Formal Requirements in an Informal World

Daniel Dietsch, Vincent Langenfeld, Bernd Westphal
Department of Computer Science, University of Freiburg
Email: {dietsch,langenfv,westphal}@informatik.uni-freiburg.de

***Abstract*—With today's increasing complexity of systems and requirements there is a need for formal analysis of requirements. Although there exist several formal requirements description languages and corresponding analysis tools that target an industrial audience, there is a large gap between the form of requirements and the training in formal methods available in industry today, and the form of requirements and the knowledge that is necessary to successfully operate the analysis tools.**

We propose a process to bridge the gap between customer requirements and formal analysis. The process is designed to support in-house formalisation and analysis as well as formalisation and analysis as a service provided by a third party. The basic idea is that we obtain dependability and comprehensibility by assuming a senior formal requirements engineer who prepares the requirements and later interprets the analysis results in tandem with the client. We obtain scalability as most of the formalisation and analysis is supposed to be conducted by junior formal requirements engineers.

In this paper, we define and analyse the process and report on experience from different instantiations, where the process was well received by customers.

I. Introduction

The quality of requirements is crucial to the development of systems and software, as there is a risk that defects introduced in the requirements analysis stage are reproduced as part of the (then, with regards to the requirements correctly) implemented product. Requirements analysis tools promise to find defects and other violations of well-formedness properties in a set of requirements. Recently, the analysis of formalised requirements has gained much attention in industrial contexts (e.g. [14], [4], [6], [18]). In these works, languages for the formalisation and tools for the analysis of formal requirements are presented (partly together with case-studies in specific contexts), but two important problems remain largely unsolved: How to obtain *formal* requirements and how to integrate analysis tools into generic software development processes.

In requirements engineering, there is the distinction between a requirement being a condition or capability of a system (in short) and its representation. Today, the predominant form of representing requirements is natural language (in that sense, we face an informal world) while formal requirements analysis tools need *formal* representations. Requirements engineers can in general not be expected to be trained in formal methods, so the question is how experts in formal methods can complement an informal requirements engineering process.

Regarding the use of tools, we can learn from the introduction of automatic formal program analysis in industrial contexts [3] that complex formal tools need well trained personnel for effective use; a lack thereof may lead to useful tools being rejected because clients are, e.g., unsure about the interpretation of analysis results. Hence formalisation and analysis tools have to be embedded into a process. A process guides who (role) does what (activity) depending on and producing what (artefact). Defined processes prevent us from forgetting or needlessly repeating things, and provides a clear plan. Roles allow to define the competences that a certain position demands, and to assign developers with the right skill set. Activities and artefacts allow us to identify interfaces, analyse risks, and propose mitigations.

The ideal process for formal requirements, would be able to formalise and validate all requirements for all relevant properties. In reality, there is no upper bound on the time it takes to create 'the perfect formalisation' of a single requirement. Validation alone, i.e., the confirmation that a formal description correctly describes the considered condition or capability, can be prohibitively expensive. In our perception, a realistic process has to work for as many (important, relevant) requirements as possible with as valid as possible formalisations. In other words, the goal is to arrive at the best requirements possible, given all constraints. Regardless of the particular formalisms and tools, a process for formal requirements applicable in a larger project context, needs to be budgetable, comprehensible to the stakeholders, and the outcome should be clearly defined.

In this work, we describe the Dietsch-Langenfeld-process model for formal requirements engineering that proposes to bring formalisation and formal analysis to scale and to a budget. The process is supposed to connect to a setting where all requirements elicitation and informal description is done and the only remaining gap is the formalisation and analysis. One feature by which the process gets attractive is the proposal to tolerate *false negatives* and thereby mitigate the risk of the hardly calculable perfect validation (see above). The proposed process can be executed by external entities so that no additional training is required on the customer's side, where a customer in the following is understood as a requirements engineer who only lacks formal methods expertise.

Moitra et al. [8], [14] have recently presented the requirements analysis tool ASSERT. They also see the gap between engineers and formal requirements, so they designed their requirements language for the tool to look familiar to engineers in their industrial domain and give explanatory tool outputs. The authors briefly explain the steps of formalisation in [15], yet they do not provide a process model. This may be partly due to their tool being used in-house, in contrast to our approach, where it is critical to have defined roles and deliverables.

The third author was supported by the DFG, reference no. WE 6198/1-1.

978-1-7281-8358-9/20/$31.00 ©2020 IEEE
DOI 10.1109/FORMREQ51202.2020.00010

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.

