Towards Successful Subcontracting for Software in Small to Medium-Sized Enterprises

Bernd Westphal∗, Daniel Dietsch ∗, Sergio Feo-Arenis∗, Andreas Podelski∗, Louis Pahlow†, Jochen Morsbach‡, Barbara Sommer‡, Anke Fuchs‡, Christine Meierhöfer‡

∗Department of Computer Science, Albert-Ludwigs-Universität Freiburg, Germany
†Department of Law, Universität des Saarlandes, Saarbrücken, Germany
‡Department of Law, Universität Mannheim, Germany

Abstract—Many small to medium sized enterprises (SMEs) that specialise in electrical or communications engineering are challenged by the increasing importance of software in their products. Although they have a strong interest in subcontracting competent partners for software development tasks, they tend to refrain from doing so. In this paper we identify three main reasons for this situation, propose an approach to overcome some of them and state remaining challenges. Those reasons are situated in the intersection of software engineering and jurisprudence and therefore need to be addressed in an integrated and multidisciplinary fashion.

I. INTRODUCTION

In the embedded systems domain, many small to medium sized enterprises (SME) are highly specialised in electrical or communications engineering but lack software engineering expertise. They have a vital interest in subcontracting software development tasks to competent partners [1] in order to maximise the efficiency of the overall development process and the quality of the final product. Yet today, many of those SMEs refrain from subcontracting software development [2]. They assess the associated technical and legal risks as too high and prefer to develop their software in-house, even if they possess only limited expertise. This situation may lead to drastically increased development times, higher costs, decreased quality of the final product, or even project failure compared to subcontracting for software development. Therefore, SMEs may miss many opportunities for synergetic effects such as strategic development alliances or commonly agreed tools or technical procedures to check whether the software is running acceptance tests to check whether the software is delivered software satisfies the requirements or not. Accepting the interpretation of the contractor can incur high costs at the customer’s side for the corresponding adjustments and lead to delayed delivery of the final product. Bringing the dispute to court, even as a last resort, is highly unattractive for SMEs for the following reasons: (a) a court ruling takes time and thus further delays the project, (b) a court ruling incurs costs, (c) it is uncertain whether the necessary compensation can be achieved, (d) a court only decides over the rights and duties of each party, but it does not suggest how to use the decision to achieve project success, and (e) mutual trust between the former partners is hampered and thereby the already achieved project progress may be lost. In addition, there is high uncertainty about the outcome. As the requirements documents are uncertain, the court may appoint an expert witness who may confirm the interpretation of the contractor and the interpretation of the customer as well.

Secondly, there is the risk of misunderstandings (or under-regulations) regarding the acceptance test procedures, i.e. procedures that are to be applied in order to decide whether the delivered software satisfies the requirements or not. If a dispute about whether the delivered software satisfies the requirements is brought to court, there is again a high uncertainty about the outcome as there are no standardised or commonly agreed tools or technical procedures to check whether the delivered software satisfies the requirements. For instance, a customer may try to maliciously delay payments to the contractor by applying a huge set of long-running acceptance tests to check whether the software is “reliable” where the contractor expected that a small set of representative tests would be sufficient. Again, an expert witness may follow the interpretation of the contractor as well as the interpretation of the customer.

And thirdly, there is a well-known risk of misunderstandings of regulations in the contract. SMEs often cannot afford intensive and specialised legal advice, so if a dispute is brought to court, there is again a high uncertainty about the outcome due to insufficient contractual regulations. Conflicts arise in particular if contracting parties state clauses that are subject to interpretations by a court or in conflict with statutory regulations. Even assuming that the requirements
are clear to both partners, and that both partners have agreed on acceptance test procedures, SMEs are uncertain as to whether the legal interpretation of the contract corresponds to their intended interpretation and whether their complete agreement is legally sound. For instance, there may be disputes over obligations of the customer. In many projects, the contractor’s work depends on technical documentation to be provided by the customer during the project. If the customer holds back such documents, the development plan of the contractor may be disturbed and negatively affect other projects at the contractor’s side.

