"Open Proof" for Railway Safety Software

A Potential Way-Out of Vendor Lock-in Advancing to Standardization, Transparency, and Software Security.

Klaus-Rüdiger Hase
Deutsche Bahn AG
Richelstrasse 3, 80634 München, Germany
Klaus-Ruediger.Hase {at} DeutscheBahn {dot} com

Abstract

"Open Proof (OP) is a new approach for safety and security critical systems and a further development of the Open Source Software (OSS) movement, not just applying OSS licensing concepts to the final software products itself, but also to the entire life cycle and all software components involved, including tools, documentation for specification, verification, implementation, maintenance and in particular including safety case documents. A potential field of applying OP could be the European Train Control System (ETCS) the new signaling and Automatic Train Protection (ATP) system to replace some 20 national legacy signaling systems in all over the European Union. The OP approach might help manufacturers, train operators, infrastructure managers as well as safety authorities alike to eventually reach the ambitious goal of an unified fully interoperable and still affordable European Train Control and Signaling System, facilitating fast and reliable cross-border rail traffic at state of the art safety and security levels.

Keywords: ATC, ATP, Critical Software, Embedded Control, ETCS, EUPL, FLOSS, Open Proof, openETCS, Train Control, Standardization.

1 Introduction

The European Train Control System (ETCS, [1]) is intended to replace several national legacy signaling and train control systems all across Europe. The system consists of facilities in infrastructure and on-board units (OBU). Especially for the ETCS on-board equipment the degree of functional complexity to be implemented is expected to be significantly higher than in conventional systems. In terms of technology, this is mostly done by software in so-called embedded control system implementations. While electronic hardware is getting continuously cheaper, the high complexity of the safety critical software has caused significant cost increases for development, homologation and maintenance of this technology. This has raised questions for many railway operators with respect to the economy of ETCS in general.

The key element for improving that situation seems to be a greater degree of standardization in particular for the ETCS onboard equipment on various levels: Hardware, software, methods and tools. Standardization by applying open source licensing concepts will be the focus of this paper.

1.1 From National Diversity to European Standard

Looking back into history of signaling and automatic train protection (ATP) for mainline railways systems, in the past 40 years a major change in technology has taken place. In the early days of ATP almost all functions were implemented in hardware, starting with pure mechanical systems, advancing to electromechanical components and later on using solid state electronics, like gates, amplifiers, and other discrete components. Software was not an issue then. Beginning in the late 1970 years an increasing number of functions were shifted into software, executed by so called micro computers. Today the actual functions of such devices are almost entirely determined by software. The dramatic performance increase of microcomputers in the past 30 years on the one hand and rising demand for more functionality on the other hand, has caused a significant increase in complexity of those embedded control systems - how such devices are usually called. Furthermore, the development from purely monitoring safety protection systems, like the German INDUSI (later called PZB: Punktfrmige Zug-Beeinflussung) or similar systems in other European countries, which only monitor speed at certain critical points and eventually stop the train, if the driver has missed a halt signal or has exceeded a safe speed level, to a more or less (semi) Automatic Train Control (ATC) systems like the German continuous train control system, called LZB (Linien-Zug-Beeinflussung), which has increasingly shifted safety responsibility from the infrastructure into the vehicle control units. Displaying signal commands inside the vehicle on certain computer screens, so called cab signaling, has resulted in greater independence from adverse weather conditions.

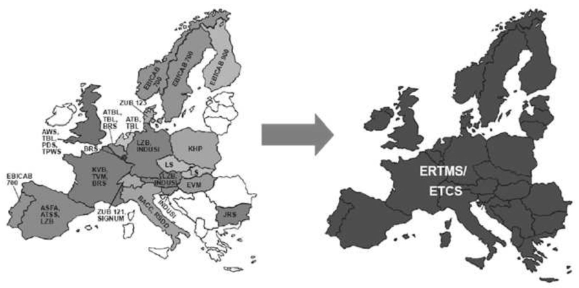


FIGURE 1: *Europes challenge is to substitute more than 20 signaling and ATP systems by just one single system, ETCS, in order to provide border crossing interstate rail transit in all over the European Union.*

In all over Europe there are more than 20 different mostly not compatible signaling and train protection systems in use (figure 1). For internationally operating high speed passenger trains or cargo locomotives up to 7 different sets of equipment have been installed, just to operate in three or four countries. Since each of those systems have their own antennas to sense signals coming from the way-side and their own data processing units and display interfaces, space limitations are making it simply impossible to equip a locomotive for operation in all EU railway networks, not to mention prohibitive cost figures for such equipment. Furthermore, some of the systems are in use for more than 70 years and may not meet todays expected safety level. Some are reaching their useful end of life causing obsolescence problems.

For a unified European rail system it is very costly to maintain this diversity of signaling systems forever and therefore the European Commission has set new rules by so called Technical Specifications for Interoperability (TSI) with the goal to implement a unified European Train Control System, which is part of the European Rail Traffic Management System (ERTMS), consisting of ETCS, GSM-R, a cab radio system based on the GSM public standard enhanced by certain rail specific extensions and the European Traffic Management Layer (ETML). Legacy ATP or so called Class B systems are supposed to be phased out within the next decades.

1.2 ETCS: A new Challenge for Europes Railways

Before launching the ETCS program, national operational rules for the railway operation were very closely linked with the technical design of the signal and train protection systems. That is going to change radically with ETCS. One single technology has to serve several different sets of operational rules and even safety philosophies.

The experience of Deutsche Bahn AG after German reunification has made very clear that it will take several years or even decades to harmonize operational rules in all over Europe. Even under nearly ideal conditions (one language, one national safety board and even within one single organization) it was a slow and laborious process to convert different rules and regulations back into one set of unified operational rules. After 40-years of separation into two independent railway organizations (Deutsche Reichsbahn in the east and Deutsche Bundesbahn, west), it took almost 15 years for Deutsche Bahn AG to get back to one single unified signaling handbook for the entire infrastructure of what is today DB Netz AG.

Therefore, it seem unrealistic to assume that there will be one set of operational rules for all ETCS lines in all over Europe any time soon (Which does not mean that these efforts should not be started as soon as possible, but without raising far too high expectations about when this will be accomplished.). That means, in order to achieve interoperability by using a single technical solution: This new system has to cope with various operational regimes for the foreseeable future. Beside this, for more than a decade there will be hundreds of transition points between track sections equipped with ETCS and sections with one of several different legacy systems. This will cause an additional increase of functional complexity for onboard devices.

1.3 Technology is not the Limiting Factor

With state of the art microcomputer technology, from a technological point of view, this degree of complexity will most likely not cause any performance problems since the enormous increase in performance of microcomputer technology in recent years can provide more than sufficient computing power and storage capacity at an acceptable cost level; to master complex algorithms and a huge amount of data.

The real limiting factor here is the human brain power. In the end it is the human mind, which has to specify these functions consistently and completely, then provide for correct designs, implement them and ultimately make sure that the system is fit for its purpose and can prove its safety and security. The tremendous increase in complexity, absorbing large numbers of engineering hours is one reason why we are observing cost figures for R&D, testing and homologation of the software components in various ETCS projects that have surpassed all other cost positions for hardware design, manufacturing and installation. This has caused a substantial cost increase for the new onboard equipment compared with legacy systems of similar functionality and performance.

Normally we would expect from any new technology a much better price to performance ratio than for the legacy technology to be replaced. Due to the fact, that this is obviously not the case for ETCS, makes it less attractive for those countries and infrastructure managers, who have already implemented a reliably performing and sufficiently safe signaling and train control system. In addition there is no improvement expected for ETCS with respect to performance and safety compared with service proven systems like LZB and PZB. In order to reach the goal of EU-wide interoperability soon, the EU Commission has implemented legal measures, regulating member states policies for infrastructure financing and vehicle homologation. While in the long run, ETCS can lower the cost for infrastructure operators, especially for Level 2 implementations making conventional line signals obsolete, the burden of cost increase stays with the vehicle owners and operators.

Therefore it became an important issue for vehicle operators to identify potential cost drivers and options for cost reduction measures, so as not to endanger the wellintentioned general goal of unrestricted interoperability.

2 Software in ETCS Vehicle Equipment

As discussed above, state of the art technology requires for almost all safety critical as well as non-safety related functions to be implemented in software. The end-user will normally receive this software not as a separate package, but integrated in his embedded control device. Therefore software is usually only provided in a format directly executable by the built-in microprocessor, a patter of ones and zeros, but therefore not well suited for humans to understand the algorithm. The original source code, a comprehensible documentation format of this software, which is used to generate the executable code by a compiler and all needed software maintenance tools are usually not made available to the users. Manufacturers are doing this, because they believe that they can protect their high R&D investment this way.