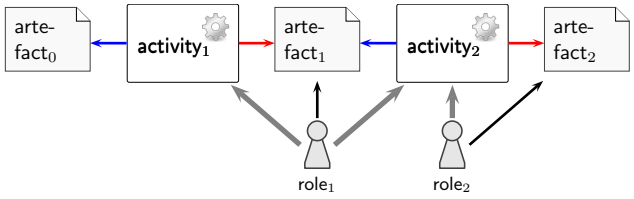

Fig. 1. Process modelling notation.

In [7], Cimatti et al. lay out a formal methods based approach to requirements validation that comes closer to a process model. The process starts with a natural language system description (from Wikipedia), which is decomposed and formalised into UML and a formal annotation language, with subsequent analysis. While the approach described is very broad, our process model can be seen as refining and concretising one activity in their work, namely the step from structured requirements to a formalisation and analysis.

In [5], Brill et al. demonstrate the use of the (then new) visual formalism Life Sequence Charts (LSCs) for requirements from the railway domain. The whole approach is embedded into the V-Model [21] where the whole development process is accompanied by formal models, (later also demonstrated in [9] on another railway application). While this approach covers the complete system development, the relation of the initial model to requirements is just assumed, in contrast to the client feedback that is used in our approach.

II. Preliminaries

A. Process Modelling

Following, e.g., the V-Model XT [21], a process model consists of *activities*, *artefacts*, and *roles*. An artefact is any kind of product occurring in a software engineering project such as documents, software, hardware etc. Artefacts are *created* or *modified* by activities, and activities may *depend* on other artefacts as inputs. Roles *participate* in activities and include capabilities, rights, and responsibilities wrt. the activity. Persons with the necessary capabilities are assigned to roles to participate in activities of the process. A role can be usually filled by multiple persons. Each artefact has exactly one role, that is *responsible* for the artefact. Artefacts may further have a *product state model*. In the V-Model XT [21], for example, an artefact may have one of the following states, 'being processed', 'submitted', and 'completed', with corresponding transitions.

Figure 1 shows a graphical representation of a process model. Activities are rectangular nodes with a cogwheel in the top-right corner ('activity$_1$' and 'activity$_2$'in Figure 1), artefacts are rectangular nodes with a dog's ear in the top-right corner ('artefact$_0$' to 'artefact$_2$' in Figure 1), and roles are game figure nodes ('role$_1$' and 'role$_2$' in Figure 1). Relations between the activities, artefacts, and roles become directed edges in the graphical representation. A blue arrow from an activity to an artefact indicates a dependency, a red arrow represents the create-or-modify relation. Arrows from roles to artefacts indicate responsibility (black arrow), and arrows from roles to activities (grey arrow) indicate participation in the activity. The product state model is not shown in the graphical representation.

For example, the process in Figure 1 shows an activity 'activity$_1$' that depends on 'artefact$_0$' and creates or modifies 'artefact$_1$'. The role 'role$_1$' is responsible for 'artefact$_1$' and only persons filling 'role$_1$' participate in 'activity$_1$'. Also 'role$_1$' and 'role$_2$' participate in 'activity$_2$', which creates 'artefact$_2$'.

Note that a process model in this sense focuses on dependencies and responsibilities, i.e., Figure 1 states that 'activity$_2$' cannot begin before some 'artefact$_1$' (in some state) exists. The process model *does not state* that 'activity$_1$' necessarily needs to be completed and terminated before 'activity$_2$' begins. Both activities can be conducted at overlapping times, e.g., to propagate changes on 'artefact$_0$' to 'artefact$_1$'. In this case, the artefact instances would have different revision numbers e.g. revision of the artefact used by 'activity$_2$' is lower than the revision created or edited by 'activity$_1$'.

Further note that one role node in the graphical representation that participates in multiple activities (such as 'role$_1$' in Figure 1) is a shorthand notation for two role nodes of the same kind. The textual description of the modelled process can impose further constraints that are not visible in the graphical representation. For example, that the person assuming 'role$_1$' in Figure 1 should be the same person all the time. Also, one person can fill multiple roles (if qualified), a textural description can be added if different persons should be needed.

B. Formal Requirements Specification Languages

There is a broad spectrum of formal languages that are used to describe different kinds of requirements. The Dietsch-Langenfeld-process is designed to be used with a wide variety of formal requirements specification languages. So far, we have applied the process on functional requirements for reactive systems from the automotive and railway domains In these contexts, a particular instance of so-called *pattern languages* (or pattern catalogues) was known to be appropriate for formalisation. In the following, we give a brief overview over the particular pattern language [17] that we used. The process is supposed to support other pattern languages, such as the Universal Pattern [20] or ASSERT [15], or Seamless Requirements [16] in the context of object-oriented programming. For examples of particular requirements, we refer the reader to [13], [17] for lack of space.