Thus, in order to support subcontracting between SMEs, we need to establish (legal) certainty. The outcome of a possible court judgement needs to be as clear as possible, which also serves the goal of successful alternative dispute resolution. To achieve legal certainty, we need (a) approaches to obtain clear and precise requirements, because they avoid the first source of uncertainty. We need (b) clear and precise acceptance testing procedures, because they avoid the second source of uncertainty. And we need (c) standardised legal contracts which integrate both and avoid the third source of uncertainty. The contract allows a judge to decide on (a) and (b), and thus increases legal certainty.

In this paper, we report on the current situation of SMEs who are involved in subcontracting for software development and we outline a new approach to solve the legal and technical problems faced by them in an integrated fashion by both disciplines, computer science and jurisprudence. We propose to govern subcontracts for software development by a modular legal contract that includes — in machine-readable form — an unambiguous representation of requirements (checkable requirements) as well as a mutual agreement on preferably tool-based acceptance testing procedures for those requirements (checking tools). The whole process is thereby partly automated, its objectiveness is improved, and it is overall accelerated.

In Section II, we report on the current situation of SMEs who are involved in subcontracting for software development. In Section III, we introduce the notions of checkable requirements and checking tools as the basis of our modular software development contract. Section IV discusses related work and Section V concludes and identifies the challenges raised by our approach.

II. CURRENT SITUATION

Despite a number of surveys on the situation of SMEs in the literature [3], [4], [5], [6], [1], we know little about the relations between contract partners.

In order to better understand the situation of SMEs, we conducted a large survey [2]. The anonymous survey was announced to about 3900 persons at about 2800 companies in the German federal state of Baden-Württemberg which are contained in the European financial database AMADEUS. Based on ca. 100 responses to our questionnaire, we formulated the hypothesis stated in the introduction that contracting is today hindered by three sources of uncertainty. Nearly half of the responses provided a contact address and agreed to be approached for an interview. In order to investigate our hypothesis further, we selected 8 companies based on the most significant answers and approached them by mail or phone. One of the companies required a very strict non-disclosure agreement, which could not be agreed with the involved universities. The application domains of the remaining 7 companies comprises wireless sensor networks, sensor/actuator systems for the automotive domain, open-source software customisation, building and factory automation, and web applications. With one exception, the considered companies develop software for embedded systems.

The interviews were conducted in an interdisciplinary setting. Interviews were selected and planned by at least one researcher from jurisprudence and one from computer science. For the execution of the interview, diverse teams were used according to their availability. We used the semi-structured interview method, having as central axis the company’s past and present experiences regarding contracting for software. Main topics of the interviews were: the role of the company in the contracting process, the extent to which written contractual clauses are used, the types of contractual partners, the requirements engineering process, quality assurance, acceptance test procedures, and past disputes.

In the following, we provide summaries of the individual interviews. Additional information from the survey is summarised in Table I.

A. Individual Interviews

1) Company A: Company A is a small company specialised in wireless communication modules that are sold individually to other manufacturers and integrated in an own product line of wireless alarm systems. The company has in total about 20 employees. The development team consists of three engineers who perform both software and hardware development activities. Most of Company A’s products are used in safety critical applications which result in elevated liability risks. The products are mostly embedded systems for which updates and patches can only be delivered by product recalls which incur elevated costs. Additionally, the alarm systems need to be backed by safety standard certifications in the countries where they are marketed.

The CEO performs as main requirements analyst, deciding upon product features at each innovation cycle iteration. In each iteration, state-of-the-art techniques are incorporated and the products are updated to comply with actual regulations and to accommodate for changing market conditions. A requirements specification is derived by the developers after selecting a suitable hardware platform. For developments as contractor, requirements documents are developed...
cooperatively with the customers and iteratively refined as the project progresses.

The hardware and software development are mostly sequential (waterfall model). The implementation phase causes usually many changes in the requirements. Due to time constraints, the requirements changes are not tracked efficiently which makes final testing more difficult to define. During development, features are individually checked, both by using simulations and field tests. Errors can often cause delays in the planned delivery dates. Final products from the own product line are delivered after successful standard certification. For individual solutions, joint acceptance tests are performed. In larger or more critical projects, Company A tries to base acceptance on third party certification.

