embeddedworld2018
Exhibition&Conference
... it's a smarter world

Certifying Linux: Lessons Learned in Three Years of SIL2LinuxMP

Andreas Platschek
OpenTech EDV Research GmbH
Augasse 21
2193 Bullendorf, AUSTRIA
andi@opentech.at

Nicholas Mc Guire
OSADL eG
Am Neuenheimer Feld 583
D-69120 Heidelberg, GERMANY
hofrat@osadl.org

Lukas Bulwahn
BMW Car IT GmbH
Moosacher Straße 86
80809 Munich, GERMANY
Lukas.Bulwahn@bmw-carit.de

***Abstract*—When the SIL2LinuxMP project was started about three years ago, many non-safety-critical systems using Linux were already built and in operation. Industry designed them in such a way mostly due to the tremendous security capabilities as well as the unmatched support for modern hardware. Both requirements are important for modern industrial applications and can be met using Linux on contemporary multi-core CPUs. However the question whether a safety argumentation for systems based on Linux can be provided and maintained was still open.**

While the ultimate goal of certifying a system based on Linux has still not been achieved as of today, it definitely is in reach for the basic components (Linux, glibc, busybox).

The SIL2LinuxMP project was started as an industrial research project with the goal to find out whether or not it is possible to build complex software-intensive safety-related systems using the Linux operating system as its foundation. During the course of those last years, a number of potential issues that were seen in the early days turned out to be mostly manageable, while other problems took us by surprise. The most striking one being the fact that to this day no certified multi-core CPU (with four or more cores) seems to be available.

This paper not only presents the issues encountered and status achieved during the last three years, it also discusses the approaches currently being proposed to resolve them.

These approaches cover all aspects of the system safety lifecycle. At the system engineering level, we devised appropriate processes to tailor the safety process, moving from a development to a controlled selection process. We developed a layered system hazard analysis to systematically derive adequate safety properties and demonstrated the capabilities of this analysis on a use case. We covered the Linux development process with an assessment for which we devised data mining methods, to quantify software quality, utilizing the available development data. Based on this data, we derived statistical arguments to demonstrate the suitability of the development process. To address residual uncertainty in the area of Linux source code and its assessment, we combined the quality assessments for multiple semi-independent Linux features, capable of mitigating the same systematic fault, into a single safety argumentation. This multilayer handling of residual faults is based on a software layers-of-protection analysis.

These approaches provide the necessary means for a Linux qualification route suitable for up to safety integrity level 2 according to IEC 61508 (SIL2).

Keywords*—*Linux, Safety, Qualification of Pre-Existing Software

I. Introduction

Over recent years, industries have announced new developments that rely on highly complex systems. Examples for this are autonomous vehicles or shared working environments for industrial robots and humans. While these new developments promise great improvements for everyone's private and work life, they have so far mostly been tackled from a functional side. However, apart from the tackle from that functional side, it will be necessary to consider the non-functional properties, such as safety and security as well, before these systems can be used by the general public.

This paper focusses on the safety properties of such highly complex systems and gives insight into the experience that was gained during the first three years of the SIL2LinuxMP [1] project, managed by the Open Source Automation Development Lab (OSADL) and a number of partner companies from various industries.

The goal of the SIL2LinuxMP project is to create a framework that can be used for providing a safety argumentation of a mainline Linux kernel that guides the certification process and reduces the effort of the qualification process as far as possible. We achieve this reduction by automating as much of the process as possible and make the quality assessment of Linux repeatable for the continuously evolving Linux kernel.

In order to verify that the framework is actually usable, we perform the qualification of Linux for a specific use case.

Before investigating solutions, this paper shows the implications from the huge step on the complexity scale, compared to other previously existing safety-critical systems. These implications, discussed in Section II, justify the need for the SIL2LinuxMP project and the (for a safety-critical system) excessively large software stack that it involves.

Then, this paper introduces the most important problems that are tackled in the project and presents the currently intermediate results. We split the discussion of these problems into two sections, depending on whether they have been

anticipated at the beginning of the project (Section III), or whether they were unexpectedly discovered in the course of the project (Section IV).

Note, that while this paper presents the more interesting and novel approaches, a significant part of the work still involves traditional safety engineering activities, which are not mentioned explicitly here.

A. Goals and Common Misunderstandings of SIL2LinuxMP

Before we dive into discussion of the technical aspects in Sections II-IV, we present the goals of the SIL2LinuxMP project. These goals are described here as clear counter statements to re-occuring misconceptions that have caused confusion with discussion partners. This makes clear what to expect and what not to expect and it summarizes how the SIL2LinuxMP project handles the particular issue.

This clarification of goals is essential as it requires a major paradigm shift for companies that are used to buy and treat the operating system as a black box. While Linux comes with the advantage of royalty-free licensing for each deployed system, the costs are shifted towards the building up knowledge about proper safety engineering in the respective companies.