A stated design goal of the pattern language [17] was, to offer a good compromise between a concise logical grounding of the requirements, as well as being accessible to an engineer with little prior training in formal methods. Therefore, each pattern is a sensible English sentence with blanks, but is also tied to a fixed formula in formal logic, with corresponding blanks. There is a set of predefined, parameterised *patterns* to describe frequently occurring relations between, e.g., inputs and outputs of reactive systems, possibly including timing aspects.

The following two examples are requirements from the pattern language, instantiated over the Boolean observables `A`, `B` and `C`.

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.

*Globally, it is **never** the case that '**A**' holds.*,

By this example it is required that the value of A is not *true* at all times on all computations of the system. A more complex example including time constraints is:

*Globally, it is always the case that if '**B**' holds then '**C**' holds after **at most** '**5**' time units.*

The requirement is satisfied by a system if and only if, on each computation, each phase where B evaluates to *true* for a positive duration is followed by a phase where C evaluates to *true* after 5 time units the latest.

Note, that patterns are not limited to be instantiated with singular Boolean observables, and may contain complex Boolean expressions over observables.

Also note, that although the patterns can be understood by the untrained user, training is required to use the patterns correctly. The intuitive understanding captures many, but not all cases. Some (edge) cases rely on the understanding of the logical translation, to be explained.

III. Process

Figure 2 gives a graphical representation of the Dietsch-Langenfeld-process to manage the formalisation of functional requirements on reactive systems with pattern language. In the following, we elaborate on its roles, artefacts, and activities.

A. Roles

The model of the Dietsch-Langenfeld-process (cf. Figure 2) includes the following three roles.

Client The client role models the representative of the customer in the Dietsch-Langenfeld-process. Recall that we assume that customers are requirements engineers that need formalisation and formal analysis for existing informal requirements. Persons assuming the client role are expected to have strong domain knowledge and a deep understanding of the system that is to be built, e.g., embedded systems engineers by training. The process benefits from advanced knowledge on the concepts and principles of requirements engineering (in the sense of, e.g., [19]), and preferably a basic understanding of formal methods. The client participates in preparation and preprocessing of raw requirements and in delivery of the report (see below), the latter two together with the supervisor.

Supervisor The supervisor role is comparable to a *lead developer* in software development. Persons assuming the supervisor role should have a strong background in the formal requirements specification language used in the process. The supervisor should have profound knowledge of requirements engineering, yet need not have more than little prior knowledge of the system to be built or its domain. The supervisor needs to be able to guide the formalisation process on the conceptual level (for example, to find abstractions for specification details in the raw requirements to ease (or only enable) the formalisation), and to interpret analysis results. The supervisor is the contact point between customer's and developer's side, as well as instructing the workers.

Worker The worker role models requirements junior engineers that conduct the formalisation and analysis of requirements under guidance of the supervisor. Persons assuming the worker role have a solid understanding of the used formal requirements specification language, yet need not be familiar with the domain of the system to be built. A worker is assumed to be able to formalise common requirements on his or her own and to identify cases that need to be escalated. In small requirements formalisation projects following our process, supervisor and worker may be the same person. Employing different persons scales the process up and has the advantage that with dedicated workers, who are not domain experts, there is a higher probability of recognising irregularities that domain experts tend to overlook because of their domain knowledge [2], [10].

B. Artefacts

The central conceptual artefacts of the Dietsch-Langenfeld-process are requirements and outcomes from formal and informal analyses of these requirements for different properties. Figure 2 distinguishes five high-level artefacts that each consist of a set of lower-level artefacts (cf. Figure 3). Each artefact instance has a revision ('rev' in Figure 3), and each formal requirement in addition has a product state of the model shown in Figure 4 ('st' in Figure 3).

There are the following high-level artefacts:

Raw Requirements The raw requirements artefact consists of a set of informal requirements (cf. Figure 3) and is provided by the customer. Each requirement in 'raw requirements' needs to have an identifier ($IRID_0$). Raw requirements are informal or semi-formal. They are described using (structured) text, plus possibly additional information like a categorisation, comments, illustrations, or preliminary transition tables or state machines. A raw requirement need not be formalisable.

