Making Agile Development Processes fit for V-style Certification Procedures*

Sergio Bezzecchi[2,3], Paolo Crisafulli[3], Charlotte Pichot[2,3], and Burkhart Wolff[1]

[1] LRI, Université Paris Sud, CNRS, Université Paris-Saclay, France
wolff@lri.fr
[2] Alstom, France
Firstname.Lastname@alstomgroup.com
[3] IRT SystemX, France
Firstname.Lastname@irt-systemx.fr

Abstract. We present a process for the development of safety and security critical components in transportation systems targeting a high-level certification (CENELEC 50126/50128, DO 178, CC ISO/IEC 15408).

The process adheres to the objectives of an "agile development" in terms of evolutionary flexibility and continuous improvement. Yet, it enforces the overall coherence of the development artifacts (ranging from proofs over tests to code) by a particular environment (CVCE) .

In particular, the validation process is built around a formal development based on the interactive theorem proving system Isabelle/HOL, by linking the business logic of the application to the operating system model, down to code and concrete hardware models thanks to a series of refinement proofs.

We apply both the process and its support in CVCE to a case-study that comprises a model of an odometric service in a railway-system with its corresponding implementation integrated in seL4 (a secure kernel for which a comprehensive Isabelle development exists). Novel techniques implemented in Isabelle enforce the coherence of semi-formal and formal definitions within to specific certification processes in order to improve their cost-effectiveness .

Keywords: Development Processes, Certification, Formal Methods, Isabelle/HOL, seL4

1 Motivation

Use of formal methods as validation technique for certification of safety and security critical systems is sometimes regarded as counterproductive to industrial development processes, even for having an advantage over competitors. This holds for the railway-industry (following CENELEC 50126/50128), the avionics (DO 178 B/C) or the industry of security critical components (Common Criteria ISO 15408). A major reason for this reluctancy is the perception that these techniques are too complex to apply, require high-skilled contributors and therefore is time-consumming and not well mastered. This contributes to the fact that regulators speak of a "certification crisis" [10] which, in the case of CC 15408, is reflected by only a handful EAL7 (level requiring formal methods) certifications after 25 years of the standards existence...

* This research work has been carried out in the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the Program "Investissements d'Avenir".

Agile Development. Agile processes have gained substantial popularity among developers because of their flexibility. It is instructing to consider their objectives, such as evolutionary, distributed development and continuous build.

For safety-critical systems development, rework is often practiced: this costs a lot and can bring inconsistency. Defining an agile process, adapted for rework and impact analysis, compliant with a V-cycle will solve this issue.

Certification procedures. CENELEC 50126/50128, DO 178, CC 15408 alltogether require a number of documents which are evaluated in a particular order and establishing traceability between these documents whose formats are prescribed in templates. Missing links, revisions, backtracks and inconsistency lead to augmented efforts and costs during the certification .

All these certification processes recommend or mandate the use of *formal methods*, whether for modeling or for proof. [4] For short, a development process targeting certification has the following particularities:

- a relatively high and certification-level dependent degree of formality
- pervasive, comprehensive traceability of requirements, environment hypotheses, etc. throughout all artifacts, and
- perfect reproducibility of all artifacts.

A key-observation for our work is that it is common sense not to enter certification procedures too early, which results in a separation of *development* and *validation*. It is our aim to enable for both a distributed, "agile" *process* based on a strong division of labor and a fast tool-supported impact analysis, as long as at any time the coherence of all artifacts can be assured.

2 A Development Process and its Support in CVCE

The presentation of a CENELEC certification process is best described with the following, V-style process scheme: A CENELEC certification requires a number of key-phases: *Requirements Definition*, the *Design*, the *Coding*, *Test* and *Commissioning* phase, each accompanied with a *Validation/Independent Safety Assessment* phase. Traditionally, the latter phases are separated from the former (as the standard requires organizational independence); however, we propose to keep them closely together in order to improve agility and to ensure organizational independence by technical means. This also applies for the accompanying documents.