2.1 Impact of Closed Source Software

However concealment of the software source code documentation has increasingly been considered as problematic, not only for economical reasons for the users, but more and more for safety and security reasons as well. Economically it is unsatisfactory for the operators to remain completely dependent from the original equipment manufacturer (OEM), no matter whether software defects have to be fixed or functions to be adapted due to changing operational requirements. For all services linked to these embedded control systems there is no competitive market, since bundling of non-standard electronic hardware together with closed source or proprietary software makes it practically and legally impossible for third parties to provide such service. This keeps prices at high levels. While malfunctions and vulnerability of software products, allowing malware (malicious software: as there are viruses, trojans, worms etc.) to harm the system, can be considered as quality deficiencies, which can practically not be discovered in proprietary software by users or independent third parties, whereas the question of the vendor lock-in due to contractual restrictions and limiting license agreements is generally foreseeable, but due to generally accepted practices in this particular market, hardly to be influenced by individual customers (e.g. railway operators). Especially security vulnerability of software must be considered as a specific characteristic of proprietary or closed source software. So-called End User License Agreements (EULA) do usually not allow end-users to analyze copy or redistribute the software freely and legally. Even analysis and improvement of the software for the users own purposes is almost generally prohibited in most EULAs. While on the one hand customers who are playing by the rules are barred from analyzing and finding potential security gaps or hazardous software parts and therefore not being able to contribute to software improvements, even not for obvious defects, however the same legal restrictions on the other hand do not prevent bad guys from disassembling (a method of reverse-engineering) and analyzing the code by using freely available tools, in order to search for security gaps and occasionally (or better: mostly) being successful in finding unauthorized access points or so-called backdoors. Intentionally implemented backdoors by irregularly working programmers or just due to lax quality assurance enforcement or simply by mistake are causing serious threats in all software projects. In most cases intentionally implemented backdoors are hard to find with conventional review methods and testing procedures. In a typical proprietary R&D environment only limited resources are allocated in commercial projects for this type of security checks and therefore stay most likely undiscovered. That backdoors cannot be considered as a minor issue, has been discussed in various papers [2, 3, 4, 5] and has already been identified as a serious threat by the EU Parliament, which has initiated resolution A5-0264/2001 in the aftermath of the Echelon interception scandal, resulting in following recommendations [6]:

... Measures to encourage self-protection by citizens and firms:

29. Urges the Commission and Member States to devise appropriate measures to promote ... and ... above all to support projects aimed at developing **user-friendly open-source encryption software**;

30. Calls on the Commission and Member States to promote software projects whose **source text is made public (open-source software), as this is the only way of guaranteeing that no backdoors are built into programmes**;

31. Calls on the Commission to lay down a **standard for the level of security** of e-mail software packages, placing those packages **whose source code has not been made public in the least reliable category; ...**

This resolution was mainly targeting electronic communication with private or business related content, which most likely will not hurt people or endanger their lives. However a recent attack by the so called STUXNET worm [7], a new type of highly sophisticated and extremely aggressive malware, which in particular was targeting industrial process control systems via its tools chain, even in safety critical applications (chemical and nuclear facilities). Systems, which are very similar in terms of architecture and software design standards with signaling and interlocking control systems. This incident has demonstrated that we have to consider such impact in railway control and command systems as well, commercially and technically.

2.2 Software Quality Issues in ETCS Projects

Despite a relatively short track record of ETCS in revenue service we had already received reports on accidents caused by software defects, like the well documented derailment of cargo train No. 43647 on 16 October 2007 at the Ltschberg base line in Switzerland [8]. German Railways has been spared so far from software errors with severe consequences, possibly due to a relatively conservative migration strategy. During the past 40 years, software was only introduced very slowly in small incremental steps into safety-critical signaling and train protection systems and carefully monitored over years of operation, before rolled out in larger quantities. Software was more or less replacing hard-wired circuits with relatively low complexity based on well serviceproven functional requirement specifications over a period of four decades. With ETCS however, a relatively large step will be taken: Virtually all new vehicles have to be equipped with ETCS from 2015 on, enforced by European legal requirements, despite the fact that no long-term experience has been made. The ongoing development of the functional ETCS specification as well as project specific adaptations to national or line-specific conditions has resulted in numerous different versions of ETCS implementations not fully interoperable. Up to now, there is still no single ETCS onboard equipment on the market that could be used on all lines in Europe, which are said to be equipped with ETCS. That means that the big goal of unrestricted interoperability would have been missed, at least until 2010. The next major release of the System Requirements Specification (SRS 3.0.0), also called "baseline 3", is expected to elimination those shortcomings. Baseline 3 has another important feature: Other than all previous SRS versions, which have been published under the copyright of UNISIG, an association of major European signaling manufacturers, SRS 3.0.0 in the opposite has been published as an official document by the European Railway Agency (ERA) a governmental organization implemented by the European Commission. This gives the SRS a status of a public domain document. That means, everyone in Europe is legally entitled to use that information in order to build ETCS compliant equipment.

2.3 Quality Deficiencies in Software Products

Everyone who has ever used software products knows that almost all software has errors and no respectable software company claims, that their software is totally free of defects. There are various opportunities to make mistakes during the life cycle of software production and maintenance: Starting with System Analysis, System Requirement Specification, Functional Requirement Specification, etc., down to the software code generation, integration, commissioning, operation and maintenance phases. A NASA Study on Flight Software Complexity [12] shows contribution to bug counts, which can be expected in different steps of software production (figure 2).

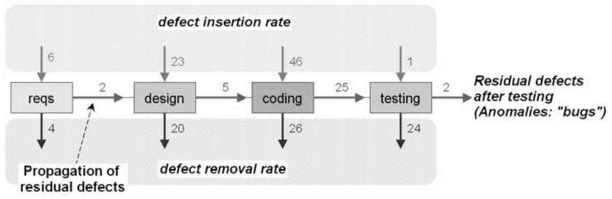

defect insertion rate
6
23
46
1
reqs
2
design
5
coding
25
testing
2
Residual defects after testing (Anomalies: "bugs")
4
20
26
24
Propagation of residual defects
defect removal rate

FIGURE 2: *Propagation of residual defects (bugs) as a result of defect insertion and defect removal rates during several stages of the software production process, according to a NASA research on high assurance software for flight control units for each 1000 lines of code (TLOC) [12].*

Figure 3 characterizes the actual situation in the European signaling industry with several equipment manufacturers working in parallel, using the same specification document in natural language precision giving room for interpretation, combined with different ways and traditions of making mistakes, resulting in a low degree of standardization, even for those components, which cannot be used for product differentiation (core functionality according to UNISIG subset 026 [1]). Since all or at least most of the documents are created by humans, there is always the human factor involved, causing ambiguities and therefore divergent results. Herbert Klaeren refers to reports in his lecture [9], which have found an average of 25 errors per 1000 Lines Of programming Code (TLOC) for newly produced software. The book Code Complete by Steve McConnell has a brief section about errors to be expected. He basically says that there is a wide range [10]:

(a) Industry Average: "about 15 - 50 errors per 1000 lines of delivered code." He further says this is usually representative of code that has some level of structured programming behind it, but probably includes a mix of coding techniques.

(b) Microsoft Applications: "about 10 - 20 defects per 1000 lines of code during inhouse testing, and 0.5 defect per TLOC in released products [10]." He attributes this to a combination of code-reading techniques and independent testing.

(c) "Harlan Mills pioneered a so called 'clean room development', a technique that has been able to achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product (Cobb and Mills 1990 [11]). A few projects - for example, the space-shuttle software - have achieved a level of 0 defects in 500,000 lines of code using a system of formal development methods, peer reviews, and statistical testing."

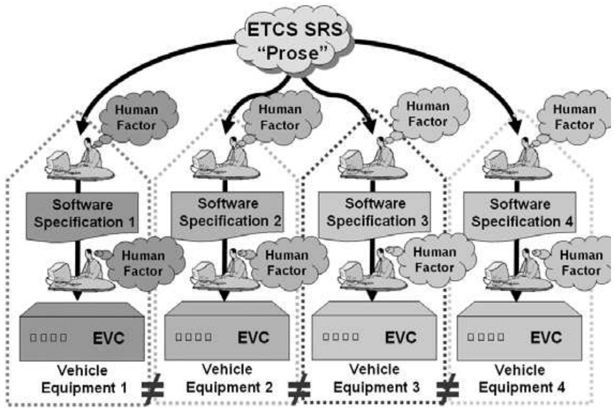
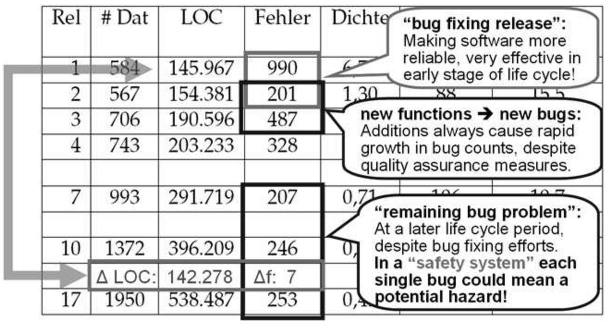


FIGURE 3: *Divergent interpretation of a common public domain ETCS System Requirement Specification (SRS) document, due to the human factor by all parties involved, causing different software solutions with deviant behavior of products from different manufacturers, which result in interoperability deficiencies and costly subsequent improvement activities.*

However the U.S. space shuttle software program came at a cost level of about U.S. $ 1,000 per line of code (3 million LOC   3 billion U.S. $ [9], cost basis 1978), not typical for the railway sector, which is more in a range between 30 per LOC for non-safety applications and up to 100 for SIL 3-4 quality (SIL: Safety Integrity Level) levels.