Informal Requirements The informal requirements artefact consists of a set of informal requirements with an identifier $IRID$ (cf. Figure 3). This artefact is basically a selection and clarification of requirements from raw requirements that is created by the supervisor, consulting the client as needed. The supervisor may, e.g., filter out irrelevant comments or requirements that are obviously not formalisable with the used formalism, join or split requirements, or propose suitable (formal) data-types for certain observables.

The client needs to accept informal requirements, because this artefact becomes the subject of the formalisation project; the final report includes results on informal requirements (not on raw requirements, yet the preprocessing establishes a relation between $IRID$ and $IRID_0$ identifiers for traceability).

Formal Requirements The formal requirements artefact consists of a set of formalised requirements, each with an

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.

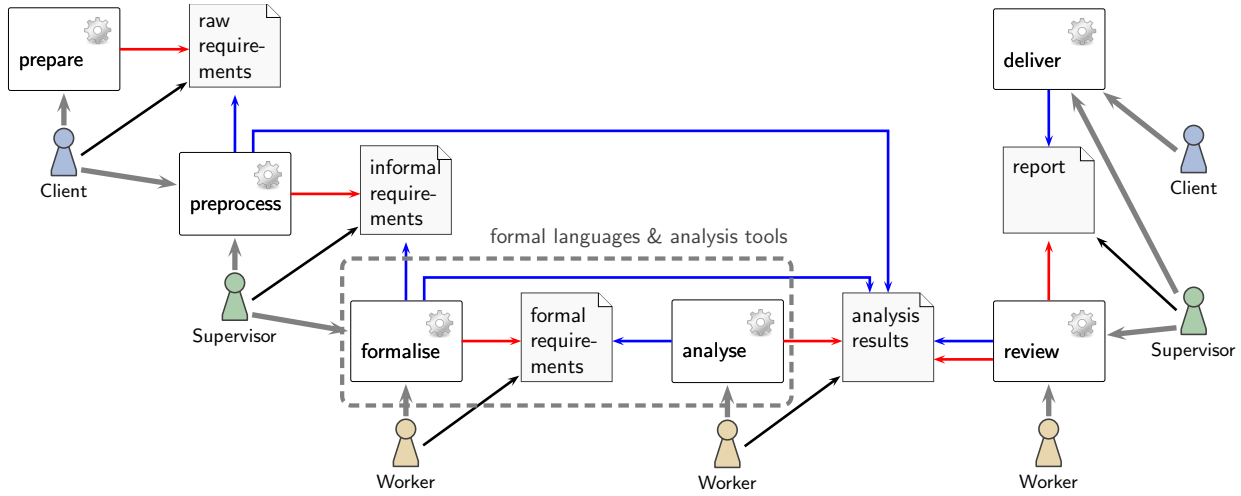

Fig. 2. Graphical representation of the Dietsch-Langenfeld-process.

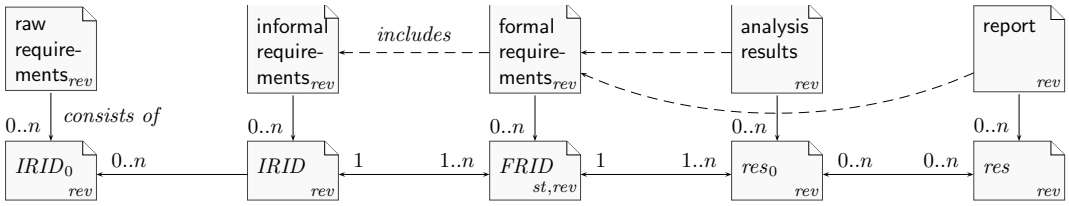

Fig. 3. Sub-structure of high-level artefacts from Figure 2.

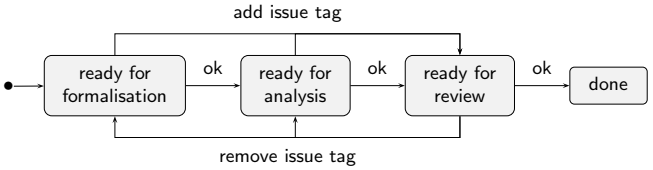

Fig. 4. Product state model of formalised requirements.

identifier $FRID$. Each formal requirement formalises one informal requirement $IRID$ (yet one $IRID$ may need multiple $FRID$), thereby the formal requirements artefact *includes* the informal requirements artefact (cf. Figure 3). Each formal requirement assumes one of four product states st from the product state model in Figure 4. As soon as an $IRID$ is considered for formalisation, a $FRID$ identity is created in state 'ready for formalisation'. If there is an issue with an $IRID$, e.g., because apparently a matching pattern is missing, or the affected observables are unclear, a corresponding so-called tag is attached to the $FRID$ and the state changes to 'ready for review'. Otherwise, the $FRID$ is associated with a pattern instance and the state changes to 'ready for analysis'. $FRID$s in state 'ready for analysis' become subject to a formal analysis with a corresponding tool and a result res_0 becomes associated. If there is an issue during analysis, e.g., due to type-checking errors or limitations of the analysis tool, again a corresponding tag is added and the state changes to 'ready for review'. $FRID$s in state 'ready for review' are reviewed and, depending on the tags and whether issues exist and can be resolved, change state back to one of the previous states or are considered done.