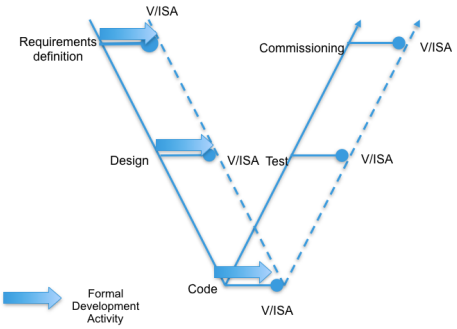

Fig. 1: CENELEC certification W-schema

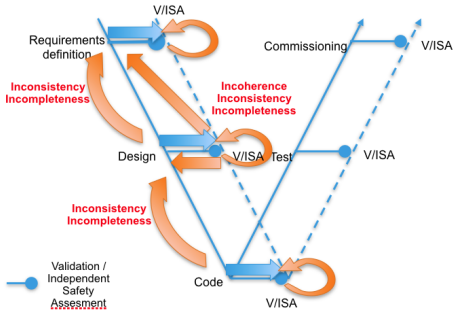

Fig. 2: CENELEC development in CVCE

[4] CENELEC EN 50128:2011 mentions in Annex D.28 CSP, HOL, Temporal Logic, and B, etc.

This paves the way for a strong automated impact analysis for changes during the development connecting both sides rather than separating them. In section 4, we will demonstrate the transition accross one layer of this diagram — the transition from a requirements definition to its V/ISA counterpart depicted by the blue arrow — using a number of tool-supported techniques, ranging from "literate specification" over validation of definitions to finally proofs and tests in order to gain and demonstrate confidence over the system and its models. In current certification processes reworking of visited phases (Figure 2) takes months and is a major cost factor; during certification, this usually happens only due to change requests of an evaluator. In our process, however, it is possible to modify these documents on a daily basis, even in a distributed manner. This results in a cultural shift of the development team, now following the motto of agile development "embrace change". Furthermore, we complement it by the motto "embrace formality" as a means to make the overall coherence machine-checkable, and to **keep the feedback time of the impact of modifications short**.

Of course, tool-support for such a process is vital. The proposed *Continuous Verification and Certification Environment(CVCE)* is basically a tool-chain built around Isabelle/HOL[7]. As a result, the development process benefits from agile and formal features of CVCE.

Isabelle is nowadays best described as a general system framework providing a programming environment, code and documentation generators as well as a powerful IDE, comprising an own session and build management. It allows extensions of the core with plugin concepts roughly similar to Eclipse. Isabelle/HOL is such a plugin that supports modeling, code-generation as well as automated and interactive theorem proving for Higher-Order Logic (HOL). For HOL, plugins such as HOL/AutoCorres or HOL/TestGen have been developed for code-verification of C programs or for model-based test-generation, respectively.

Moreover, Isabelle is used as a central tool for the entire project document generation; particular setups have been developed by the authors in order to mark the different items of a certification (requirement, assumption, test-case, justification, ...) and their evaluation results (validated proofs or tests, for example). These markers along with associated traceability have been implemented by Isabelle's concept of antiquotations[13] and are supported directly in the IDE, enabling direct checking of all types of links directly when editing and before document generation, which can take considerable time. The validation mechanism of Isabelle have been extended by the authors by a particular generic "ontology support" which has been instantiated for CENELEC; this can be seen as an validation-checker for semi-formal content of the document imposing a particular syntactic structure of the overall documents and enforcing a consistent use of links between the different documentation parts.

3 Features of CVCE and its Benefits for the Development Process

Version vs. Acces Control. The core technologies which ensure modeling, proving, and coherence management, were integrated into pervasive version management (in our case implemented via Git (c. f. `https://git-scm.com/`). Even for the early phases we encourage the versioning of notes (possibly complemented by sketches and, eg., photos from blackboards) as a text-basis to be improved during the process.

Incrementality by Gradual Improvement. Support of gradual improvements with respect to the progression of text quality, degree of formality, degree of confidence, executability, testability, efficiency and finally document coherence wrt. to a standard are of vital importance for CVCE. We advocate techniques for model validation, metrics to measure confidence, and strengthening coherence by a transition from liberal to more and more constraining document ontologies during the process.

Global Document, Information Filtering and Retrieval. CVCE accomodates the entire collection of primary and generated artifacts as part of a *global* (versioned) document

containing mutual links and coherence constraints to be taken care of. We present a number of techniques to browse and filter formal content, and to use meta-information to produce stake-holder specific "views" of the global document.