2.4 Life Cycle of Complex Software

While on the one hand, electronic components are becoming increasingly powerful, yet lower in cost, on the other hand, cost levels of complex software products are increasingly rising not only because the amount of code lines need tremendous men power, but for those lines of code a poof of correctness, or also called safety case, has to be delivered in order to reach approval for operation in revenue service. Some manufacturers have already reported software volumes of over 250 TLOC for the ETCS core functionality defined by ETCS SRS subset 026 [1]. It is very difficult to receive reliable statistics about errors on safety related soft- ware products, because almost all software manufacturers hide their source code using proprietary license agreements. However we can assume that software in other mission critical systems, like communication servers, may have the same characteristics with respect to bug counts. One of those rare examples published, was taken from an AT&T communication software project, which from its size is in the same order of magnitude as todays ETCS onboard software packages (figure 4) [9],[13]. One particular characteristic in figure 4 is quite obvious: The size of the software is continuously growing from version to version, despite the fact that this software was always serving the same purpose. Starting with a relatively high bug count of almost 1000 bugs in less than 150 TLOC, the software matures after several release changes and is reaching a residual bug density, which is less than a tenth of the initial bug density. During its early life the absolute number of bugs is oscillating and stabilizes in its more mature life period. At a later phase of the life cycle the absolute number of bugs is slightly growing despite a decreasing bug density. The late life-cycle bug density is mainly determined by the effectiveness and quality measures taken on the one hand and the number of functional changes and adaptations built in since last release on the other hand. The actual number of bugs is often unknown and can only subsequently been determined.



Rel	# Dat	LOC	Fehler	Dichte		
1	584	145.967	990	[illegible]	[illegible]	[illegible]
2	567	154.381	201	1,30	88	15,5
3	706	190.596	487	[illegible]	[illegible]	[illegible]
4	743	203.233	328	[illegible]	[illegible]	[illegible]
7	993	291.719	207	0,71	[illegible]	[illegible]
10	1372	396.209	246	[illegible]	[illegible]	[illegible]
	Δ LOC:	142.278	Δf: 7			
17	1950	538.487	253	[illegible]	[illegible]	[illegible]

"bug fixing release": Making software more reliable, very effective in early stage of life cycle!

new functions ➔ new bugs: Additions always cause rapid growth in bug counts, despite quality assurance measures.

"remaining bug problem": At a later life cycle period, despite bug fixing efforts. **In a "safety system" each single bug could mean a potential hazard!**

FIGURE 4: *Bug fixing history and growth statistics of an AT&T communication server software package over a life cycle of 17 releases [9], [13].*

Even though that extensive testing has been and still is proposed as an effective method for detecting errors, it has become evident, that by testing alone the correctness of software cannot be proven, because tests can only detect those errors for which the test engineer is looking for [14]. This means ultimately that there is no way to base a safety case on testing alone, because the goal is not to find errors, but to prove the absence of errors. One of the great pioneers of software engineering, Edsgar W. Dijkstra, has put it into the following words [15]:

Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.

We have even to admit that at the current state of software technology, there is no generally accepted single method to prove the correctness of software that means, there is no way to prove the absence of bugs, at least not for software in a range of 100 TLOC or more. The only promising strategy for minimizing errors is:

- The development of functional and design specifications (architecture) has to be given top priority and needs adequate resources,
- The safety part of the software has to be kept as small as possible, and
- The software life-cycle has to undergo a broad as possible, manifold, and continuous review process.

Successful software projects require for the first point, the specification, at least between 20% and 40% [12] of the total development cost, depending on the size of the final software package. Trying to save at this point will almost certainly result in inflated costs during later project phases (see figure 5)

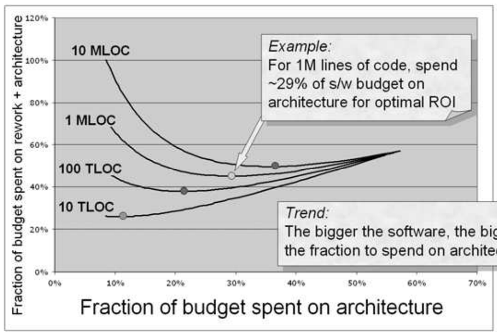


FIGURE 5: *Fraction of the over all project budgets spent on specification (architecture) versus fraction of budget spend on rework + architecture, which defines a so called sweet spot where it reaches its minimum [12]. However this cost function does not take any potential damages into account, which might result from fatalities caused by software bugs.*

Formal modeling methods and close communication with the end-user may be helpful in this stage, especially when operational scenarios can be modeled formally as well in order to verify and validate the design. Specification, modeling and reviews by closely involving the customer may even require several cycles in order to come to a satisfactory result.

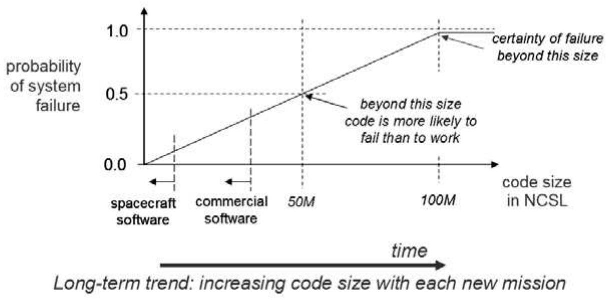


FIGURE 6: *Graph taken from a NASA Study on Flight Software Complexity [12] suggesting a reasonable limit for software size, determine a level, which results in certainty of failure beyond this size (NCSL: Non-Commentary Source Lines = LOC as used in this paper).*

Large railway operators in the opposite may have several hundreds of trains, representing the same level of material value (but carrying hundreds of passengers) operating at the same time. Assuming that a mission critical failure in software of a particular size may show up once in 1000 years, would mean for a space mission duration of 1 year, a probability for a mission failure of about 0,1% due to software (equal distribution assumed). For the railway operator however, who operates 1000 trains at the same time, having the same size and quality of software on board may cause a mission failure event about once a year, making drastically clear that code size matters. While the third point is very difficult to get implemented within conventional closed source software projects, simply because highly qualified review resources are always very limited in commercial projects. Therefore errors are often diagnosed at a late stage in the process. Their removal is expensive and time consuming. That is why big software projects often fail due to schedule overruns and cost levels out of control and often even been abandoned altogether. Never the less, continuous further development and consistent use of quality assurance measures can result in a remarkable process of maturation of software products, which is demonstrated by the fact that in our example in figure 4 the bug density has been reduced by more than an order of magnitude (initially above 6 bugs/TLOC down to below 0.5 bugs/TLOC). On the other hand, in a later stage of the life cycle, due to the continuous growth of the number of code lines, which seem to go faster than the reduction of the bug density, a slight increase of the total number of bugs, can be observed. Given a certain methodology and set of quality assurance measures on the one hand and a number of change requests to be implemented per release, then this will result in a certain number of bugs that remains in the software. Many of those bugs stay unrecognized forever. However some are also known errors, but their elimination is either not possible for some reason or can only be repaired at an unreasonably high level of cost. The revelation of the unknown bugs can take several thousand unit operation years (number of units times number of years of operation) and must be considered as a random process. That means for the operator, that even after many years of flawless operation, unpleasant surprises have to be expected at any time. In Europe, in a not too distant future up to 50,000 trains will operate with ETCS, carrying millions of passengers daily, plus unnumbered trains with hazardous material. Then the idea is rather scary that in any of those European Vital Computers (EVC), the core element of the ETCS vehicle equipment, between 100 and 1000 undetected errors are most likely left over, even after successfully passing safety case assessments and after required authorization has been granted. Even if we would assume, that only one out of 100 bugs might eventually cause a hazard [12], that still means 1 to 10 mission critical defects per unit. Further more, there will be several different manufacturer-specific variants of fault patterns under way.

2.5 New Technologies Have to Have At Least Same Level of Safety

According to German law (and equally most other EU states) defined in the EBO (Eisenbahn Bau- und Betriebsordnung: German railway building and operation regulations) any new technology has to maintain at least the same safety level as provided by the preceding technology [16]. Assuming that the more complex ETCS technology requires about ten times more software code than legacy technology like LZB and PZB as an average and given the fact, that PZB and LZB have already reached a very mature stage of the software integrated after almost 3 decades of continues development, then it seams very unlikely, that the at least same level of safety can be proven by using the same technical rules and design practices for a relatively immature ETCS technology. In addition, due to less service experience the criticality of deviations from expected reaction patterns are difficult to assess. This raises the question whether proprietary software combined with a business model that sells the software together with the hardware device, and then - as up to now - will be operated largely without any defined maintenance strategy, might be inadequate for such a gigantic European project. A project eventually replacing all legacy, but well service proven signaling and train protection systems with one single unified, but less service proven technology, especially when independent verification and system validation is only provided at rare occasions by a very limited number of experts .... or ... then again after a critical incident has taken place only? Instead, a broad and continues peer review scheme with full transparency in all stages of the life cycle of ETCS on-board software products would be highly recommended. In particular during the critical specification, verification and validation phases, following the so called Linus's Law, according to Eric S. Raymond, which states that:

given enough eyeballs, all bugs are shallow.

More formally:

Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone.