Analysis results The analysis results artefact collects all findings on any formal requirement, i.e., its state and tags, any relevant tool output from analysis, – including so-called counter-examples, i.e., exemplary input sequences that witness an issue – and possibly additional comments, questions, or general feedback on the requirements quality. All findings for one $FRID$ constitute one preliminary result res_0, preliminary results refer to their $FRID$ for traceability, and in this sense analysis results include formal requirements (cf. Figure 3).

Report The report artefact consists of one (final) result per $IRID$. The report is prepared with the client as target audience, hence it summarises the analysis results and may, e.g., contain additional explanations for more complicated issues. The form of these artefacts may be prescribed by the customer, e.g., using a particular reporting sheet.
Note that the report includes formal requirements (cf. Figure 3), i.e., the customer receives the formalised requirements for further reference, yet it does not directly include the analysis results, which are of more technical nature.

C. Activities

We distinguish the following six activities.

Prepare The client is responsible for providing raw requirements.

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.

Preprocess The supervisor is responsible for preparing informal requirements from raw requirements. The supervisor inspects the raw requirements for, e.g., obvious errors, irrelevant information, or for constructs that are known to be unsupported by the used formalism, and other road blocks for the formalisation. Issues are resolved with support from the client.

Note that the activity 'Preprocess' also depends on analysis results: Internal results may indicate issues that require changes to informal requirements (which need to be conducted in cooperation with the client).

Formalise & Analyse Workers are responsible for creating formal requirements from informal ones (Formalise) and for analysing formal requirements (Analyse), in particular by the operation of formal analysis tools.

Note that formal requirements that are not in the product state 'ready for analysis' are not analysed. Issues during this activity, if not solvable by the worker, are recorded by product state and documented in analysis results.

The exact instructions for the analyse and formalise activities have to be given by the analysis tool (e.g. BTC [4], ASSERT [14] or HANFOR).

Review The supervisor is responsible to review analysis results as provided by workers. Note that, because analysis results effectively include formal requirements, the formalisation as such is subject of the review. The worker participating in the review is primarily meant to represent the workers who conducted the formalisation and analysis. Depending on the experience of particular workers, the process can be extended to an iteration where workers review other workers' results to lower the workload of the supervisor. Further note that this activity modifies both, analysis results and report, thereby introducing a feedback loop to activities 'Preprocess' and 'Formalise' (and transitively to 'Analyse'). The goal of this activity is to understand issues that occurred during formalisation or analysis and either to resolve these issues (e.g. in the case of false positives) or to summarise and explain them in the report.

Deliver The supervisor delivers the report to client. This can be a meeting or a video conference where the supervisor explains positive and negative results from the analysis.

Note that report may include formal requirements that were, e.g., not formalisable or analysable even after (re-)consulting the client in activity 'Preprocess'. If these issues should be solved, we propose to start over with the process for a new project, where the report becomes part of the raw requirements.

D. Scheduling & Budgeting

The process model can be operated in a strict waterfall-like fashion. The process starts with agreeing on a fixed set of informal requirements, formalise and analyse activities are conducted on the informal requirements, and after the report is produced by the review, the client is contacted, and the results are discussed. If issues arise in the review, a new project is started. While this approach is the most predictable, it may also take the longest overall, as no feedback loop is used.

The process can also be operated in a more agile fashion, by allowing the supervisor to contact the client whenever deemed necessary or according to a schedule, to discuss issues and findings. It is even possible to allow the client to continuously add requirements to *IRID* as development continues.

An approach to budget a requirements formalisation project following the Dietsch-Langenfeld-process from informal requirements to report is to fix the available time (cf. [1]). Then the contract is to analyse as much as possible with the given budget, possibly with priorities. This approach allows perfect prediction of time and cost for both parties, but the extension of the results may vary. An estimation of the extension is also relevant for customers, yet obviously hard for the first project with a new client. But in our experience, good estimations are possible already for the second project with the same client in the same domain. The process could also be budgeted on a per-requirement basis, where a fixed price per requirement is agreed upon. This would require the supervisor to set a price according to his experience, and may be most suitable for in-house projects.