Impact Analysis on Local and Global Documents. A change somewhere in the global document will raise the inconsistencies/incoherences. The more formal and semi-formal content has been integrated, the finer the grid will be to trace problems as a result of change. Isabelle offers a particular form of fine-grained parallelism [11,12] that allows for larger portions of the global document (so-called *sessions*) to produce fast feedback for changes (within the limits of the document structure, computing complexity and computing power).

Continuous Build. It may be necessary to structure the Isabelle documents into several components (called *sessions*) and to rebuild them periodically in order to maintain agility during their development. We implemented this side of large-scale continuous rebuild by a particular configuration of Jenkins (c. f. `https://jenkins.io/`). Continuous rebuild of components increases both the enforcement of verifiation and validation processes as well as the development speed itself, by direct reuse of pre-compiled Isabelle sessions from the Jenkins server.

Advanced Configuration Management. We suggest an integrated configuration management based on Docker (c. f. `https://www.docker.com/`) which allows an abstraction from the OS configurations including different versions of script-interpreters, compilers, simulators, etc. "Dockerization" of the entire environment also facilitates the empowerment of various team members to execute and simulate low-level artifacts for critical cases whenever they are detected. In our case study, we greatly profited from the fact that the seL4-project provides already a dockerized verification, build and test environment for both the code-generation as well as the code-verification step.

4 The Odometry Case Study

In this section, we demonstrate the development techniques within CVCE by an example drawn from a major case-study, the Odometry Subsystem of a train converting sensor data into safety-critical information.

Due to space limitations, we will concentrate only on one particular slice of the development, the transition from *Requirements Definition* to *V/ISA* (this corresponds to the topmost left blue arrow in Figure 1, which is now decomposed into a series of different techniques structuring this transition). The combined document is called *Requirements Analysis* (or: `Odo_ReqAna`) in our case study.

The "scaling up" of our business logic to a *subsystem* (comprising also operating system and hardware) is described in the next section — this scope of our case study is typical for the embedded systems domain.

In the rest of this section, the involved formalization techniques are highlighted in boxes and meta-level commentaries are displayed in ordinary font.

Early Phase: Capture of Requirements and Definitions

Mechanisms: Isabelle structuring commands `chapter`, `section`, `text` using markers.

The frame below contains an extract of the original specification of our case study, with Isabelle structuring commands highlighted. This activity — the capture of requirements definitions — can be done by system-engineers and domain experts with no Isabelle knowledge.

chapter The Odometric Function

section Introduction

text *The proposed use case comprises two services:*

– *Odometrics module, which processes the signals issued by an incremental shaft encoder attached to a bogie's axle, producing a real-time estimation of the train's progress.*
– *Kinematics module, which calculates:*
1. *the train's relative position, and*
2. *the train's absolute speed, acceleration and jerk.* [...]

subsection General Assumptions

text *For the purpose of this study, we assume*

– *the train's wheel profile is perfectly circular, with a given, constant radius,*
– *negligible slip between the train's wheel (to which the shaft encoder is installed) and the track,*
– *the shaft encoder's path between teeth is the same and constant, and* [...]
– *the sampling rate of the encoder's input is a given constant, fast enough to avoid missing codes.*

section The Odometric Subsystem

text *We call* `tpw` *the number of teeth per wheelturn.*
The proposed incremental encoder provides cyclical outputs when its shaft is rotated, at a pace of tpw counts per revolution. To produce a sound value, the encoder has three outputs, called C1, C2 and C3, which are 120 degrees out of Phase. *Each tooth is read by the 3 sensors, each with the corresponding shift. Each sensor output can present a logical value of 0 or 1.* [...]

subsection Additional Encoder Properties

text *The geometrical construction of the encoder ensures the following relationships representing information redundancy allowing to detect faults at the physical aspect of the odometer.*

– C1 & C2 & C3 = 0 *(bitwise logical AND operation)*
– C1 | C2 | C3 = 1 *(bitwise logical OR operation)*

[...]

subsubsection Precision of Calculations

text *The resolution of time, distance, speed and acceleration data, in International System Units, shall be:*

– *Time:* 10^{-2}s *the resolution needed for calculation.*
– *Distance:* 10^{-3}m *(i.e. 1mm)*
– *Speed:* 1.3 x 10^{-3}m/s (i.e. 0.005 km/h)
– *Acceleration:* 0.005m/s^2
– *Jerk:* 0.005m/s^3

The intended accuracy shall be propagated throughout internal calculations so as to insure that the output data respects the specified resolution.

