviromental modeling</td>
</tr>
<tr>
<td>Design reviews</td>
<td>Timing and memory tests</td>
<td></td>
</tr>
<tr>
<td>Sneak circuit analysis</td>
<td>Boundary value testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stress testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Probabilistic testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error guessing</td>
<td></td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional testing</td>
<td>Enviromental modeling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software prototyping / animation</td>
<td></td>
</tr>
</tbody>
</table>
open source products are not more dependable than commercial software (Lawrie & Gacek 2002). It is also said that open source development reduce cycle-time, decrease implementation and quality costs (Graham 2000). Open source development is highly iterative process which uses some of the same practices as agile methods.

This paper doesn’t respond which one commercial or open source software has higher quality. The paper only presents techniques used in open source project and what is required from safety critical software.

3.1 Testing

Zhao et al. (Zhao & Elbaum 1999; Zhao & Elbaum 2003) are examining testing in open source projects. This chapter is mostly based on their results.

Results show that 80% of the survey open source projects don’t have testing plans. A technique, that open source use for most of projects, is simulation. Simulation means that inputs are trying to imitate use behavior. Fourth of the survey projects test extreme values and assertions. Half of the projects use baseline test suite which enables regression testing.

As a testing tool most common is debugger. Not even half use testing tools and it is more common in big projects than in small ones. Also there is a correlation between the age of language and testing tools. The older language is more tools it has. Exception is java which has many testing tools.

Half of the projects don’t take advantage of the coverage concept. Small projects have longer testing time than big ones. Seidel et al. (Seidel & Niedermeier 2003) argues that open source projects test application in many different platforms and environments.

Comparing to safety critical software requirements only few projects fulfill the requirement of early test plans. Testing techniques (simulation, boundary values, regression testing) could be enough for safety critical software but they and testing environment should be defined. IEC 61508 requires that all the test results are documented. This is not mentioned in researches.

3.2 Inspections and reviews

Inspections and reviews are activities where someone other than the author of the artifact reads through the artifact with the purpose of finding errors, omissions or flaws and making suggestions and recommendations to the author. In open source projects inspections are mostly used for source code. Other areas where inspections are done are requirements and design.

Inspection and reviews are used in big projects and its importance is scaled to 3 in 1-5 scaling system (Zhao & Elbaum 2000). Mainly inspections are done in development phase. It is thought that anyone who downloads application will check source code (Zhao & Elbaum 2000) which is not very convincing. From reference (Zhao & Elbaum 2000) it could be understood that inspections and reviews are not very commonly used but then reference (Stark 2000) states that 50% of projects use it.

Inspections and reviews are suitable for open source project because of the organizational culture of the projects. Many developers in open source projects do reviews without management encouragement.

Static analysis in safety critical software is required for different development layers. Static testing can be reviews, inspections, checklists etc. If used in open source development static analysis seems to be enough for safety critical software if all the results are documented.

3.3 User participation

In open source projects users validate and verify software. Developers do not use testing techniques or tools. The role of the user is also to give suggestions as project requirement. According to (Zhao & Elbaum 2003) user participation is high, in big projects user act as bug trackers and in small projects users give suggestions for project.

User role in testing is in field-tests and in reviews. Users find hard faults normally in medium and big size projects. Also more mature projects are more faults are found by user (Zhao & Elbaum 2003).

In small projects user generate 20% of the changes and developers think that suggestions are normally reasonable (Zhao & Elbaum 2003).

The standards and guidelines recommend that safety critical software should have independent tester or even better independent testing organization from development organization. This is handled well in open source development where users do the testing.

User participation to change suggestions is not handled in IEC 61508.

3.4 Defects

Defects are handled in reference Zhao et al. (Zhao & Elbaum 2000 & Zhao & Elbaum 2003). According to Zhao et al. (Zhao & Elbaum 2003) 61% of survived projects has debugging tool. Most of the defects are found from the user interface. Second is software logic. Especially C++ has logical defects (Zhao & Elbaum 2000).

Defect and change handling should be considered in safety plan which could be also project plan. All the testing results should be documented. Problem is that in debugging tool only defects are documented.

3.5 Configuration

Configuration management is normally well handled in open source projects because developers and users are all over the world. Globalization requires sophisticated configuration management and tools. This is shown in Zhao et al. (Zhao & Elbaum 2003) which states that 89% of the survived projects used CVS tool.

Difference between ordinary software project and open source project is change management. In an ordinary software project change is first reviewed and then it is coded and new code is put to configuration management tool. In open source
projects change is first done, code is taken from to the CVS which is reviewed by the reviewer and the code is put back to CVS if reviewer has accepted change (Stark 2000). So in open source project there could be many changes that are done but never accepted.

Configuration management and tools are well specified in open source development and these will fulfill safety critical software requirements. Change control should be more documented and managed. IEC 61508 requires that all the change impacts have to be specified and documented.