- **Goal:** *Establish a framework that gives guidance on how to build Linux-based safety-related systems.*
  Common Misconception: *Some people perceive that SIL2LinuxMP is creating a product that will be available in a shrink-wrapped package without additional engineering effort.*

  The idea that SIL2LinuxMP creates a packaged product is a very common misunderstanding, and unfortunately one that seems to keep some companies from actively participating in the SIL2LinuxMP project.
  The authors do not think that there can and will ever be a shrink-wrapped package with a Linux version that can be executed on arbitrary hardware without any restrictions and still be used for safety-critical applications.
  This is a dream many seem to have (some try to name that dream SEooC), but unfortunately, this dream is unrealistic for a variety of reasons. The most important one being, that the interface to the Linux kernel is rather big. It is just under 1500 API calls, and this does not even include other interfaces into the kernel, e.g. the proc or sys pseudo filesystems.
  Analyzing it down to the last system call may be doable in theory, but it definitely is not maintainable and thus not economical. At the time of this writing, the list of system calls used in the SIL2LinuxMP use case is about 30-35 API calls (system calls as well as library calls, such as e.g. `memset()`), with additional restrictions on their parameters, i.e., only certain flags are allowed, and on their usage, i.e., some calls may only be used during system initialization. All of these API calls are considered commonly used in most of the existing applications that run on Linux, none of the more esoteric and rarely used calls are used.
  In addition, their usage is restricted to specific combinations, e.g., *allocation of memory* requires a combination of `malloc()`, `mlockall()` and `memset()` and is further only allowed during system initialization.
  Furthermore, we only implement mechanisms to counter-act use-case specific faults that were revealed during the hazard analysis.
  In contrast to the non-implementable care-free shrink-wrapped pre-certified Linux package, the goal of the SIL2LinuxMP project is to create a framework that guides the safety engineer through the process of certifying a system based on Linux.
  That means, it will be necessary to do a specific analysis for every Linux-based safety-critical system. Of course with time progressing, there will be certain patterns that emerge and using Linux in safety-critical systems will become easier, but nevertheless, an operating system of this complexity can never be expected to be resilient against all credible faults in all current and future applications.

- **Goal:** *Qualify Linux as pre-existing software element following Route 3_S in IEC 61508-3 [2].*
  Common Misconception: *Since Linux is widely in use, a Proven-in-use strategy can be done.*

  A proven-in-use argument (named Route 2_S in IEC 61508-3 [2]) seems to be the first naive attempt for everyone who has never thought about the issue of Linux certification before. Unfortunately, proven-in-use qualifies as unusable as soon as one studies the pre-requisites for the collected historic data of such an argument in the relevant standards (e.g. in IEC 61508-7, C.2.10.1 [2]).

  In contrast, the SIL2LinuxMP project provides the safety qualification argument for the pre-existing software elements (Linux kernel, glibc, busybox) following IEC 61508, Route 3_S. This means, we provide an argument explaining why the development process of those pre-existing software elements satisfies the high standards of IEC 61508.
  Nevertheless, the SIL2LinuxMP project makes use of the popularity and wide usage of Linux by considering it in the selection process (see Section III-A for details)—but only as an additional parameter for (de-)selection and not as the sole or main argument.

- **Goal:** *Provide a minimal run-time environment for safety-critical applications up to a level of SIL 2.*
  Common Misconception: *A Full-fledged distribution, e.g. Debian, Yocto, will be available.*

  Unfortunately, running a full-fledged distribution with bells and whistles is not seen as doable (and practical for that matter), as the code base of all packages in a distribution is just too big.
  Therefore the SIL2LinuxMP project is restricted to the Linux kernel, a minimum set of standard libraries and a minimum run-time environment (based on busybox). While some out there may still hope for a SIL2-certified Android or Yocto-based system, including graphics stack and everything—this is certainly not our goal.

Co-location of a SIL0 container in an overall mixed-criticality system is being considered. While this will allow somewhat more non-safe functionality to run in parallel, it also will not provide a full-fledged Linux distribution without any restrictions.
The SIL2LinuxMP approach is to keep those software elements to the QM/SIL0 [1] container where all kinds of non-safety applications can be executed (see Figure 3) and keep the safety-critical application to a bare minimum. Integration of applications with mixed criticality is done using isolation mechanisms in the Linux kernel, as described later in Section III-C.

II. Implications of the Complexity Increase

As already mentioned in the introduction, a significant increase in complexity is happening in many industries. This increase in complexity is driven by the applications that are developed for the near (and not so near) future. We derive a number of implications from these anticipated applications that have a significant impact on non-functional properties of the systems.

1) **Computing Performance –** The performance needs for these highly complex applications is much higher than in traditional systems. This implies not only much higher energy consumption [3], but also that CPUs that are to date not used in industrial applications but, e.g., in the server market, need to be used as otherwise the necessary performance cannot be provided.
2) **Concurrent Computation Capabilities –** The above mentioned need for state-of-the-art processors due to performance requirements also mandates an operating system that is able to manage such modern multi-core CPUs and is able to take advantage of as many performance-enhancing features as possible.
3) **Security –** Another common need across all industries is the need to connect systems to the outside world—often through the internet. This inevitably leads to security issues.