The rule was formulated and named by Eric S. Raymond in his essay The Cathedral and the Bazaar [17]. Presenting the code to multiple developers with the purpose of reaching consensus about its acceptance is a simple form of the software reviewing process. Researchers and practitioners have repeatedly shown the effectiveness of the reviewing process in finding bugs and security issues [18].

2.6 Changing Business Model for Software: From Sales to Service

Such problems can be solved by changing the business model. Looking at software as a service rather than a commodity or product (in its traditional definition) is justified by the fact that software is growing continuously in size (but not necessarily increasing the value to the user) as shown in figure 4. Over a life-cycle of 17 releases, the total size of that software grew by more than 300%. Furthermore, about 40% of the modules of the first release had to be fixed or rewritten, due to one or more bugs per module. That means only 60% of the original modules were reusable for second or later releases. It is fair to assume that half of the original 60% virtually bug free code had to be adapted or otherwise modified to use it for functional enhancements. This results in not more than 30% of remaining code, which is about 50 TLOC of the original code having a chance to survive unchanged up to release No. 17. Biggerstaff [19] and Rix [20] suggest that these assumptions might even be too optimistic, as long as no specific measures have been taken in order to support reusability of code. It can be assumed that a potential sales price of all versions would be at the same level, since all versions serve in principle the same functions. That means during its life cycle only 10% (50 TLOC out of about 500 TLOC) of the final code was left unchanged from the first version in this particular example. In other words: 90% of the work (code lines) has been added by software maintenance and continuous further development efforts over the observed life cycle, which can be considered as servicing the software. In order to make this a viable business, users and software producers have to contract so called service agreements for a certain period of time. This kind of business model can be a win-win for both, users and manufacturers alike. Manufacturers are generating continuous cash flow, allowing them to maintain a team of experts over an extended period of time, dedicated to continuously improving the software. Users in exchange are having guaranteed access to a qualified technical support team ensuring fast response in a case of critical software failures. Proprietary software makes it mostly impossible for the user to switch software service providers later on, but leave users in a vendor lock-in situation with no competition on the software service market. Competition however is the most effective driver for quality improvement and cost efficiency.

Considering commercial, technical, safety and security aspects, the risks associated with complex closed source software should be reason enough for the railway operators to consider alternatives, in particular when a large economic body, like the European Union, defines a new technological standard. Watts Humphrey, a fellow of the Software Engineering Institute and a Recipient of the 2003 National Medal of Technology (US), has put the general problem of growing software complexity in these words [21]:

While technology can change quickly, getting your people to change takes a great deal longer. That is why the people-intensive job of developing software has had essentially the same problems for over 40 years. It is also why, unless you do something, the situation wont improve by itself. In fact, current trends suggest that your future products will use more software and be more complex than those of to- day. This means that more of your people will work on software and that their work will be harder to track and more difficult to manage. Unless you make some changes in the way your software work is done, your current problems will likely get much worse.

3 Proposal: Free / Libre Open Source Software for ETCS

A promising solution for the previously described difficulties could be given by providing an Open Source ETCS onboard system, making the embedded software source code and relevant documentation open to the entire railway sector. Open Source Software, Free Software or Libre Software more often called Free/Libre Open Source Software short: FLOSS [22], is software that:

- Can be used for any purpose,
- Can be studied by analyzing the source code,
- Can be improved and modified and
- Can be distributed with or without modifications.

This basic definition of FLOSS is identical to the Four Freedoms, with which the Free Software Foundation (FSF, USA, [23]) has defined free software and is in line with the open source definition formulated by the Open Source Initiative (OSI) [24].

3.1 Public License for an European Project

A potential candidate for a license agreement text could be the most widely used General Public License (GPL) or occasionally called GNU Public License, which has been published by the Free Software Foundation [23]. Because this license text (and several similar license texts as well) is based on the Anglo-American legal system. In Europe applicability and enforceability of certain provisions of the GPL are considered as critical by many legal experts. The European Union has recognized this problem some time ago and has issued the European Union Public License text [25], which not only is available in 22 official EU languages, but is adapted to the European legal systems, so that it meets essential requirements for copyright and legal liability issues. The EU Commission recommends and uses this particular License for its own European eGovernment Services project (iDABC [26]). A key feature of the aforementioned license types is the so-called strong "Copy Left" [27]. The Copy-Left requires a user who modifies, extends or improves the software and distributes it for commercial or non-profit purposes, to make also the source code of the modified version available to the community under the same or at least equivalent license conditions, which has applied to the original software. That means everybody will get access to all improvements and further developments of the software in the future. The distribution in principle has to be done free of charge, however add-on services for a fee are permissible. That means for embedded control systems, that software-hardware integration work, vehicle integration, homologation and authorization costs can be charged to the customer as well as service level agreements for a fee are allowed within the EUPL. By applying such license concept to the core functionality of the ETCS vehicle function as defined and already published in UNISIG subset 026 of the SRS v3.0.0 [1] all equipment manufacturers as well as end-users would be free to use this ETCS software partly or as a whole in their own hardware products or own vehicles. Due to the fact that a software package of substantial value would be then available to all parties, there would be not much incentive any more for newcomers to start their own ETCS software development project from scratch, but would more likely participate in the OSS project and utilize the effect of cost sharing. Also established manufacturers, who already may have a product on their own, might consider sharing in for all further add-on functions by trying to provide an interface to the OSS software modules with their own existing software. This will result in some kind of an informal or even formally set up consortium of co-developing firms and a so called open source eco-system around this core is most likely to evolve. This has been demonstrated by many similar FLOSS projects. The effect of cooperation of otherwise in competition operating firms, based on a common standard core product, is often called co-competition. In analogy with other similar open projects the name openETCS has been suggested for such a project. Occasionally expressed concern that such a model would squander costly acquired intellectual property of the manufacturers to competitors does not really hit the point, because on the one hand the essential functional knowledge, which is basically concentrated in the specification, has already been published by UNISIG and ERA within the SRS and cannot be used as unique selling point. On the other hand implementation know-how for specific hardware architecture and vehicle integration as well as service knowledge will not be affected and has the potential to become part of the core business for the industry. In addition, for the pioneering manufacturer open up his own software could not be better investment money, if this software becomes part of an industrial standard, which is very likely (if others are not quickly following this move) as demonstrated several times in the software industry. Not only that, but since safety related software products are closely related to the design process, tools and quality assurance measures, the pioneering OSS supplier would automatically make his way of designing software to an industrial standard as well (process standardization). Late followers had simply to accept those procedures and may end up with more or less higher switching costs, giving the pioneer a head start. Even in the case that one or two competitors would do the same thing quickly, those companies could form a consortium sharing their R&D cost and utilizing the effect of quality improving feedback from third parties and therefore improving their competitive position compared to those firms sticking with a proprietary product concept. The UNUMERIT study on FLOSS [22] has shown cost lowering (R&D average of 36%) and quality improving effects of open source compared with closed source product lines.

3.2 ETCS Vehicle On-Board Units with openETCS

Software that comes with a FLOSS license and a Copy-Left, represent some kind of a gift with a commitment, namely as such that the donor has almost a claim to receive any improvements made and further distributed by the recipient. That means all the technical improvements, which have been based on collective experiences of other users/developers and integrated into product improvements need to be distributed so that even the original investor gets the benefits. By recalling the fact that during the life cycle of a large software product, as shown in figure 4, more than 90% of the code and improvements were made after the first product launch, means that sharing the original software investment with a community (eco-system) becomes a smart investment for railway operators and manufacturers alike, by simply reducing their future upgrade and maintenance costs significantly. Rather than starting to develop a new open source software package from scratch, the easiest and fastest way for a user to reach that goal would be simply by selecting one of the already existing and (as far as possible) service proven products from the market and put it under an appropriate open source license. There are numerous examples from the IT sector, such as the software development tool Eclipse, the successor to IBMs Visual Age for Java 4.0., source code was released in 2001 [28], the Internet browser Mozilla FireFox (former: Netscape Navigator), and office communication software Open Office (former: StarOffice) and many more.