3.6 Documentation

As expected documentation is not so important aspect in open source projects. Most of the projects use some kind of To-Do lists (Zhao & Elbaum 2003). Some of the projects do also installation and building guidelines, design documents and few (normally big projects) use release plans.

Documentation is the most problematic aspect in using open source software for safety critical system. Safety critical software has to be very well documented. All the design documents, test plans, results, release plans etc. have to be specified.

3.7 Organizational issues

Preconditions for good quality software that is open source are that there is an existing software artifact and talented coordinator or integrator (Graham 2000). Open source development is much more human centric development model (Lawrie & Gacek 2002) which is depended on tools and experience of developers.

Generally open source developers are committed to quality (Stark 2000), which requires internalization of quality management. This means that developers accept ideas as they were their own.

Open source project is highly organized (Lawrie & Gacek 2002) but still they don’t have strong leader. Strong leader could affect on motivation for example encourage doing reviews. But then because people usually work as volunteers in open source project this could affect on whole motivation. Though leader is not strong, leader is respected on decisions (Stark 2000).

3.8 Summary of quality assurance in open source development

IEC 61508 requires safety plan and it is not mentioned in open source researches whether projects use project plan or not. Configuration management and tools are defined in most of the projects. The release or other documents are not used in most of the open source projects.

The problem is that open source project do not use test plans that define the wanted behavior. The tester and developers have their own ideas and tests are done according to everyone’s own ideas. Test results are only documented if there is a defect in software. Test tools and environment varies in different open source projects.

There are several of testing techniques and environments in open source projects. Intention of these tests is to show that software works as it is wanted to. Most used techniques in open source development are simulation and reviews.

Testing independence is well handled in open source development where user is the main tester.

Organizational issues like motivation are not handled in IEC 61508. As shown the developers of the open source projects have high motivation which results better quality.

4. Examples of using open source components in safety critical system

This chapter will first introduce briefly what sort of safety critical system open source software has been used. Then the most used open source software Linux is introduced from its quality assurance point of view. Then example of what practices NASA is used for using open source software is described.

Most used open source software in safety critical application is Linux (Klounikas & Yovine 2003; Broggi & Fascoli 1999; Ortega 1998). Linux was the only open source software that was found to be used affecting directly safety critical functions. Both Linux and Linux real-time are used. In these cases Linux has been operating system for some safety critical system. Linux has been used in hundreds of thousands of complex environments for almost 10 years and it has been evaluated by thousands of developers (Locke 2003).

Other open source software has been used for tools modeling safety critical system (Philippi 2003) and for user interfaces (Norris 2004).

4.1 Linux in safety critical applications

Healthy and Safety Executive, HSE (Pierce 2002) has done preliminary assessment of Linux for safety critical systems. Their suggestion is that Linux can be used in safety integrity level 1 and 2. Safety integrity level 3 can achieved from certification.

Criteria that operating system can be used in safety critical system are that operating system must be sufficiently well understood, it must have suitable characteristics for safety critical application and it must be sufficiently reliable. Features for applications, where Linux has been used, are that applications have relatively modest timing requirements, the provision of high-resolution screen displays and need for disc file support.

Linux has been used in several safety critical applications
  • Automatic train control display system
  • railway control system
  • SCADA system for railway
  • process plant displays and control systems
  • embedded systems

4.1.1 Quality practices in Linux project

Linux uses for configuration management CVS – repositories. Changes go through reviewer which accepts changes.

Linux uses user reporting bug systems. Bugs are fixed
quickly.

Documentation is done volunteer work. Linux has how-to, mini how-to and FAQs. Drawback is that Linux doesn’t have one single definition.

The need for Linux to be safety integrity level 3 is that it needs through rough documented testing and analysis.

4.1.2 Linux in autonomous vehicles

Linux is running in safety critical software which is intended to autonomously drive vehicles under environmental conditions and sudden danger. Requirements for operating system are real-time response, smart algorithms, powerful computing engines and low price. System was tested in test journey where 2000 kilometers were driven in different landscape. Every data was put in to a log file. Linux and the whole system had been extremely reliable, there were no faults. (Broggi & Fascioli 1999)

4.1.3 Linux in international space program

In this project Linux controls rendezvous and docking operations for a servicing spacecraft. Application is not exactly safety application but mission critical which has same requirements that safety application has. Application for spacecraft must do complex mathematical equations and is must be highly fault tolerant. Linux was chosen because its reliability, performance, portability and affordability. (Ortega 1998)

4.2 Practices for using open source software

NASA uses open source components in some applications. NASA has used in one mission-critical application eight open source components (Castor, Java Expression Parser, Xerces-J, MySQL, MySQL Connector, HSQL Database Engine, Virtual Reality Modeling Language and Skaringa). They have found some good experiences on open source software. Software has had high quality, developers have been highly motivated and included detailed unit- and component level tests. It has also taken less time when fault has been found to correct it than with commercial software suppliers.