   While SIL2LinuxMP does not focus on security, the project has set itself the goal to check every design decision on the architecture in order to assure that the design does not conflict with generally applied security concepts.

These properties of a computing platform for up-coming high-complexity applications make Linux the prime suspect as the basis for such a computing platform, since Linux is being used in high-performance as well as security-demanding applications for many years. Therefore it provides a number of mechanisms that allow an optimal utilization of the given resources while providing outstanding security capabilities (protection and monitoring).

However, the usage of Linux for safety-critical systems has so far been restricted to some very specific cases [4], [5] but a general approach to certify an unmodified mainline kernel is not available even though discussed in the past [6]. To close this gap and allow the usage of Linux-based computing platforms that satisfy the performance as well as safety needs for future applications, the SIL2LinuxMP project was started.

[1] Without going into detail, please note, that QM/SIL0 does not mean arbitrarily crappy code may be executed in the QM/SIL0 container!

III. Anticipated Research Questions

This section gives a summary on research questions that we anticipated from the start of the project. This does not mean that the solution was fully clear, only that the problems in principle were recognized with some concepts in place on how to tackle them.

A. From Implementation to Selection

The main difference between the SIL2LinuxMP platform and the approach commonly used is that the basic software elements are pre-existing software that have not been subject to dedicated development.

This means that there are no provisions in the lifecycle for fault elimination and this implies a strong concentration on fault mitigation—a fundamentally wrong approach to system safety.

Thus the safety lifecycle is altered in such a way, that it mitigates this flaw by adjustment of the safety lifecycle. Specifically, the V-Model is split into two parts, where the upper part is the system specification and architecture. This part is developed for this particular system and thus done in the regular route 1_S development as defined in IEC 61508 [2]. The bottom part is where usually, the dedicated software would be designed, developed and integrated. Since pre-existing software elements are used, this bottom part is replaced by a software selection process, as shown in Figure 1.

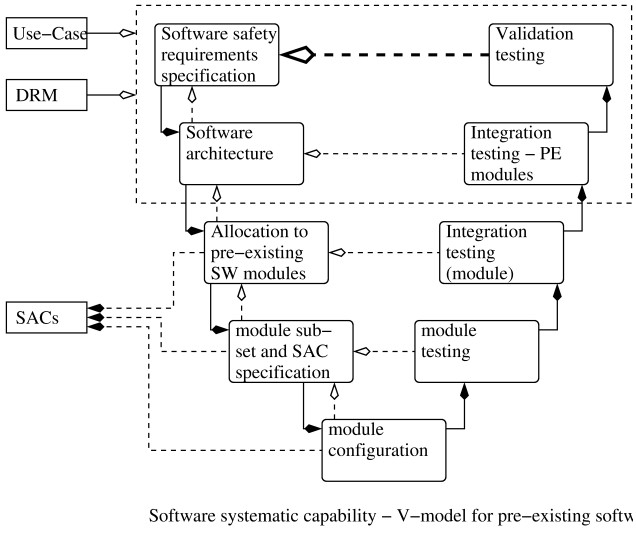

Fig. 1. Safety Lifecycle: Selection Process for Pre-Existing Elements.

This adapted lifecycle model describes the workflow of how elements are selected. Depending on the element, the possible selection items vary. For the elements in the SIL2LinuxMP project, the following variables are up for selection:

- **Kernel Version** – A new stable version of the Linux kernel is released approximately every two months. In

addition about once a year one of these stable versions is made a long-term stable (LTS) version is released. Not every version is as stable as the others, thus an important part of the SIL2LinuxMP projects selection process is to select stable kernels (see Section III-D for details on how to use development data to identify a stable version), that are long-term supported (LTS) and ideally used in a very broad context (e.g. used by major distributions) and thus well tested.

- **Kernel Configuration** – The goal of the SIL2LinuxMP project is not to provide a certificate of the full kernel and allow it to be configured however one might want to, but rather to establish a framework that allows the certification of one specific configuration, the assumption being that this configuration is reduced to functions that are:
  - Needed by the application to satisfy functional or non-functional requirements, while their selection is driven by the maturity and quality of the respective candidate, or
  - Established as de-facto standard, i.e., a kernel configuration without that configuration item would be far away from every other kernel configuration in use.

  Thus the kernel configuration also allows the (de-)selection of certain subsystems based on criteria, such as their development history, novelty of design, size, and known bug rate.
- **Non-Kernel Elements** – In addition to the kernel, other pre-existing elements will be used, e.g., C library, and math library. Usually, different variants for these libraries are available. For these non-kernel elements, the selection process thus starts with the selection of the variant that shall be used. This selection can—amongst the criteria used for the kernel itself—also include the width of deployment and the activity of the development. Using a C library that is only deployed in a few systems and was not maintained for some years just does not make any sense. Instead the advantage of a broad and active community has to be considered by selecting an active and healthy project.