3.3 Tools and Documents Need to be Included

In the long term it will not be enough only to make the software in the on-board equipment open. Tools for specification, modeling and simulation as well as software development, testing and documentation are also essential for securing quality and lowering life cycle cost. To meet the request for more competition in the after sales software service business and avoiding vendor lock-in effects, requires third parties to be in a position to maintain software, prepare safety case documents, and get the modified software authorized again without depending on proprietary information. A request no one would seriously deny for other safety critical elements e.g. in the mechanical parts section of a railway vehicle, like loadbearing car body parts or wheel discs. The past has shown that software tools are becoming obsolete quite often due to new releases, changing of operating systems, or tool suppliers simply going out of business, leaving customers alone with little or no support for their proprietary products. Railway vehicles are often in revenue service for more than 40 years and electronic equipment is expected to be serviced for at least 20 years and tools need to be up to the required technical level for the whole period. The aircraft industry with similar or sometimes even longer product lifecycles has realizing this decades ago, starting with ADA compiler in the 1980th, specifically designed for developing high assurance software for embedded control design projects, originally initiated by the US Air Force and developed by the New York University (GNAT: GNU NYU Ada Translator), which is available in the public domain and further developed by AdaCore and the GNU Project [3], [23] and a somewhat more sophisticated tools chain, which is called TOPCASED, initiated by AIRBUS Industries [29]. TOPCASED represents a tools set, based on ECLIPSE (another OSS software development tools platform [28]) for safety critical flight control applications with the objective to cover the whole life cycle of such software products, including formal specification, modeling, software generation, verification and validation, based on FLOSS in order to guarantee long term availability. TOPCASED seams to be a reasonable candidate for a future openETCS reference tools platform, since it is a highly flexible open framework, adaptable in various ways for meeting a wide range of requirements. Today manufacturers in the rail segment are using a mix of proprietary and open source tools, since some software development tools like ADA and other Products from the GNU Compiler Collection (GCC) [24] have already been used in several railway projects. Even FLOSS tools, not specifically designed for safety applications, like BugZilla for bug tracing and record keeping, have already been found its way into SIL 4 R&D programs for railway signaling [30]. The importance of qualified and certified tools is rising, since it became obvious, that poor quality tools or even malware infected tools can have a devastating effect on the quality of the final software product. 6.6 of proposed prEN 50128:2009 norm [31], modification and change control, requires to take care of the software development tools chain and processes, which in the future formally have to comply with requirements for the respective SIL level of the final product. Recent news about the STUXNET attack, a type of malware (worm) specifically designed to target industrial process control computers via its tools chain (maintenance PCs with closed source operating system) has made pretty clear, that no one can be lulled into security even not with control and monitoring systems designed for safety critical embedded applications [7]. Ken Thompson, one of the pioneers of the B Language, a predecessor of C, and UNIX operating system design has demonstrated in his Reflections on Trusting Trust [2] that compilers can be infected with malicious software parts in a way that the resulting executable software (e.g. an operating system) generated by this compiler out of a given clean (means: free of malware) source code, can be infected with a backdoor, almost invisible for the programmer. It took several years of research until David A. Wheeler suggested in his dissertation thesis (2009) a method called Diverse Double-Compiling [32], based on open source tools for countering the so called Thomson Hack. Therefore Wheeler suggests on his personal website:

Normal mathematicians publish their proofs, and then depend on worldwide peer review to find the errors and weaknesses in their proofs. And for good reason; it turns out that many formally published math articles (which went through expert peer review before publication) have had flaws discovered later, and had to be corrected later or withdrawn. Only through lengthy, public worldwide review have these problems surfaced. If those who dedicate their lives to mathematics often make mistakes, its only reasonable to suspect that software developers who hide their code and proofs from others are far more likely to get it wrong. ... At least for safety-critical work making FLOSS (or at least world-readable) code and proofs would make sense. Why should we accept safety software that cannot undergo worldwide review? Are mathematical proofs really more important than software that protects peoples lives? [3]

3.4 Open Proof the ultimate Objective for openETCS

Wheelers statement confirms the need for an open source tools chain to cover the software production and documentation process for verification and validation into the open source concept in total, providing an Open Proof (OP) methodology [33]. OP should be then the ultimate objective for an openETCS project, in order to make the system as robust as possible for reliability, safety as well as for security reasons. An essential precondition for any high quality product is an un-ambiguous specification. Until this day only a written more or less-structured text in natural language is the basis for ETCS product development, leaving more room for divergent interpretation (figure 3) than desirable. A potential solution for avoiding ambiguities right in the beginning of the product development process could be the conversion into a formal that means mathemati- cal description of the functional requirement specification. As recommended by Jan Peleska in his Habiltationsschrift (post doctorial thesis) [34]:

... how the software crisis should be tackled in the future:

- *The complexity of today's applications can only be managed by applying a combination of methods; each of them specialised to support specific development steps in an optimised way during the system development process.*

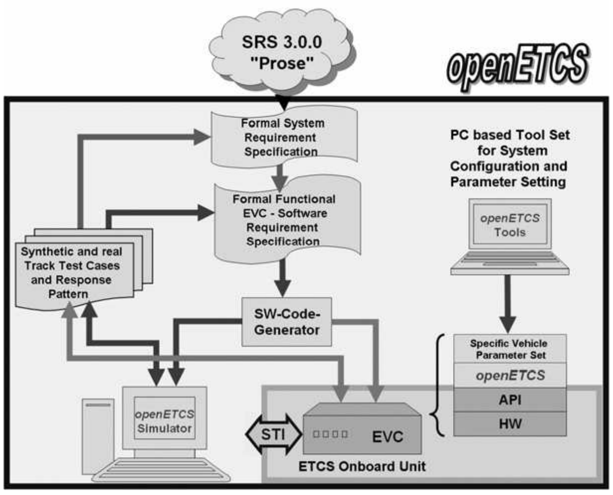
- *The application of formal methods should be supported by development standards, i.e., explanations or "recipes" showing how to apply the methods in the most efficient way to a specific type of development task....*

- *The application of formal methods for the development of dependable systems will only become cost-effective if the degree of re-usability is increased by means of re-usable (generic) specifications, re-usable proofs, code and even re-usable development processes.*

Despite the fact that several attempts have been made in the past, a comprehensive Formal Functional Requirement Specification (FFRS) has never been completed for ETCS due to lack of resources and/or funding. Based on proprietary software business concepts there is obviously not a positive business case for suppliers for a FFRS. Formal specification works does not have to be started from scratch, because there are already a number of partial results from a series of earlier work, although that different approaches, methods and tools have been used [35], [36], [37]. Evaluating those results and trying to apply a method successfully applied in several open source projects and known as a so called Stone Soup Development Methodology might be able to bring all those elements and all experts involved together in order to contribute to such project at relatively low cost [3], [38].

3.5 Formal Methods to validate Specification for openETCS

In the first step of formalization only a generic, purely functional and therefore not implementation related specification has to be developed. This can be mainly done in the academic sector and by R&D institutes. However railway operators have to feed in their operational experience, in order to make sure that man-machine-interactions and case studies for test definitions are covering real life operational scenarios and not only synthetic test cases of solely academic interest.



SRS 3.0.0 "Prose"
openETCS
Formal System Requirement Specification
PC based Tool Set for System Configuration and Parameter Setting
Formal Functional EVC - Software Requirement Specification
openETCS Tools
Synthetic and real Track Test Cases and Response Pattern
SW-Code-Generator
Specific Vehicle Parameter Set
openETCS
API
HW
openETCS Simulator
STI
EVC
ETCS Onboard Unit

FIGURE 7: *Proposed openETCS based on ERAs base line 3 SRS natural English specification text (= prose) converting into a formal functional specification to define software for modeling as well as embedded control integration, providing equipment manufacturers to integrate the openETCS kernel software via API into their particular EVC hardware design.*

For verification purposes a test case data base need to be derived from the functional specification and supplemented by a response pattern data base, which defines the expected outcome of a certain test case. That database needs to be open for all parties and should collect even all real world cases of potentially critical situations and in particular those cases, which have already caused safety relevant incidents. That means this type of formalized database will keep growing and continuously being completed to make sure that all lessons learned are on record for future tests. State of the art formal specification tools do not only provide formatting support for unambiguous graphical and textual representation of a specification document, but provide also a modeling platform to execute the model in a more or less dynamical way. This modeling can be used to verify the correctness and integrity of the ETCS specification itself not only statically, but also dynamically. In addition transitions to and from class B systems need to be specified formally as well and that might depend on national rules, even in those cases where the same class B system is used (e.g. for PZB-STMs hot stand-by functions are handled differently in Germany and Austria). Based on a particular reference architecture the resulting formal functional specification can be transformed in a formal software specification and then converted into executable software code. Even without existing real target hardware, those elements can be used to simulate the ETCS behavior and modeling critical operational test cases in a so called Software-in-the-Loop modeling set-up. Once the specification of the functionality has been approved and validated, the code generation can be done for the EVC embedded control system. Standardization can be accomplished by providing an Application Programmer Interface (API) similar to the approach successfully applied in the automotive industry within the AUTOSAR project [39] or for industrial process control systems based on open Programmable Logic Control (PLC) within the PLCopen project [40] including safety critical systems. In addition to the software specification, generation, verification and validation tools chain also tools for maintenance (parameter setting, system configuration, software upload services) have to be included in the OSS concept, as shown in figure 7.

3.6 How FLOSS can meet Safety and Security Requirements

For many railway experts, not familiar with open source development methodology, open source is often associated with some kind of chaotic and arbitrary access to the software source code by amateur programmers (hackers), completely out of control and therefore not suited for any kind of quality software production. This may have been an issue of the past and still being in existence with some low level projects, adequate for their purpose. However since OSS license and R&D methodologies concepts have successfully been applied to unnumbered serious business projects, even for the highest safety and security levels for governmental administration, e.g. within the iDABC, European eGovernment Services Project [26] as well as commercial, avionics [29] and military use [24], a concept based on a group of qualified and so called Trusted Developers (figure 8) having exclusively access to a so called Trusted Repository, which on the other hand can be watched and closely monitored by a large community of developers, being able to post bug reports and other findings visible to the whole community, has made this so called bazaar process [17] to a much more robust methodology compared with any other proprietary development scheme. According to several research projects, OSS projects in general tend to find malicious code faster than closed source projects, which is indicated for example in the average life time of so called backdoors, a potential security threat, which might exist in closed source software for several month or even years, while having an average survival time of days or few weeks, at the most, in the case of well managed OSS projects [3], [4], [5], [32]. Figure 8 demonstrates the principle information and source code flow for a typical FLOSS development set-up.