Note, that the quality of the raw requirements and the 'distance' between raw requirements, informal requirements, and formal requirements may have a significant impact on project duration and cost. It is well-known that, in general, preparing raw requirements alone can be a substantial project [19], hence we exclude it from the discussion here. Our current experience is based on raw requirements of good quality, i.e., adequately structured and atomic, comparable to the requirements inspected in [13]. Project efforts for raw requirements of different qualities, e.g., if starting from a Wikipedia page like [7], may vary.

IV. DISCUSSION

The process model presented in Section III clearly integrates formal requirements and formal analysis into a larger requirements engineering process to improve requirements quality. In this section, we use the process model to analyse what kind of results one can expect from the process and to which risks the process and its results are subject to.

The final outcome of the process is the report. Recall that report is prepared wrt. informal requirements (also cf. Figure 3). In the report, we can distinguish the following kinds of results. Firstly, the report can be seen as an expert review, i.e., the client receives feedback on standard quality aspects of provided requirements (even on non-formalisable requirements). The creation of this result is inherent in the Dietsch-Langenfeld-process because formalisation is a manual activity of supervisor and workers (in our experience, the precision of today's automatic, natural language processing-based formalisation is too low for requirements on critical systems in the automotive domain). Secondly, there are statements on the violation of formally defined quality aspects of requirements such as well-typedness or (formal) consistency (the set of available outcomes depends on the tools used in activity 'analyse', cf. Figure 2).

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.

Finally, there is the formalisation as such that can be used in simulation, in subcontracting development, or in test case generation.

For each of the former two results, the outcome can be positive (low quality wrt. standard aspects; violation of formal quality aspect) or negative (no low quality indications or violation detected), like in testing. As we have human work involved, there is a risk for errors, each outcome can be true or false, i.e., each result can suffer from type I (false positive) or type II (false negative) errors. For the first kind of results, we face the same situation as any expert review: Outcome quality depends on the experience of the supervisor and worker. There is a risk for false negatives, yet we expect it to be lower than in general because here the review is systematic and tool-supported. For example, requirements are not simply forgotten or overlooked because they will be found missing in the set of $FRID$s.

The second kind of result needs a closer look: The process produces a statement (in form of the report) first of all on $FRID$ (in the following, we identify requirements and their identities), yet the primary customer interest is $IRID$. Whether statements on $FRID$ are true or false positives or negatives basically hinges on the correctness and capabilities of the employed tool and can be tackled by tool developers. Statements on $IRID$ in contrast depend on the validity of its $FRID$, i.e., whether this set of formal requirements denotes exactly the same set of system behaviours that the authors of the $IRID$ text had in mind. Here lies a risk for false positives or negatives which is inherent in and well-known from the work with informal requirements specifications.

In the following, we discuss all four cases, first the positives and then the negatives, and analyse the possible impacts and responses in our process. In the positive case, the tool reports indications of a defect on $FRID$ (which is true, if we assume the tool to be sound), yet the positive could be true or false wrt. $IRID$. We propose that the supervisor supports the client in understanding whether a positive result is true or false wrt. $IRID$. To this end, the supervisor interprets the tool output (the res_0) wrt. the $IRID$; in rare cases, if the client is not able to resolve the issue on the level of $IRID$, the outcome may be inconclusive. True positives are usually appreciated by the customer, false positives not necessarily. False positives can be perceived as unnecessary expense, and customers confronted with too many false positives may lose motivation to engage in the analysis of positives as well as losing confidence in the method. Unfortunately, in formal analysis, one slight mistake can result in many true positive results res_0 for many formal requirements $FRID$. For this reason, we distinguish between results and the report. The supervisor is expected to interpret the results and provide support in understanding the root cause in the report.

In the negative case, the tool reports no indications of a defect on $FRID$. If the $IRID$ also do not have that defect, we have a true negative, which is a desirable outcome. Otherwise, the negative is false, i.e., the $IRID$ have a defect yet the $FRID$ do not because (if we assume the analysis tool to be sound) the $FRID$ happen not to be valid formalisations of the $IRID$. Perfect validation can be very expensive [19]. Extensively validating all $FRID$ in our process is prohibitively expensive. To obtain a report in the spirit of a *dependability case* [11], we discuss the risks for uncovered invalidities. We do not estimate this risk to be unaccetably high, since an invalid $FRID$ needs to get past the following indication: (i) the preprocessing of supervisor (manual), (ii) the formalisation was possible (tool support) and (iii) the formalised (valid or invalid) requirement has been confirmed to be well-typed (tool support), and (iv) the formalised requirement has passed the more involved analyses (tool support), if only a few requirements are invalid, their invalidity may show up since combinations of requirements are checked and the invalid requirement may be identified as vacuous.