B. Adapting Methods

One issue in many safety-related projects is tailoring methods or arguing entirely new methods. The SIL2LinuxMP project with the outlined selection process of pre-existing software elements (see Section III-A requires that not only the software (i.e. source code) itself has to be considered, but also the development environment including all tools that are used in the development of the pre-existing software elements need to be investigated. This investigation entails an analysis of the contribution to safety by the tools (if any).

Furthermore, the increased complexity requires that methods not suitable for such applications are replaced by state-of-the-art methods that can handle this increased complexity. Unfortunately the methods suggested by standards, such as IEC 61508 [2] are in large parts rather antiquated and proper replacement needs to be argued.

Finally, the application of methods to pre-existing elements also changes the properties of applied established methods, e.g., the specification of an pre-existing element in retrospective can not provide a contribution to *"Freedom from intrinsic specification faults, ..."* as suggested in the measures and techniques properties of Annex C, i.e., 61508-3 Table C.1.

Thus the list of methods that need to be tailored or newly introduced is quite significant and we considered it as good practice to not try and wildly argue methods, but rather build up a systematic process that not only allows the argumentation of tailored as well as new methods, but also shows that original intent of IEC 61508 [2] is addressed with regards to the covered properties by the new set/combination of used methods.

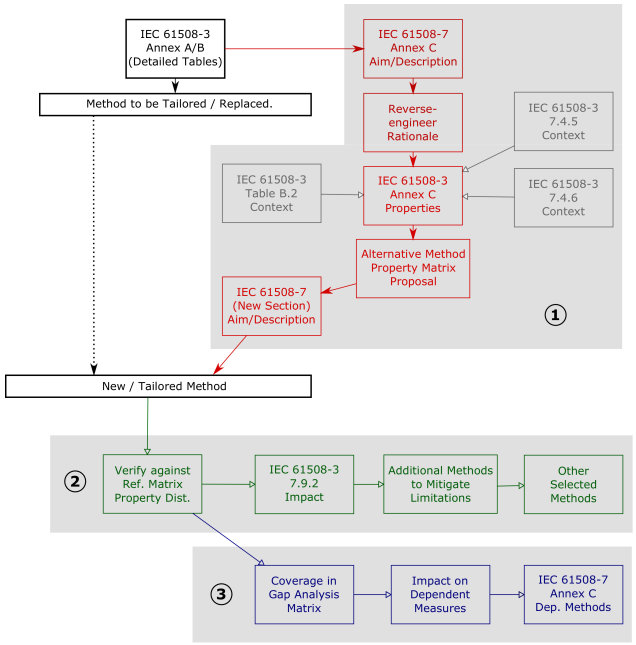

Fig. 2. Example of Tailoring a Method to the Context of Modern Computing Platforms.

In Figure 2, the workflow for tailoring a method is outlined. Basically, it shows the transition (represented by the black arrow with the dotted line) from a current method that shall be tailored/replaced by a new/tailored method.

The approach taken (see the red path going through area ① in Figure 2), is to reverse-engineer the rationale in IEC 61508-7 in order to find out what the intend of the original was. Then IEC 61508-3, Annex C is used to retrieve the properties that the method contributes to the system development. Now the contribution to those properties by the newly introduced (or tailored) method is evaluated.

Next (following the green path going through area ②in Figure 2) the contribution to the properties by the new methods are compared to the contribution of the original method. Note, that at this point, multiple new methods could be used to replace one original method. If this is the case, the contribution of all the (relevant) new methods are compared to the contribution of the original method at least at a semi-quantitative level.

The last step (following the blue path going through area ③

in Figure 2) is to do a gap analysis between the contribution to properties by the new and original method(s) and, if necessary adjust the method set to cover those gaps. This iterative process is done until an appropriate coverage is reached.

C. Isolation Properties

One of the key capabilities of the Linux kernel that are widely used are its isolation mechanisms. Previously, they were mainly used in security-aware systems, but nowadays, with the rise of containers used for simplified system deployment, system adminisation and application development, these isolation mechanism are used virtually in every running system. These isolation mechanisms are used to build containers that are isolated from the rest of the system, in order to make the failure of the contained applications independent of the core system, to limit the impact of a security breach to the container and to assure that unrelated applications can not crash each other or the core system.

The availability of multiple independently designed isolation and protection mechanisms allows to build up layered isolation architectures. Figure 3 illustrates how independent applications of mixed-criticality are isolated, using multiple layers of protection.