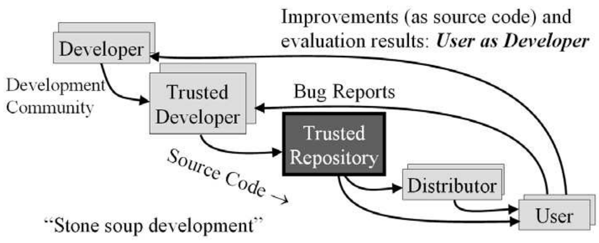


FIGURE 8: *The classical Stone Soup Development Methodology often applied in Open Source Software projects according to [3], where the User in most cases is also active as Developer, which need to be adapted to the rail sector, where Users my be more in a reporting rather developing role. Only trusted developers are privileged to make changes to the source code in the trusted repository, all others have read only access.*

It is not in question that well acknowledged and mandatory rules and regulations according to state of the art R&D processes and procedures (e.g. EN 50128) have to be applied to any software part in order to get approval from safety authorities before going into revenue service. While open source eco-systems in the IT industry are generally driven by users, having the expertise and therefore being in a position to contribute to the software source code themselves, so it seems unlikely for the railway segment to find many end users of embedded control equipment for ETCS (here: railway operators or railway vehicle owners), who will have this level of expertise. Therefore the classical OSS development concept and organization has to be adapted to the railway sector. Figure 9 shows a proposal for an open source software development eco-system for openETCS utilizing a neutral organization to coordinate the so-called "co-competition" business model for cooperating several competing equipment integrators and distributors for ETCS onboard products and services based on a common FLOSS standard core module, adapted to the needs of the railway signaling sector providing high assurance products to be authorized by safety authorities (NSA, NoBo). The concept as shown in figure 9 assumes a license with Copy-Left, requiring in general distributing the source code free of charge, even if code has been added or modified and further distributed, so that the community can re-use the improvements as well. That means that only certain added values can be sold for a fee. Typical added values can be service for software maintenance (bug-fixing), software adaptation for specific applications, integration into embedded control hardware and integration into the vehicle system, test and homologation services, training for personnel and so forth.

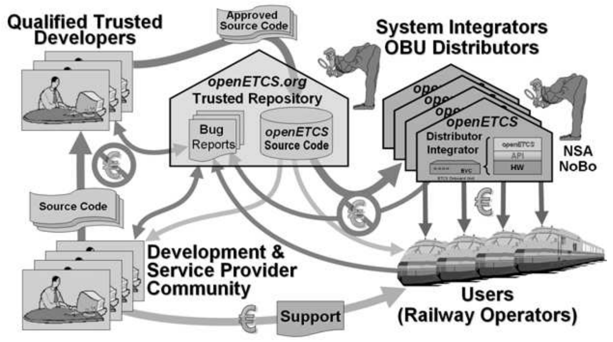

Qualified Trusted Developers
Approved Source Code
System Integrators OBU Distributors
openETCS.org Trusted Repository
Bug Reports
openETCS Source Code
Distributor Integrator
openETCS API HW
NSA NoBo
Source Code
Development & Service Provider Community
Support
Users (Railway Operators)

FIGURE 9: *Proposal for an openETCS eco-system using a neutral organization to coordinate a so-called "co-competition" business model, showing flow of software source code, bug reports and ad-on services provided for a fee.*

For further development of the software, especially for the development of new complex add-on functions, costly functional improvements, etc., it might be difficult to find funding, since a Copy-Left in the FLOSS license requires to publish that software free of charge, when distributed.

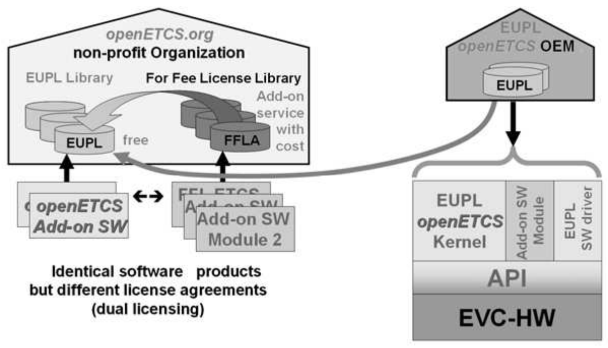


FIGURE 10: *Dual-Licensing concept providing a cost sharing scheme for financing new functions (Add-on SW Modules) and improvements for those users who need it, by keeping cost low for those not requiring enhanced functionality.*

Therefore many OSS projects are using a so called dual licensing policy (or even multi license policy), by offering the identical software under two (or more) different license agreements (figure 10). One might be the European EUPL, a Copy-Left type FLOSS license and the other one can be a For-Fee-License (without Copy-Left), which does not require publishing all modification. In exchange ac certain fee has to be paid, which may also provide for warranty and other services. Combined with a scheduled release scheme (e.g. defining a fixed release day per year or any other reasonable frequency), all new modules will be available only under the For-Fee-License first, until R&D costs have been paid off by those users, who want to make use of the new functionality, while all others can stick with the older, but free of charge software versions. Once the new features are paid off, those particular software modules can then be set under the FLOSS license (EUPL). That allows fair cost sharing for all early implementers and does not leave an undesired burden on those users, who can live with- out the additional functions for a while, but still being able to upgrade later on.

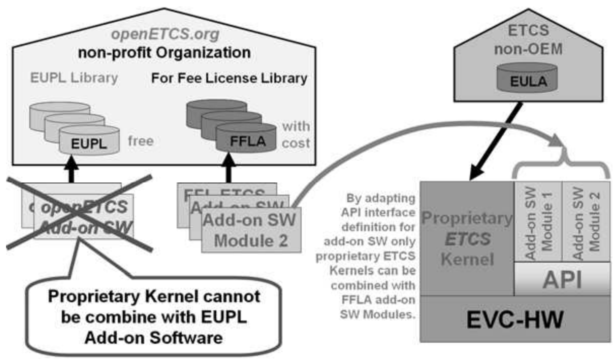


FIGURE 11: *Applying a dual-licensing model helps even those non-OSS ETCS suppliers participating in cost sharing for add-on modules even if the usage of the openETCS software kernel is not possible due to technical or licensing incompatibilities, but by providing a Mini-API some or all future new add-on functions can be integrated.*

Since those upgrades will be provided by service level agreements through OEMs or software service providers, customers have the choice to either opt for low cost, but later upgrade service or higher priced early implementing services, whatever fits best to their business needs. The dual-licensing scheme has an additional advantage, allowing even those ETCS suppliers, who are not able, due to technical limitations or legal restrictions caused by their legacy system design or other reasons, to put their software under an OSS license, never the less being able to participate in the cost sharing effects for further add-on functional development. In most cases it is technically much easier to implement a small API, interfacing just for the add-on functions, rather than providing a fully functional API for the whole kernel (figure 11).

If the non-OSS supplier wants to make use of those Add-on-SW-Modules from the library, he cannot use the OSS-licensed software, but can combine any proprietary software with alterative licensed software, not including a Copy-Left provision. Besides commercial matters also technical constrains have to be taken into account when combining software parts, developed for different architectural designs. A concept of hardware virtualization has already been discussed to overcome potential security issues [43].

3.7 How to Phase-in an OSS Approach into a Proprietary Environment?

Even though the original concept of the ETCS goes far back into the early 1990 years projecting an open white-box design of interchangeable building blocks, independent from certain manufacturers, based on a common specification and mainly driven by the railway operators organized in the UIC (Union International des Chemin de Fer = International Union of Railways), software was not a central issue and open source software concepts were in its infancy [41], [42]. Since then a lot of conceptual effort and detailed product development work has been done, but the white box approach has never been adapted by the manufacturing industry. Despites various difficulties and shortcomings, as mentioned earlier, the European signal manufacturers have developed several products, more or less fit for its purpose and it would be unwise to ignore this status of development and start a brand new development path from scratch. This would just lead to another product competing in an even more fragmented market rather than promoting an effective product standard. In addition, it needs at least one strong manufacturer with undoubted reputation and a sound financial basis in combination with a sufficient customer base to enforce a standard in a certain market. Therefore starting a new product line, by having the need to catch up with more than a decade of R&D efforts is not an option.

Based on this insight, a viable strategy has to act in two ways:

- 1. Ground work has to be started to provide an open source reference system, based on an unambiguous specification, which means using formal methods, in order to deliver a reference onboard system as soon as possible, which can be used to compare various products on the market in a simulated as well as real world infrastructure test environment. This device needs to be functionally correct, however does not to be a vital (or fail-safe) implementation.
- 2. At least one or better more manufacturers have to be convinced to share-in into an open source software based business approach by simply converting their existing and approved proprietary ETCS onboard product into an open source software product by just switching to a FLOSS license agreement, preferably by using the European Union Public License (EUPL), including interface definition and safety case documentation. No technical changes are required.
- 3. Once a formally specified reference FLOSS package has been provided, implemented on a non-vital reference hardware architecture, according to step 1, in a future step by step approach all add-on functions and enhancements and future major software releases should be based on formal specifications, allowing a migration of the original manufacturers software design solution into the formal method based approach, due to the openness of the product(s) from step 2.