Early Phase Formalization of Key Notions

Mechanisms: Isabelle specification constructs `definition`, `fun`

The anaysis of the previous text reveals that Integers and machine representation of integer ("unsigned integer 32 bits") play a major role for the formal arguments in this problem domain. Consequently, we base this document on a logical context supported by libraries for machine-words (`Word.thy`) and the standard library of Isabelle/HOL called `Main`.

We can now start to enrich the informal text sections by formal definitions; for example:

```
record shaft_encoder_state = C1 :: bool  C2 :: bool  C3 :: bool
```

defining an input-type of the odometer as a triple of boolean values C1, C2, and C3. The informally mentioned Phase function maps position codes into this triplet; we proceed by recursive definition:

```
fun phase₀ :: "nat →shaft_encoder_state" where
      "phase₀ (0) =      (| C1 = False, C2 = False, C3 = True |)"
     |"phase₀ (1) =      (| C1 = True, C2 = False, C3 = True |)"
     |"phase₀ (2) =      (| C1 = True, C2 = False, C3 = False|)"
     |"phase₀ (3) =      (| C1 = True, C2 = True, C3 = False|)"
     |"phase₀ (4) =      (| C1 = False, C2 = True, C3 = False|)"
     |"phase₀ (5) =      (| C1 = False, C2 = True, C3 = True |)"
     |"phase₀ x  =       phase₀(x − 6)"

definition Phase :: "nat → shaft_encoder_state"
where  "Phase (x) = phase₀ (x−1) "
```

Gaining Confidence by Validation

Mechanisms: value, **assert**, and code-generation.

Once stated, definitions can be in most cases immediately used in validation commands that execute them on ground values (no variables), in a way that is similar to OCaml or SML command shells. We recommend this form of validation as early as possible in order to gain confidence in the given definitions. For example:

```
value "Phase 7"
assert "Phase 1 = (|C1 = False, C2 = False, C3 = True|)"
```

where the first command just attempts to evaluate the given expression and presents the result in the Isabelle output window. The assert command checks additionally that the result is true; otherwise the command fails which leads to an error-message in the interactive mode of Isabelle and a build-failure in batch-mode checks of CVCE. In particular the **assert**-command is useful to document corner cases of definitions early.

The resulting document of this activity is what we call a "formalized requirements definition". This activity can be done by system-engineers, domain experts, and programmers with some general mathematical knowledge and functional programming skills.

Strengthening Formal Content in Informal Parts

Mechanisms: Isabelle Antiquotations `@{const ...}`, `@{term ...}`, `@{type ...}`, `@{thm ...}`, `@{value ...}`, `@{file ...}`

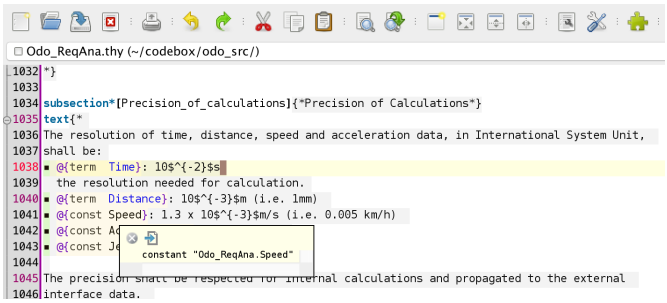

Fig. 3: Giving Text a Formal Status

Figure 3 presents a screenshot of a piece of text from the requirements definition that has been "truffled" with antiquotations. Providing this kind of meta-information is a common technique in typesetting systems; however, Isabelle allows for *semantic* checks wrt. formal definitions and evaluations conform to them.

Gaining Confidence by Theory Development

Mechanisms: **lemma**, **theorem**, and interactive proof.

We proceed with the *requiments analysis* of the *requirements definition*. For example, the encoder properties stated in section 3 can be now proven formally; methodologically, this can be seen as a proof obligation stated from the requirements definition team and discharged by the verification team. The reformulation of the "properties" and their proof looks as follows:

```
lemma Encoder_Property_1: "(C1(Phase x) ∧C2(Phase x) ∧C3(Phase x)) = False"
  proof (cases x)
    case 0 then show ?thesis by (simp add: Phase_def)
  next
    case (Suc n) then show ?thesis
         by(simp add: Phase_def,rule_tac n = n in cycle_case_split,simp_all)
  qed
```

Acknowledging that the theory of the core definitions can be a quite substantial amount of text, we still advise to present it textually close to the corresponding definitions and validations. This principle "establish the theory of a definition early" results from the global objective to keep documents integrated and to avoid document separations (even between requirements definition team and V&V team) in order to improve communication and speed up impact analysis under change.