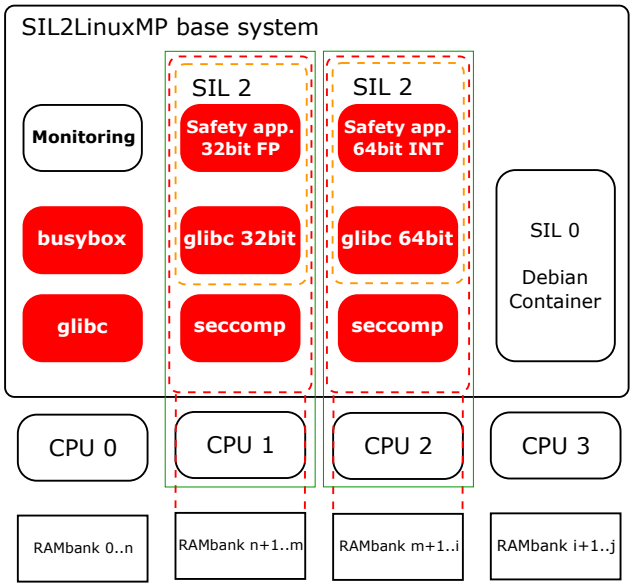

Fig. 3. SIL2LinuxMP – One Possible Architecture

Notably, there are two boundaries on which these isolation properties are used in SIL2LinuxMP:

- to achieve isolation between independent applications (of different criticality), and
- to assure that API constraints specified by the result of the hazard analysis (see Section IV-B) are honored.

While this immediately sounds intriguing—having unrelated applications, and even applications with mixed criticality—without the worry of inter-dependence between those applications, the old problem that appears when investigating the usability for safety, always is **how to verify that this is adequately safe, and that the isolation properties that allow independence of applications failure are trustworthy**.

In this particular case, it was realized that this is a problem that is similar to a problem well-known in the process industry, well formulated by Audrey Canning:

"Yet further concerns relate to whether a consequence can be so severe that the frequency of the hazardous situation should not be taken into account, thus negating the concept for 'risk' in selecting the appropriate set of implementation techniques. In order to address this concern IEC 61511 formalized the concept of 'layers of protection' requiring diversity between the different layers." [7]

The situation in this particular case is similar, in so far, as the risk can not be evaluated because the frequency of the hazardous situation cannot—at least not with reasonable effort—be obtained. For that reason, our solution leans on the basics for layers of protection analysis (LOPA). The intention is to assign multiple layers of protection for each class of hazards. The (usually already very unlikely) event of the hazardous situation will then only happen undetected if all the layers of protection fail at same time, making the event of the hazardous situation even less likely—arguably extremely unlikely.

In order to employ a LOPA and truthfully conclude the previous assertion, the isolation mechanisms satisfy the basic properties of independent protection layers (IPL), cf. [8, Section 1.3]:

- **Independence:** a LOPA only comes to a correct risk assessment, if the protection layers are sufficiently independent from each other. If physical isolation is the target, then this might be problematic with software, therefore the focus is shifted towards logical isolation.
- **Effectiveness:** It needs to be assured that the functionality of the layer protects or mitigates from the studied consequences and works even if the hazard happens. Each layer of protection shall provide sufficient protection or mitigation on its own, the use of multiple layers is meant to add an additional safety net for weaknesses in the argumentation or analysis of the individual layers. This does not mean, that it allows the developers to just not do an analysis, but only refers to situations where the exact quantitative frequency of failure is simply not available or is attached with an uncertainty that is too high for a proper argumentation.
- **Auditability:** It shall be possible to inspect the design and development of the IPLs as well as the IPLs themselves to assure the safety of the individual IPLs. Since the SIL2LinuxMP project uses only free/libre open-source software (FLOSS) the source code of the IPLs themselves is available for analysis, as is a plethora of documentation, and the development history (revision control system, mailing lists, bug reports, etc.).

With this LOPA executed, we show that the isolation mechanisms are sufficient to provide a proper *logical* isolation of the different applications running on the same, shared, Linux-based system.

D. Statistical Modelling

Traditionally, safety-critical software development follows a rigorous development process guided by the relevant standard. The assumption is, that this rigorous development process leads to a residual bug rate that satisfies the targeted SIL level. Alternatively, we can ask the question *What process is responsible for the presence of a software fault?* We anticipate answering this question with statistical methods in SIL2LinuxMP. With other words, we target to provide statistical statements about faults introduced by a stochastic (human) process in the development lifecycle activities.

Traditional safety-related systems had, not too surprising, no means of quantifying systematic faults in software due to the small software size and the statistically low number of iterative re-designs. Instead, a qualitative defense with deeper analysis was considered. Essentially, this pans out to take sets of methods, applying these sets by qualified teams of engineers and wrapping this all into a controlled process for which metrics can serve as indicators of systematic capabilities bestowed on the software elements.

Provided adequate trace data and development metadata is available for such a pre-existing element, we are able to infer adequate process compliance on statistical basis through an indirect metric on the non-compliant development. The basic model is depicted in Figure 4.