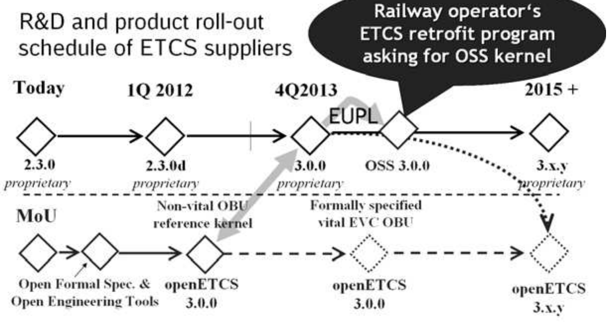

R&D and product roll-out schedule of ETCS suppliers
Railway operator's ETCS retrofit program asking for OSS kernel
Today
1Q 2012
4Q2013
2015 +
EUPL
2.3.0 proprietary
2.3.0d proprietary
3.0.0 proprietary
OSS 3.0.0
3.x.y proprietary
MoU
Non-vital OBU reference kernel
Formally specified vital EVC OBU
Open Formal Spec. & Open Engineering Tools
openETCS 3.0.0
openETCS 3.0.0
openETCS 3.x.y
openETCS project including FLOSS tools

FIGURE 12: *Interaction between openETCS project providing formally specified non-vital reference OBU for validating proprietary as well into OSS converted industrial products and for future migration to a fully formally specified openETCS software version to implemented in a market product.*

Figure 12 demonstrates this two path approach with a conventional roll-out scheme, as planned by a supplier, based on proprietary designs (upper half) and major mile stones for the openETCS project, providing a non-vital OBU based on formal specification and later migrating to a formally specified vendor specific implementation of the kernel software (lower half).

Trying to implement an independent formal open source software package without the backing of at least one strong manufacturer, will most likely fail if no approved and certified product can be used to start with. The only promising way to accomplish the crucial second step in this concept is by using a tender for a sufficiently attractive (large enough) ETCS retrofit project by adding a request for an OSS license for the software to be delivered. The EU commission has provided a guideline for such OSS driven tenders, the so called OSOR Procurement Guide (Guideline on public procurement of Open Source Software, issued March 2010, [26]). As an example, figure 12 shows the time line for an ETCS retrofit project for high speed passenger trains to be equipped by 2012 with an pre-baseline 3 proprietary software in 2012, to be added by an open source license as soon as the first baseline 3 software package is expected to be released.

3.8 Economical Aspects of openETCS for Europes Railway Sector

A free of charge, high-quality ETCS vehicle software product on the market, makes it less attractive, under economical aspects, to start a new software development or even further development of a different but functionally identical proprietary software product. This will lead sooner or later to some kind of cooperation of competing ETCS equipment suppliers, a co-competition with all those suppliers who can and will adapt their own products by providing an API to their particular system. Due to the fact that very different design and safety philosophies have been evolved in the past years, some of the manufacturers have to decide either to convert their systems or share-in into the co-competition grouping, or otherwise stick with costly proprietary software maintenance on their own. As figure 4 demonstrates clearly that the increase of the software volume over time may exceed the original volume by a factor of 3. It is unlikely to assume that the development of the ETCS vehicle software will run much differently. Then it will be very obvious that for a relatively limited market, of perhaps up to 50,000 rail cars to be equipped with ETCS in Europe, a larger number of parallel software product development lines will hardly be able to survive. A study funded by the EU Commission [22] has identified a potential average cost reduction of 36% for the corresponding R&D by the use of FLOSS. As a result, a significantly lower cost of ownership for vehicle operators would accelerate the ETCS migration on the vehicle side.

3.9 Benefits for the ETCS Manufacturers

The core of the ETCS software functionality defined by UNISIG subset 026, to be implemented in each EVC, is a published and binding standard requirement and therefore not suitable for defining an Unique Selling Proposition (USP). As a result it makes perfectly sense from the perspective of manufacturers, to share the development cost and the risk for all R&D of the ETCS core functionality even with their competitors, often practiced in other industrial sectors (e.g. automotive). The involvement of several manufacturers in the development of openETCS will help to enhance the quality in terms of security and reliability (stability and safety) of the software, because different design traditions and experiences can easily complement each other. As a FLOSS-based business model can no longer rely on the sales of the software as such, the business focus has to be shifted to services around the software and even other add-on features to the product. That means the business has to evolve into service contracts for product maintenance (further development, performance enhancements and bug fixes). It thereby helps the ETCS equipment manufacturers to generate a dependable long-term cash flow, funding software maintenance teams even long after the hardware product has been discontinued and to cover long term maintenance obligations for the product even by third parties, helping to reserves scarce software development resources for future product R&D. With respect to the scarcity of well educated software engineers from Universities, FLOSS has the side effect, that openETCS can and most likely will become subject to academic research, generating numerous master and dissertation thesiss and student research projects.

3.10 Benefits for Operators and Vehicle Owners

The use of openETCS is a better protection for the vehicle owners investment, because an obsolescence problem on the hardware side does not necessarily mean discontinued software service. Modification of the ETCS kernel can also be developed by independent software producers. This enables competition on after-sales services and enhancements, because not only the software sources but also associated software development tools are accessible to all parties. As shown above, due to the complexity of the software, malfunctions of the system may show up many years, even decades after commissioning. Conventional procure- ment processes are therefore not suitable, since they provide only a few years of warranty coverage for those kinds of defects. These concepts imply that customers would be able to find all potential defects within this limited time frame, just by applying reasonable care and observation of the product by the user, which does not match experiences with complex software packages with more than 100,000 lines of code. This finding suggests that complex software will need care during the whole lifecycle. Since software matures during long term quality maintenance, means that during early usage, or after major changes, the software may need more intensive care whereas in its later period of use, service intensity may slow down. But as long as the software is in use, a stand-by team is needed to counter unforeseeable malfunctions, triggered by extremely rare operational conditions. As the ETCS onboard software can be considered as mission critical, operators are well advised to maintain a service level agreement to get the systems up and running again, even after worst case scenarios. Railway operators and vehicle owners are usually not be able to provide that software support for themselves. They usually rely on services provided by the OEM. However due to slowing service intensity after several years of operation, this service model may not match the OEMs cost structure in particular after the hardware has been phased out. In those cases OEMs are likely to increase prices or even to discontinue this kind of serve. A typical escrow agreement for proprietary software might help, but has its price too, because alternative service providers have first to learn how to deal with the software. Only a well established FLOSS-eco-system can fill in the gap at reasonable cost for the end user, and that is only possible with FLOSS. DBs experience with FLOSS is very positive in general. For more than a decade, DB is using FLOSS in various ways: In office applications, for the intranet and DBs official internet presence and services on more than 2000 servers world-wide and even in business critical applications. The original decision in favor of FLOSS was mainly driven by expected savings on license cost. However looking back, quality became a more important issue over time, since FLOSS application have had never caused a service level breach, which cannot be said for proprietary software, selected by applying the same quality criteria. This supports the impression that FLOSS does tend to have a higher quality.

4 Conclusion

The major goal of unified European train control, signaling, and train protection system, ETCS, has led to highly complex functionality for the onboard units, which converts into a level of complexity for the safety critical software not seen on rail vehicles before. A lack of standardization on various levels, different national homologation procedures and a diversity of operational rules to be covered, combined with interfacing to several legacy systems during a lengthy transitional period has to be considered as a major cost driver. Therefore, even compared with some of the more sophisticated legacy ATP and ATC systems in Europe ETCS has turned out to be far more expensive without providing much if any additional performance or safety advantages. Due to ambiguities in the system requirement specification (SRS) various deviations have been revealed in several projects, so that even the ultimate goal of full interoperability has not yet been accomplished. Therefore the development of ETCS has to be considered as **work in progress**, resulting in many software upgrades to be expected in the near and distant future. Since almost all products on the market are based on proprietary software, this means a low degree of standardization for the most complex component as well as life-long dependency to the original equipment manufacturers with high cost of ownership for vehicle holders and operators. Therefore an open source approach has been suggested, not only covering the embedded control software of the ETCS onboard unit itself, but including all tools and documents in order to make the whole product life cycle as transparent as possible optimizing economy, reliability, safety and security alike. This concept is called open proof a new approach for the railway signaling sector. A dual licensing concept is suggested, based on the European Union Public License with a copy left provision on the one hand, combined with a non-copy left for-fee-license on the other hand to provide a cost sharing effect for participating suppliers and service providers. By offering a trusted repository, a dedicated sources code access policy in combination with a release schedule policy, economical as well as safety and security considerations can be taken into account. A two step approach, providing a formally specified non-vital reference system and a procurement program, asking for converting existing commercial products from closed source into open source, and later merging those two approaches, is expected to enhance quality and safety parameters in the long run. A neutral independent and mainly not-for-profit organization is suggested to manage the project involving all major stake holders to define the future product strategy. The whole openETCS project has to be considered as a business conversion project from a purely competitive sales oriented market into a co-competitive service market, enhancing cooperation on standards by enabling competition on implementation and services. It is well understood that such a change cannot be accomplished even by one of the largest railway operators alone. Therefore several EU railway organizations, as there are: ATOC (UK), DB (D), NS (NL), SNCF (F) and Trenitalia (I) have already signed a Memorandum of Understanding promoting the openETCS concept in the framework of an international project.