The most substantial safety proof done in the requirements analysis part of the odometry case study is a proof that for given configuration parameters tpw, w_d (wheel diameter), a given class of normally_behaved_distance_functions *df* (assuming boundaries on speed and accelleration), there is a minimal sampling frequency that the odometric measurements must assure in order not to miss codes in a sampling sequence. The property is stated as follows (the proof requires a large number of details that cannot be presented here):

```
theorem no_loss_by_sampling :
  assumes ∗ : "normally_behaved_distance_function df"
    and ∗∗ : "δ_odo ∗ Speed_Max < δs_odo"
  shows "∀ δt≤δ_odo. 0<δt ⟶(∃f::nat⇒nat. retracting f ∧
                                sampling df init_enc_pos δt = (sampling df init_enc_pos δt_odo) o f)"
```

where sampling is defined in terms of an encoding sequence:

definition sampling:: "distance_function ⇒ nat ⇒ real ⇒ nat ⇒ shaft_encoder_state"
where "sampling df init$_{enc_pos}$ sample$_{itvl}$ ≡λn. encoding df init$_{enc_pos}$ (n ∗ sample$_{itvl}$)"

In particular the assumption ∗∗ establishes a requirement on the minimal sampling time-interval δt_{odo} that is actually also a constraint on the minimal speed of the calculations to be executed on the hardware. This type of assumption — called a *safety related application condition* (or: *srac*) in CENELEC terminology — must be tracked throughout the certification and finally validated by hardware tests.

Adding Ontological Meta-Information and Ontological Links

Mechanisms: section∗, **text**∗, etc, and ontological antiquotations.

We added an own module to Isabelle that allows the definition of an ontology imposed by a certification standard. Due to space limitations, we can not present it in detail; however, for the sake of this paper, it is sufficient to view ontologies as a kind of document type definition (dtd) known from XML. *Ontological classes* are, similar to document types in XML or classes in object-oriented programming, organized in an inheritance relation representing the ontological "is_a" subrelation. They can have attributes (like: "status" of a "srac", its "owner" in organisational terms, etc), which are in our framework fully typed in contrast to XML. Ontological classes may have *instances*, i. e. links which may be the building blocks to annotate text entities in Isabelle documents.

For example, the declaration of a text as a *srac* is done by a family of variants of the Isabelle/Isar standard commands. These variants chapter∗, section∗, **text**∗, etc., implemented in our ontology support, accept this type of meta-information in an additional parameter where the first parameter is the label representing the link to the ontological class instance; this label must be unique. The application document reference in our integrated document

```
814 text*[enough_samples::srac]{* Note that the theorem above establishes a constraint between
815 @{consts w_circ}, @{consts tpw} , @{consts Speed_Max} and  sample_frequency; since this
816 exported constraint is fundamental for the safe functioning of  odometer and therefore
817 a safety-related exported application constraint. It is formally expressed as follows:
818 *}
819
```

Fig. 4: A declaration of a text block as a CENELEC "srac"

is shown in Figure 5: The astute reader may notice that we reference the "srac" as an *exported*

```
822
823 text{* Summing up, the property that the odometer provides sufficient sampling
824 precision --- meaning no wheel encodings were ``lost'' compared to any sampling done with
825 a higher sampling rate --- can be established under the set of general hypothesis captured
826 in @{docref ‹general_hyps›} (formally expressed in @{thm normally_behaved_distance_function_def})
827 and the SRAC @{ec ‹enough_samples›} formally expressed by @{thm srac_1_def}. *}
828
```

Fig. 5: An application of a "srac" ontological reference.

constraint (or :*ec*) consistent with the "is_a" subrelation defined in our CENELEC ontology. Checking the link consistency and jumping to the corresponding text element is done by just a mouse-click in the Isabelle IDE.