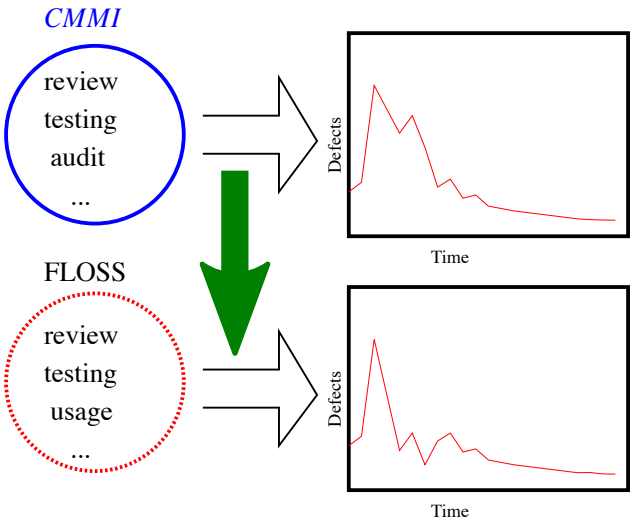

Fig. 4. Principle of statistical modelling using development data.

Establishing process compliance for a highly-complex software element, e.g., the Linux kernel, is a two-step process:

1) Establish the principle existence of mandatory activities, essentially this is what route 3_S in IEC 61508-3 Ed 2 7.4.2.13 a-i encodes, and
2) Establish the actual effectiveness of these methods based on statistical analysis of process metadata.

The Linux kernel developers define a quite rigorous development process [9] which, in principle, can address most of the requirements for a structured and managed process set forth in IEC 61508 Ed 2 part 3. But clearly this FLOSS project lacks the safety management structure to claim any particular rigor on the applied methodology—even though rigor R1 (see IEC 61508-3 Ed 2 Annex C.1.1) *R1: without objective acceptance criteria, or with limited objective acceptance criteria. E.g., black-box testing based on judgement, field trials."* might seem to be calling for very little. If we statistically establish a clear relation between activities called for and effective findings, we can establish a overall claim of principle achievement of the objectives of IEC 61508, specifically the fifth objective of clause 7.4 is being addressed here [2, Part3, 7.4] *The fifth objective of the requirements of this sub-clause is to verify that the requirements for safety-related software (in terms of the required software safety functions and the software systematic capability) have been achieved.*

Such a predictive model is an evaluation of the presumed stable underlying process of development, not an assessment of the systematic faults in a particular version itself. To achieve this, we model successive cycles of the kernel DLC and deduce continuity and trends of improvement for the overall kernel as well as for a particular selected configuration. At the heart, these are regression analysis of the -stable releases for Long-Term-Stable (LTS) kernels using negative-binomial regression models bootstrapped on the development trace data of the Linux kernel.

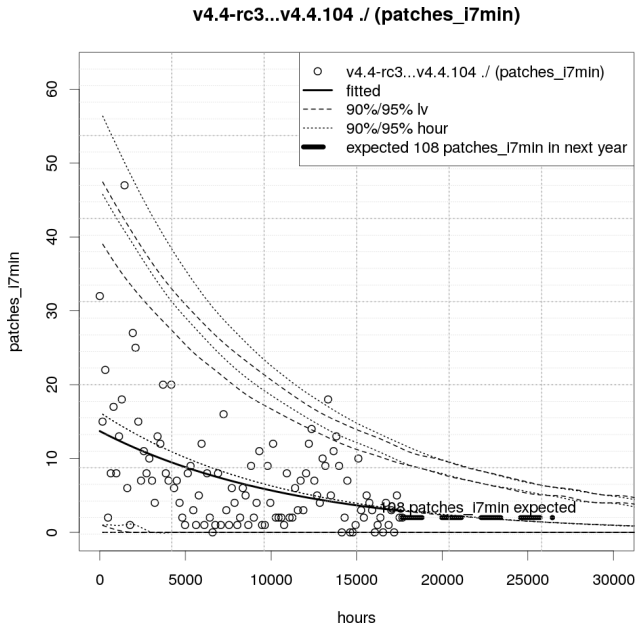

Fig. 5. Linux 4.4 patches development over SUBVERSIONS (Use-Case config)

Modeling the development of patches over stable kernel developments based on development of patches shown in Figure 5 respectively by analysis of the specific hunks of applicable patches applied in Figure 6 in the selected kernel feature set, allows estimating the residual bugs in the kernel as well as an overall judgement of process robustness. The goal of such models is not to imply we know the number of yet-to-be-discovered bugs in the kernel, rather it allows to judge if the development is comparable to a bespoke development, and, equally important, if the rate of reported bugs can be managed in a safety lifecycle or not.

From current, still quite limited set of root-cause analysis

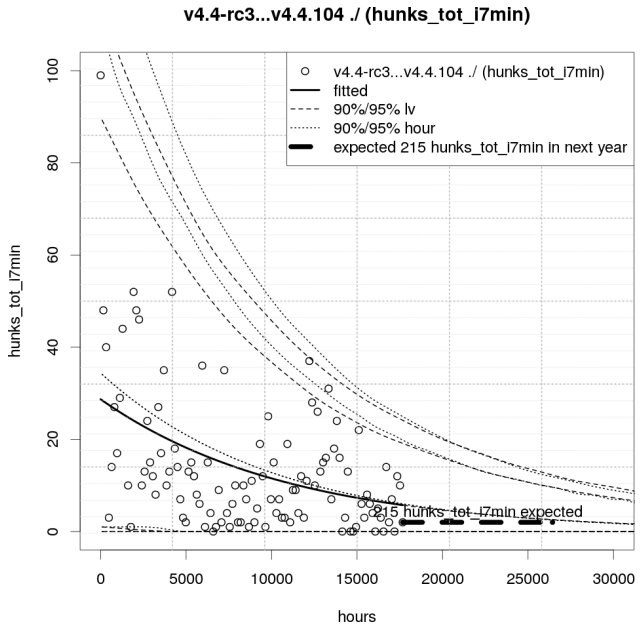

Fig. 6. Linux 4.4 applicable hunks development over SUBVERSIONS (Use-Case config)