So a negative means that all validly formalised requirements do not have defects. Thereby, formal analysis (in our process) lowers the number of possible reasons for later issues with the system, and hence increases the probability of finally obtaining a dependable system. In other words, the 'clou' of the Dietsch-Langenfeld-process is that the remaining probability for false negatives is *simply accepted*. The process as is seems to offer a good balance between effort and results (cf. Section V).

V. Experience

We have conducted a number of requirements formalisation projects following the Dietsch-Langenfeld-process with clients from companies in, e.g., the automotive and the railway domain, using real-world requirements. The supervisor role was assumed by university researchers, the majority of workers were students with a bachelors degree in computer science. We received sets of raw requirements ($IRID_0$) that contained between 20 and 1000 requirements and did not need extensive preprocessing. We conducted the process in a mild agile fashion, that is, we made use of the feedback loops and approached the client with questions during the project. Still, there was a clearly defined project end. Note that, for now, we can only report on preliminary results and impressions, a more detailed and quantitative analysis of the projects, e.g., wrt. to severity of uncovered issues, is future work.

The process activities were supported by the two tools HANFOR and REQANALYZER[1]. HANFOR is a web-based tool that offers basic requirements management (by, for example, keeping track of identifiers, tags, and product states), that includes a requirements editor (with, e.g., pattern selection and syntax completion for known observables to avoid typing errors and aliasing), and that verifies that all expressions are well-typed wrt. to observable declarations and the type of fields in a pattern. Reporting is supported in HANFOR by functions to search and group requirements by tags or status. REQANALYZER [12] is a formal requirements analysis tool that analyses sets of requirements for the defects *inconsistency*, *vacuity*, and *rt-inconsistency*.

[1]both tools are available at https://ultimate-pa.github.io/hanfor

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.

In the following, we report some observations from the different activities. In the prepare activity, comma separated values (CSV) files turned out to be the least problematic means to exchange requirements. In the formalise activity, most of the requirements have been found easy to formalise because they are very simple in structure (invariants, or a kind of bounded response), clearly written, or adhere to a certain grammatical template. The small number of hard to formalise requirements were firstly postponed to gather further information on the context, while formalising other simple requirements. Often it also helped to read comments and headlines around the requirement (which is typically not necessary for the easy requirements), otherwise the requirement was discussed with the client.

Overall, following the Dietsch-Langenfeld-process was well received by our industrial partners. Among several advantages over an in-house solution, our partners mentioned the aspects that the own personnel did not require particular training, that the project did not bind own personnel with an uncertain outcome, and that the formalisation procedure qualifies as an external review in certain industry contexts, that is, even if the formal analysis only yields (possibly false) negatives, the requirements have still been externally reviewed in a systematical way.

In all conducted projects, the partners considered the results to be relevant because critical defects in the requirements were found and because the client appreciated the overall input from the external review. Issues found and tagged during formalisation can appear in the final report and may help the client to improve the requirements quality. The overall number of false positives was rather low, including the false positives handled by supervisor and worker. It is our impression that the raw requirements provided by the customer were already of good quality, hence formalisation and analyses primarily increased confidence in quality and opens up the path to, e.g., generating test cases automatically from the formalised requirements.

We anticipate that the success of our projects depended strongly on the implementation of the activities preprocess and deliver, where a knowledgeable supervisor provides interpretations of the analysis outcomes. Positive outcomes (true or false) that need to be escalated to the client level are in general not trivial to understand, and it remains a challenge to report them comprehensibly. In particular findings of advanced requirements properties such as rt-inconsistency tend to be hard to understand yet may cause serious issues in the system and development if they go undetected (as reported by clients).

VI. Conclusion and Future Work

We have presented a process that bridges the gap between the informal requirements usually found in industry, and formal requirements analysis.

The process facilitates a third party that contributes its expertise in formal methods and requirements engineering. In the process, the formalisation activity serves as an additional review of the requirements. Results of the review and of the formal analysis are presented to the client by an expert that is able to give comprehensive explanations on the process requirements and the formal background, if necessary. The process in particular allows to use (or assess) formal requirements analysis without the effort for training personnel in formal methods.

We have reported qualitative results from multiple real world projects that uncovered critical issues in the considered requirements. Ongoing and future work comprises a thorough quantitative analysis of data gathered in the conducted projects.