5 The Odometry-Service Study on Top of seL4

In order to demonstrate that our method and tool chain CVCE *scales up* to subsystems, not just some module in C of finally relatively modest size, we integrated the entire theory architecture of seL4 (developed as an open-source by the Australian research group NICTA, see [4]) and integrated the odometry module as a safety critical component on top of it.

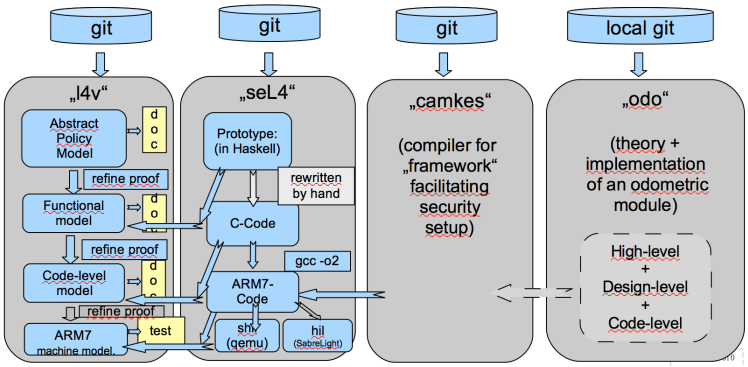

Fig. 6: The CVCE-instance for the Odometry Case Study

Figure 6 shows the major components of the CVCE instance for Odo. On the left side, the process "l4v" is the main validation process, which is running as a continuous build activity. It chains basically the refinement proofs going from the security model of seL4 down to the implementation in ARM 7 assembler code which is either executed on a QEMU emulator or on a Sabre Light board. The entire proof stack comprises 200 theories with about 200000 lines of code (loc). "seL4" is the classical compilation process that compiles the C code of seL4; it compiles under gcc -o2 (the generated code is checked inside l4v). seL4 comprises about 8000 loc and provides the basic functionality of an OS system kernel of the L4 family (like PikeOS) enhanced with very strong security mechanisms apt to ensure process separation. The C-code is also compiled into a model of the C-Code (including, among many other things, the Memory Model of the C execution) which is proven correct against its contracts given in pre- post- condition style; these contracts are linked via refinements to the design model on the one hand and the assembler code on the other. "camkes" is a small component framework “glueing” components together and integrate them on top of seL4 OS. The resulting module “odo” is such a component providing the business logic of the subsystem.

6 Conclusion

We have shown a software development method and a tool chain called CVCE targeting high-level certifications for safety and/or security critical systems. The method has been demonstrated on a case study, the development of the odometric subsystem as used in the railway domain. From high-level formal system modeling till code, the different links were formally proven or extensively tested; the result is to be run on the seL4 platform which has been verified down to assembly code by the seL4 project. To our knowledge, this capacity of Isabelle/HOL for comprehensive verification, made realistic by reusing substantial parts of the Isabelle/HOL community contributions, is a unique capacity of this verification framework.

The case study on the odometric subsystem proceeds through the classical steps: Requirements Analysis, Design Analysis, and Code(+Verification), where the key functions can be seen each as a kind of refinement from another.

We highlight the main results (the second and third were not presented here in detail):

1. **Requirements Analysis:** Establishment of the dictionary of the physical system, the principles of sampling into encoder sequences, and the interface of the module. The main theorem establishes conditions under which the sampling can be accurate in principle.
2. **Design Analysis:** A computable definition for the odo_{step} function which is the heart of the odometric module calculations. The main theorem establishes that odo_{step} indeed approximates distance, speed and acceleration in its calculations assuming a rational arithmetic with unlimited precision. odo_{step} is converted into executable code as a reference for precision tests.
3. **Code Analysis:** We provide a handwritten C function and verify it via the C-to-HOL compiler in the Isabelle/AutoCorres module against odo_{step}. The main theorem establishes that the C-level calculations done on bounded machine arithmetics indeed approximate the calculations of odo_{step} under certain conditions.

This paper focuses on a methodological aspect: the method attempts to reconcile the objectives of agile software development with the needs of classical, distributed structured software engineering. The sharpest contrast to common understanding of agile development is that we embrace documentation and formality, as well as upfront efforts like requirements analysis and design before coding, in order to provide the technical means for fast impact analysis and machine-checkable coherence. We agree with mainstream agile development on the importance of early testing and validation, but extend this to the level of requirements and design definitions and complement it, where necessary, with interactive and automated proof efforts. The current verification stack is, however, not a complete verification; due to limited ressources, we adopted a strategy to concentrate on the most critical parts.