data, where bug-fixes in -stable kernels were analyzed, we estimate that $\leq 1/30$ bugs are safety related *for our specific use case* and thus the expected number of bugs is manageable. Regressions though are assuming that bugs are field findings and thus discovered by a time-dependent process, naturally this is not true for all bugs—the recently emerging meltdown and spectra bugs, which led to a very significant updated of critical kernel elements, demonstrates that such predictions have their limits. Nevertheless it is a first quantification of potential impact and thus an important metric for selecting a particular kernel version and configuration selection.

IV. Unexpected Troubles

While the things that were presented above were already known from the beginning, there were some further issues that we did not anticipate at the beginning of SIL2LinuxMP project.

A. Impact Analysis

A question that is raised at discussions quite often is *How do you plan to certify an operating system with 19million lines of code?*. The simple answer is: **This was never the plan.** As already mentioned in Section III-A, the selection of the configuration items is part of the safety development lifecycle. Essentially, the selection is not only a step to eliminate faults, as well as minimize residual faults, but also a step allowing to dramatically reduce the code base.

The Linux kernel configuration takes care that only a fraction of the actual code base is actually used. In addition, every analysis that is done (e.g. the statistical models presented in III-D) ultimately should focus its consideration on those commits that have an impact on the specific configuration.

In order to being able to do this, the SIL2LinuxMP project uses two tools developed in the context of the project:

- The **minimization** [10] tool was developed by Hitachi in the context of the SIL2LinuxMP project. It allows to—based on the kernel configuration—produce a code base where all the code that is not used is stripped based on C-macros used for configuration.
- The **patch impact tester**(PIT) is a tool that is currently still under development (but shows very promising results). In contrast to the minimization tool, it does not work a single version of the kernel, but rather it tests whether a given patch has an impact in a given configuration. This way, the development data of the individual changes is preserved, and the number of commits that have to be considered is reduced.
  While this problem may seem trivial at first, it has quite a number of cases where this is not a trivial problem.
  The PIT itself is based on a GCC plugin. This plugin is used when compiling the kernel and configuration that shall be used. The information provided by this plugin is collected in a database, and can then be used to check whether a given patch has an impact on the configuration, or not.

B. HD^3 – Hazard driven Decomposition, Design and Development

While the complexity of the use case considered in the SIL2LinuxMP project is far below the complexity of intended future applications, e.g., autonomous driving, it became obvious during hazard analysis that this kind of complexity is not controllable with traditional hazard analysis methods. For that reason, a new approach based on the hazard and operability study (HAZOP) method was investigated. This new approach is called Hazard-driven Decomposition, Design and Development (HD^3).

The primary premise of HD^3 is that the design of the system shall be driven by identified hazards. The idea is to use the hazards as design input and use this to eliminate them at the design level, where possible. This way, the need for mitigation mechanisms is minimized, preventing the system complexity to unnecessarily increase.

The general procedure of HD^3 is to start with the basic functionality of the system. In the SIL2LinuxMP use case, this was *"Measure the quality of water."*. Based on this basic functionality, a technology-agnostic process was derived, i.e., the process as if done by a biochemist in a laboratory setup. This technology-agnostic process is then subject to the first hazard analysis. A traditional HAZOP is conducted on the technology-agnostic process, revealing the hazards at this highly abstract level. Elimination conditions and mitigation capabilities are recorded in the form of Safety Application Conditions (SACs) for each of the analysis levels. These SACs are then consolidated into a set of derived items—still at a technology-agnostic level.

The results of the hazard analysis at the technology-agnostic level are used as input for a technology-aware design while still staying technology-unspecific. That means, that an automated system is designed by allocating unspecific devices (motors, pumps, valves, sensors, etc.) to perform the actions

that are performed by the bio-chemist in the technology-agnostic process. At this level no specific devices is yet allocated (e.g. only a "pump" is used without knowing whether it's a diaphragm pump, radial pump, peristaltic pump, etc.).

This technology-aware unspecific design is then fed into yet another hazard analysis. The result of this second round of hazard analysis is used to do a more detailed technology-aware design using specific devices. The important part here is, that based on the hazards at the higher level of abstraction, it was possible to select specific devices that are inherently safe against a number of the specific identified faults. This leaves a limited (ideally minimized) set of hazards that cannot be eliminated this way. Only for this limited set, a mitigation mechanism has to be introduced into the system design to assure a low residual probability of failure.