References

[1] S. F. Arenis, B. Westphal, D. Dietsch, M. Muñiz, A. S. Andisha, and A. Podelski. Ready for testing: ensuring conformance to industrial standards through formal verification. *Formal Asp. Comput.*, 28(3):499–527, 2016.

[2] D. M. Berry. The importance of ignorance in requirements engineering. *J. Syst. Softw.*, 28(2):179–184, 1995.

[3] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Gros, A. Kamsky, S. McPeak, et al. A few billion lines of code later: using static analysis to find bugs in the real world. *CACM*, 53(2):66–75, 2010.

[4] T. Bienmüller, T. Teige, A. Eggers, and M. Stasch. Modeling requirements for quantitative consistency analysis and automatic test case generation. In *FM&MDD*, 2016.

[5] M. Brill, R. Buschermöhle, W. Damm, J. Klose, B. Westphal, and H. Wittke. Formal verification of LSCs in the development process. In H. Ehrig et al., editors, *INT*, number 3147 in LNCS, pages 494–516. Springer, 2004.

[6] X. Chen, Z. Zhong, Z. Jin, M. Zhang, T. Li, X. Chen, and T. Zhou. Automating consistency verification of safety requirements for railway interlocking systems. In D. E. Damian, A. Perini, and S. Lee, editors, *RE*, pages 308–318. IEEE, 2019.

[7] A. Cimatti, M. Roveri, A. Susi, et al. From informal requirements to property-driven formal validation. In D. D. Cofer and A. Fantechi, editors, *FMICS*, volume 5596 of *LNCS*, pages 166–181. Springer, 2008.

[8] A. W. Crapo, A. Moitra, C. McMillan, and D. Russell. Requirements capture and analysis in ASSERT. In *RE*, pages 283–291. IEEE, 2017.

[9] M. Filax, T. Gonschorek, and F. Ortmeier. Correct formalization of requirement specifications: A V-Model for building formal models. In T. Lecomte, R. Pinger, and A. B. Romanovsky, editors, *RSSRail*, volume 9707 of *LNCS*, pages 106–122. Springer, 2016.

[10] I. Hadar, P. Soffer, and K. Kenzi. The role of domain knowledge in requirements elicitation via interviews: an exploratory study. *Requir. Eng.*, 19(2):143–159, 2014.

[11] D. Jackson. A direct path to dependable software. *CACM*, 52(4), 2009.

[12] V. Langenfeld, D. Dietsch, B. Westphal, J. Hoenicke, and A. Post. Scalable analysis of real-time requirements. In D. E. Damian, A. Perini, and S. Lee, editors, *RE*, pages 234–244. IEEE, 2019.

[13] V. Langenfeld, A. Post, and A. Podelski. Requirements defects over a project lifetime: An empirical analysis of defect data from a 5-year automotive project at bosch. In M. Daneva and O. Pastor, editors, *REFSQ*, volume 9619 of *LNCS*, pages 145–160. Springer, 2016.

[14] A. Moitra, K. Siu, A. W. Crapo, H. R. Chamarthi, M. Durling, M. Li, H. Yu, P. Manolios, and M. Meiners. Towards development of complete and conflict-free requirements. In *RE*, pages 286–296. IEEE, 2018.

[15] A. Moitra, K. Siu, A. W. Crapo, M. Durling, M. Li, P. Manolios, M. Meiners, and C. McMillan. Automating requirements analysis and test case generation. *Requir. Eng.*, 24(3):341–364, 2019.

[16] A. Naumchev and B. Meyer. Seamless requirements. *Comput. Lang. Syst. Struct.*, 49:119–132, 2017.

[17] A. Post, I. Menzel, and A. Podelski. Applying restricted english grammar on automotive requirements — does it work? In *REFSQ*, pages 166—180, 2011.

[18] H. Roehm, T. Heinz, and E. C. Mayer. Stlinspector: STL validation with guarantees. In R. Majumdar and V. Kuncak, editors, *CAV*, volume 10426 of *LNCS*, pages 225–232. Springer, 2017.

[19] C. Rupp and die SOPHISTen. *Requirements-Engineering und -Management*. Hanser, 6th edition, 2014.

[20] T. Teige, T. Bienmüller, and H. J. Holberg. Universal pattern: Formalization, testing, coverage, verification, and test case generation for safety-critical requirements. In *MBMV*, pages 6–9, 2016.

[21] *V-Modell XT*, 2019. Version 2.3.

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on December 15,2021 at 11:53:56 UTC from IEEE Xplore. Restrictions apply.