References

[1] *The European Railway Agency (ERA): ERTMS Technical Documentation, System requirements Specification - Baseline 3, SUBSET-026 v300* published 01/01/2010, http://www.era.europa.eu/Document-Register/Pages/SUBSET-026v300.aspx

[2] Thompson, Ken: *Reflections on Trusting Trust* ; Reprinted from Communication of the ACM, Vol. 27, No. 8, August 1984, pp. 761-763. http://cm.bell-labs.com/who/ken/trust.html

[3] Wheeler, David A.: *High Assurance (for Security or Safety) and Free-Libre / Open Source Software (FLOSS); updated 20/11/2009*; http://www.dwheeler.com/essays/high-assurance-floss.html

[4] Wysopal, Chris; Eng, Chris: *Static Detection of Application Backdoors*, Veracode Inc., Burlington, MA USA, 2007, http://www.veracode.com/images/stories/static-detection-of-backdoors-1.0.pdf

[5] Poulsen, Kevin, *Borland Interbase backdoor exposed*, The Register, Jan. 2001, http://www.theregister.co.uk/2001/01/12/borland_interbase_backdoor_exposed

[6] EUROPEAN PARLIAMENT: *REPORT on the existence of a global system for the interception of private and commercial communications (ECHELON interception system)*, (2001/2098(INI)), Part 1: Motion for a resolution: A5-0264/2001, 11. July 2001. http://www.europarl.europa.eu/comparl/ tempcom/echelon/pdf/rapport_echelon_en.pdf

[7] FINANCIAL TIMES Europe: *Stuxnet worm causes worldwide alarm*, by Joseph Menn and Mary Watkins, Published: Sept. 24, 2010, Pages 1 and 3 or online version: http://www.ft.com/cms/s/0/cbf707d2-c737-11df-aeb1-00144feab49a.html

[8] Schweizerische Eidgenossenschaft, UUS: *Schlussbericht der Unfalluntersuchungsstelle Bahnen und Schiffe ber die Entgleisung von Gterzug 43647 der BLS AG vom Dienstag*, 16. Oktober 2007, in Frutigen. http://www.uus.admin.ch//pdf/07101601_SB.pdf

[9] Klaeren, Herbert: *Skriptum softwaretechnik*, Universitt Tbingen, Okt. 2007, http://www-pu.informatik.uni-tuebingen.de/users/klaeren/sweinf.pdf

[10] McConnell, Steve: *Code Complete*, 2nd ed. 2004, Microsoft Press; Redmond, Washington 98052-6399, USA, ISBN 0-7356-1967-0

[11] Richard H. Cobb, Harlan D. Mills: *Engineering Software under Statistical Quality Control.*, IEEE Software 7(6): 44-54 (1990)

[12] Dvorak, Daniel L., (Editor): *NASA Study on Flight Software Complexity*, Final Report, California Institute of Technology, 2008, Report: http://www.nasa.gov/pdf/ 418878main_FSWC_Final_Report.pdf, Presentation: http://pmchallenge.gsfc.nasa.gov/ docs/2009/presentations/Dvorak.Dan.pdf

[13] Ostrand, T. J. et al: *Where the Bugs Are.* In: Rothermel, G. (Hrsg.): Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Vol. 29, 2004, Pages 86-96; http://portal.acm.org/ ; see also: http://www-pu.informatik.uni-tuebingen.de/users/klaeren/sw.pdf (German)

[14] Randell, B.: *The NATO Software Engineering Conferences*, 1968/1969: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/

[15] Dijkstra, Edsger W.: *The Humble Programmer*, ACM Turing Lecture 1972. http://userweb.cs.utexas.edu/users/ EWD/ewd03xx/EWD340.PDF

[16] Sukale, Margret: *Taschenbuch der Eisenbahngesetze*, Hestra-Verlag, 13.Auflage 2002

[17] Raymond, Eric Steven: *The Cathedral and the Bazaar*, version 3.0, 11 Sept. 2000 http://www.catb.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html

[18] Pfleeger, Charles P.; *Pfleeger, Shari Lawrence: Security in Computing.* Fourth edition. ISBN 0-13-239077-9

[19] Biggerstaff, Ted J.: *A Perspective of Generative Reuse, Technical Report*, MSR-TR-97- 26,1997, Microsoft Corporation http://research.microsoft.com/pubs/69632/tr-97-26.pdf

[20] Rix, Malcolm: *"Case Study of a Successful Firmware Reuse Program,"* WISR (Workshop on the Institutionalization of Reuse), Palo Alto, CA,. ftp://gandalf.umcs.maine.edu/pub/WISR/wisr5/ proceedings/ .

[21] Watts S. Humphrey; *Winning with Software: An Executive Strategy*, 2001 by Addison- Wesley, 1st Edition; ISBN-10: 0-201-77639-1

[22] UNU-MERIT, (NL): *Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector* http://ec.europa.eu/enterprise/sectors/ict/files/ 2006-11-20-flossimpact_en.pdf

[23] Free Software Foundation, Inc.; 51 Franklin Street, Boston, MA 02110-1301, USA: http://www.gnu.org/philosophy/free-sw.html ,

[24] David A. Wheeler: *Open Source Software (OSS or FLOSS) and the U.S. Department of Defense*, November 4, 2009; http://www.dwheeler.com/essays/dod-oss.ppt

[25] European Commission, *European Union Public License - EUPL v.1.1*, Jan. 9, 2009. http://ec.europa.eu/idabc/en/document/7774

[26] European Commission, iDABC, European eGovernment Services; OSOR; *Guideline on public procurement of Open Source Software*, March 2010, http://www.osor.eu/idabc-studies/OSS-procurement-guideline

[27] Wikipedia, terminology: *Copyleft* http://en.wikipedia.org/wiki/Copyleft

[28] Eclipse Foundation, *About the Eclipse Foundation*, http://www.eclipse.org/org/#about

[29] *TOPCASED: The Open Source Toolkit for Critical Systems*; http://www.topcased.org/

[30] Duhoux, Maarten: *Respecting EN 50128 change control requirements using BugZilla variants*, Signal+Draht, Heft 07+08/2010, EurailPress http://www.eurailpress.de/sd-archiv/number/07_082010-1.html

[31] DIN EN 50128; VDE 0831-128:2009-10; *Railway applications - Communication, signal- ling and processing systems - Software for railway control and protection systems*; version prEN 50128:2009; Beuth Verlag, Germany, http://www.vde-verlag.de/previewpdf/71831014.pdf (index only)

[32] Wheeler, David A.: *Countering the Trusting Trust through Diverse Double-Compiling (DDC)*, 2009 PhD dissertation, George Mason University, Fairfax, Virginia http://www.dwheeler.com/trusting-trust/

[33] *Open Proof:* http://www.openproofs.org/

[34] Jan Peleska: *Formal Methods and the Development of Dependable Systems, Habilitationsschrift*, Bericht Nr. 9612,Universitt Bremen, 1996. http://www.informatik.uni-bremen.de/agbs/jp/papers/habil.ps.gz

[35] Anne E. Haxthausen, Jan Peleska and Sebastian Kinder: *A formal approach for the construction and verification of railway control systems*, Journal: Formal Aspects of Computing. Published online: 17 December 2009. DOI: 10.1007/s00165-009-0143-6 Springer, ISSN 0934-5043 (Print) 1433-299X (Online) http://springerlink.metapress.com/content/ l3707144674h14m5/fulltext.pdf

[36] Lorenz Dubler, Michael Meyer zu Hrste, Gert Bikker, Eckehard Schnieder; *Formale Spezifikation von Zugleitsystemen mit STEP*, iVA, Techn. Univ. Braunschweig, 2002; http://www.iva.ing.tu-bs.de/institut/projekte/Handout_STEP.pdf

[37] Padberg, J. and Jansen, L. and Heckel, R. and Ehrig, H.: *Interoperability in Train Control Systems: Specification of Scenarios Using Open Nets*; in Proc. IDPT 1998 (Integrated De- sign and Process Technology), Berlin 1998, pages 17 - 28

[38] Gary Rathwell: *Stone Soup Development Methodology*: Last updated December 5, 2000 http://www.pera.net/Stonesoup.html

[39] *AUTOSAR (AUTomotive Open System ARchitecture)*; http://www.autosar.org/

[40] PLCopen; Molenstraat 34, 4201 CX Gorinchem, NL,: http://www.plcopen.org/

[41] UIC/ERRI A200: ETCS, European Train Control System, Overall Project Declaration including the contribution to be made by UIC, Utrecht, NL, Jan. 1992,

[42] UIC/ERRI A200: *Brochure ETCS, European Train Control System, The new standard train control system for the European railways*, Aug. 1993, 2nd. Rev. Oct. 1995

[43] Johannes Feuser, Jan Peleska: *Security in Open Model Software with Hardware Virtualization The Railway Control System Perspective*. Univ. Bremen, 2010 http://opencert.iist.unu.edu/Papers/2010-paper-2-B.pdf