6.1 Related Work

There is a growing interest in combining agile and formal methods, reflected by a number of workshops addressing this combination [2,1] . A number of works emphasize the value of formal techniques inside agile development in particular wrt. *automated test generation techniques* from models (see [3], or [9,8]) While we fully adhere to this idea (and applied the verified test vector generator technique ourselves in our case study), we argue that the scope of formal method application is much wider and covers in particular — via ontology support — aspects of linking semi-formal with formal content.

Already in 2010, the combination of formal and agile methods was investigated in [5], who came to a merely negative view. In our view, this is partly because the authors understandibly identify Agile Methods with its Manifesto and anticipated part of the criticism of [6]. We follow the latter in its distinction between principles and practices of agile development, where we adhere to the former, but not the latter.

6.2 Known Limitations of CVCE

The current environment has still a number of limitations:

- CVCE and its notion of "integrated document" is currently solely text based; diagrammatic notations as common in UML are simply not available. So far, we favored textual documents since we crucially depend on globally available merging and conflict resolution mechanisms.
- The PDF document generation via LaTeX is relatively slow since it is part of the post-processing of global checking. While we added a lot of IDE support to circumvent PDF previewing, more light-weight feed-back wrt. printable versions is highly desirable.
- Access control on individual parts of documents (so: text-parts or formal definitions) has not been a priority so far; however, it may be useful when scaling up to larger developments.

Recent developments of an Isabelle/PIDE Interface based on Visual Studio Code paves the way to integrate Markdown-LaTeX plugins offering fast preview on theory presentations. This may help to overcome the first two limitations soon.

References

1. FormSERA '12: Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches. IEEE Press, Piscataway, NJ, USA (2012), iEEE Catalog Number: CFP1286S-ART
2. Gruner, S.: Fm+am'09: workshop on formal methods and agile methods. Innovations in Systems and Software Engineering 6(1), 135–136 (Mar 2010), https://doi.org/10.1007/s11334-009-0101-8
3. Haehnle, R.: Agile formal methods. Keynote at the Key Symposiiun (2007), http://i12www.iti.uni-karlsruhe.de/key/keysymposium07/slides/haehnle-agile.pdf
4. Klein, G., Andronick, J., Elphinstone, K., Murray, T., Sewell, T., Kolanski, R., Heiser, G.: Comprehensive formal verification of an os microkernel. ACM Trans. Comput. Syst. 32(1), 2:1–2:70 (Feb 2014), http://doi.acm.org/10.1145/2560537
5. Larsen, P.G., Fitzgerald, J.S., Wolff, S.: Are formal methods ready for agility? A reality check. In: FM+AM 2010 - Second International Workshop on Formal Methods and Agile Methods, 17 September 2010, Pisa (Italy). pp. 13–25 (2010), http://subs.emis.de/LNI/Proceedings/Proceedings179/article6226.html
6. Meyer, B.: Agile! - The Good, the Hype and the Ugly. Springer (2014), https://doi.org/10.1007/978-3-319-05155-0
7. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002), https://doi.org/10.1007/3-540-45949-9
8. Rumpe, B.: Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring. Xpert.press, Springer Berlin Heidelberg (2012), https://books.google.fr/books?id=bcIoBAAAQBAJ
9. Rumpe, B.: Agile Modeling with the UML, pp. 297–309. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), https://doi.org/10.1007/978-3-540-24626-8_21
10. Tzafalias, A.E.P.O.: Podiumsdiscussion at the workshop on security certification of ict products (March 16 2016), https://www.enisa.europa.eu/activities/Resilience-and-CIIP/workshops-1/2016/ict-security-certification-for-industry
11. Wenzel, M.: Shared-memory multiprocessing for interactive theorem proving. In: Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings. pp. 418–434 (2013), https://doi.org/10.1007/978-3-642-39634-2_30
12. Wenzel, M.: Asynchronous user interaction and tool integration in isabelle/pide. In: Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. pp. 515–530 (2014), https://doi.org/10.1007/978-3-319-08970-6_33
13. Wenzel, M.: The isabelle/isar reference manual — isabelle version 2016-1 (March 16 2016), http://isabelle.in.tum.de/documentation.html