The actual allocation of mitigations finally can be at the level of the specific safety related application or at the unspecific (generic) level of selected elements (see LOPA).

The result of the second hazard analysis is then used to go into a third level of hazard analysis where the unspecific devices are replaced by the selected specific devices.

It is important to note, that the HD^3 approach is then completed with further intermediate layers of derived requirements that are necessary to get the needed hazard information for the next level of design.

Furthermore, each hazard analysis also emits Safety Application Conditions (SACs) that put conditions on the system that have to be met. The approach of HD^3 results in these SACs showing hierarchical properties, as SACs from higher levels of abstraction map to more fine-grained SACs at the detailed level. For example, at the technology-agnostic level, the SAC *"Critical data must be verified after write."* at the technology-agnostic level is refined as follows: In the unspecific technology-aware level, this is reflected in the form of *"Grey-channel the storage media."*, and in the specific technology-aware level this becomes: *"Written data must be read back."*, *"Individual measurement values shall be times-tamped."*, *"A CRC shall be stored for individual measurement values."*.

This crude example shows how SACs are refined conducting the hazard analysis at the various levels of abstraction. Similarly, the required API manifests itself. What can not be seen in this example is that the SACs and the used API calls that are part of the result constitute

- a minimum set of parameter constrained API calls, and
- a maximum set of constraints.

Thus, reducing the functional subset that needs to be analyzed. This way, the complexity of the software components is handled with a minimized effort.

While the experience with HD^3 is still limited due to its novelty, the first results look very promising and a thorough analysis from the high-level design down to the implementation was possible for the SIL2LinuxMP use case. Maybe a traditional hazard analysis would have been possible for this level of complexity, but from our experience the effort to do so would have been significantly higher, and more importantly, honoring the important rule of "first eliminate, then mitigate" would not have been possible to the same extent.

V. Conclusion

While the goal of a SIL2-certified platform has not been reached within the first three years of the SIL2LinuxMP project, partly due to the lack of certified multi-core CPU hardware, it has been shown that this goal is not out of reach, especially for our main investigation subject, the Linux kernel.

The above sections present the progress that has been made in various parts of the safety lifecycle. The biggest issue in the project's endeavor was to find ways to handle the complexity. This was achieved using the HD^3 method (introduced in Section IV-B) to perform hazard analysis and do the system design. Second, the software LOPA (Section III-C) provides argumentation for the partitioning of the problem by separating applications of same and different criticality and allowing them to be handled separately. Furthermore, the impact analysis, as described in Section IV-A, allows the automatic reduction of the Linux kernel code base to those lines of code that do have a direct impact on the specific configuration in use, thus reducing the effort as everything else can be discarded.

For the re-use of pre-existing open-source elements, the most important steps were the transition from a traditional V-model for software development to the selection process outlined in Section III-A, as well as the systematic process to arguing the use of new methods as well as the tailoring existing methods, discussed in Section III-B.

In summary, the overall progress of the SIL2LinuxMP project is at a point where the authors are confident that the goal can be achieved by completing these sketched activities.

VI. Acknowledgements

The SIL2LinuxMP project is organized by the Open-Source Automation Development Lab (OSADL), as well as the SIL2LinuxMP partner companies. We thank them for their support and their funding.

References

[1] OSADL, *SIL2LinuxMP Webpage*, https://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html, 2016

[2] *IEC 61508 Edition2, Functional safety of electrical/electronic/programmable electronic safety-related systems*IEC, 2010

[3] Bloomberg, *Driverless Cars Giving Engineers a Fuel Economy Headache*, https://www.bloomberg.com/news/articles/2017-10-11/driverless-cars-are-giving-engineers-a-fuel-economy-headache, October 2017

[4] Andreas Gerstinger, Heinz Kantz and Christoph Scherrer, *TAS Control Platform: A Platform for Safety-Critical Railway Applications*, https://publik.tuwien.ac.at/files/PubDat_167529.pdf

[5] Peter Sieverding and Detlef John, *SICAS ECC - die Platform fr Siemens-ESTW fr den Nahverkehr*, Signal und Draht, May 2008

[6] CSE International Limited for the Health and Safety Executive, *RESEARCH REPORT 011: Preliminary assessment of Linux for safety related systems*, 2002

[7] Audrey Canning, *Functional Safety: Where have we come from? Where are we going?* in Proceedings of the Twenty-fifth Safety-critical Systems Symposium, Briston, UK, 2017

[8] *Guidelines for Initiating Events and Independent Protection Layers in Layer of Protection Analysis*, Center for Chemical Process Safety (CCPS), 2015, Published by Wiley&Sons

[9] *A guide to the Kernel Development Process*, https://www.kernel.org/doc/html/latest/process/development-process.html, 2018

[10] *GIT Repository of the Minimization Tool*, https://github.com/Hitachi-India-Pvt-Ltd-RD/minimization, 2017