NASA has found three factors that have to be investigated when finding open source software (Norris 2004):

- project’s maturity
- project’s longevity
- flexibility

Maturity depends on how long project has existed. Older project is more mature is. More mature project better practices it has. Maturity correlated also to the reliability of the software.

Longevity is also depending on how long project has existed. Longer project has been exist more active are developers and users.

Flexibility gives opportunity to make changes and improvements to software. Flexibility means that do open source developers do changes that are asked and that the developers are willing to be part of the project.

Every part of the open source software was part of the NASA’s test plan. Open source components had automated unit and component level tests. NASA took quite much care of quality assurance or at least lead the way.

An example of using open source software development in process industry has gathered success factor for good quality in open source software project (Seidel & Niedermeier 2003). Successful open source project need:

- support from management
- clarification of legal aspects
- suitable communication structure / infrastructure
- skilled and self-motivated developers
- some key highly knowledgeable major developers
- two-tier management structure
- light-weight processes
- relaxed time-tables
- fun and pride

4.3 Summary

The only found open source software that is used as actual safety critical software is Linux. Other open source software is used in applications which are related to safety critical system like modeling tool and user interface.

Linux has not used (or not mentioned) safety or project plan which would have specified the whole lifecycle of the software. Linux project defines configuration tools and change control. Project does not have specified test plans and defects are gathered to debugging tool but not other test results are documented. It is not mentioned what testing techniques are used and who does testing. The most problematic in Linux is also documents which do not satisfy safety critical software requirements.

NASA gives good guidelines to have project management in own organization. That means open source software is tested in NASA and NASA gives guidelines to open source project how to do quality assurance.

In next chapter all the results are gathered.

5. Conclusions

Though experiences did not give much of details of quality assurance of open source components it can be seen that own test plans and verification routines has to be used. Open source components can be used but user has to be very careful from which open source project it takes software and are developers willing to make changes.

There are many collisions between IEC 61508 and open source development. IEC 61508 requires test plans already at the design stage. Open source projects ordinary do not even have any test plans. According to IEC 61508 test results have to be documented which was not mentioned in open source studies. But if there are no test plans then there might not be test results. Faults are reported in open source projects in debugging databases which doesn’t documents working part of the code. The need for traceability and test coverage is not fulfilled in open source development even though open source software is tested well in different platforms and environments.
The most problematic practice in open source development is documenting or better lack of documenting. IEC 61508 and all the other safety standards require lot of documenting; tests, guidelines, manuals and design. For example Linux can not be used in higher safety integrity levels because of its lack of documents and analysis. If all tests would be documented and some one would track the results test documenting requirements would be fulfilled.

Change management in open source project is done wrong way compared to IEC 61508. The standard requires impact analysis which is not done. There is also a lack of regression testing.

Testing in open source is concentrated mostly on functional (simulation) testing. Some static testing (reviews and inspection) is done. Static testing is done mostly in development phase. IEC 61508 requires static analysis and other testing for designs. IEC 61508 recommends many testing techniques which are not used in open source projects like boundary analysis and structure analysis. Testing techniques do not fulfill higher safety integrity levels. In open source projects users are testers which make tests independent. This is noticed to be good practice according to (Meyer 1996).

Good practices and techniques in open source development are configuration management and motivation. Configuration management is handled very well in most of the projects. Most open source projects use CVS –tool. Open source developers are mostly volunteers and they have high motivation for project and for quality of the software. This is not necessary case in safety critical project. Developers in safety critical project are not so motivated and connected to project. They might not have the big picture of the project because they only do little part of the project (Leveson 1995). As a conclusions could be said that open source components can not be used as such in safety critical system. Open source quality assurance practices are not enough for standards defining safety critical system requirements. If considering of using open source components in safety critical system the developer of safety critical system should careful check the state of the open source project. How motivated developers in open source project are? It should also be checked what sort of verification and configuration routines open source project has. Can the developer of the safety critical system affect on practices and tools used in open source project?

The developer of safety critical system has to define test plans for open source components and verification has to done and documented. Good thing is that source code of open source software is available and that way developer of safety critical system can review and do inspections for source code. Better would be of course if developers in open source project would be so motivated also to the safety critical system that they would do unit and integration tests. Then safety critical system developer could do system integration tests. All the changes to open source software should go through the safety critical system developer so that he could do analysis on change impacts to other system.

Even though open source software is said to be reliable it does not make software safe. Fuggetta has written (Fuggetta 2003) “Software can be inspected to check whether it is compliant to security and safety requirements”. This can be used in this context by summarizing the point. Open source software can be used but it should be tested and analyzed very carefully when using it in safety critical application.

REFERENCE LIST