Company A finds acceptance of tailor-made solutions complicated and names insufficient ability to track change requests and differences in technical knowledge of the contract partners as the main reason. They also state that current acceptance test methods are not able to reduce liability risks sufficiently. The wish for a more structured requirements process and an improved quality assurance underlines the perceived importance of increasing certainty throughout the execution of contractual agreements at Company A.

2) Company B: Company B is an engineering bureau with 8 developers which offers development of custom, microcontroller-based embedded systems hardware according to customer specifications. The purpose of the developed hardware ranges from closed-loop control to elaborate graphical user interface components. Customers range from SMEs to major enterprises, e.g., from the household appliances domain. Software development is offered on demand. The freedom of design decisions depends on the project.

The methods employed for requirements engineering are adapted to the demands of the customer. Changes and clarifications of requirements are tracked by exchanging word processor documents, modifications are partly the duty of the customer and partly of developers at Company B. Quality assurance mainly relies on software testing, including the use of simulators, hardware-in-the-loop, and integrated system tests. Acceptance testing of software is conducted for the customer on the final system consisting of hardware and software. A detailed test protocol is provided as the basis to declare acceptance.

The CEO of Company B and the lead software developer agree on the prevalence of issues regarding the clarity of requirements. Often, the requirements are not precise enough to decide during acceptance tests whether they are fulfilled in their entirety. The company has experienced disputes in this regard. Those disputes could have potentially been avoided by a stricter regulation of acceptance testing procedures and a more structured requirements engineering process.

3) Company C: Company C is a medium-sized developing company specialised in building and factory automation systems. They offer a product line of highly customisable monitoring and controlling systems and engineering services for related projects. A team of 8 developers takes care of the

### Table I

**SUMMARY OF THE PARTICIPATING COMPANIES**

<table>
<thead>
<tr>
<th>Interviewing Team</th>
<th>Company A</th>
<th>Company B</th>
<th>Company C</th>
<th>Company D</th>
<th>Company E</th>
<th>Company F</th>
<th>Company G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 Comp.Sc.</td>
<td>3 Comp.Sc.</td>
<td>2 Jur., 1 Comp.Sc.</td>
<td>3 Jur., 1 Comp.Sc.</td>
<td>2 Jur., 1 Comp.Sc.</td>
<td>2 Jur., 1 Comp.Sc.</td>
<td>2 Jur.</td>
</tr>
<tr>
<td>Interview Partner(s)</td>
<td>CEO, Developer</td>
<td>CEO</td>
<td>CEO &amp; Founder</td>
<td>Sales Manager, Development Manager</td>
<td>CEO</td>
<td>CEO &amp; Lead Developer</td>
<td>CEO</td>
</tr>
<tr>
<td>Branch</td>
<td>HF Wireless components</td>
<td>PCB development and manufacturing</td>
<td>Building and Factory Automation</td>
<td>Automotive OEM</td>
<td>CRM/ERP Web Application Development</td>
<td>Open-source customisation</td>
<td>Hydraulic components</td>
</tr>
<tr>
<td>Role(s)</td>
<td>Customer, Contractor</td>
<td>Contractor</td>
<td>Customer, Contractor</td>
<td>Customer</td>
<td>Contractor</td>
<td>Contractor</td>
<td>Customer</td>
</tr>
<tr>
<td>Company Size (Persons)</td>
<td>20-249</td>
<td>5-19</td>
<td>20-249</td>
<td>20-249</td>
<td>1-5</td>
<td>5-19</td>
<td>20-249</td>
</tr>
<tr>
<td>Development Team Size</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>.</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Average Project Size (KLOC)</td>
<td>10-50</td>
<td>≥ 100</td>
<td>50-100</td>
<td>10-50</td>
<td>.</td>
<td>10-50</td>
<td>unknown</td>
</tr>
<tr>
<td>Customer Location</td>
<td>International</td>
<td>National</td>
<td>Regional</td>
<td>National</td>
<td>International</td>
<td>International</td>
<td>National</td>
</tr>
<tr>
<td>Acceptance Test Contractual Regulation</td>
<td>Sometimes function tests</td>
<td>Function tests</td>
<td>No</td>
<td>No</td>
<td>Sometimes function tests</td>
<td>No</td>
<td>Sometimes function tests and fictitious acceptance</td>
</tr>
<tr>
<td>Past Disputes</td>
<td>x/S/x/S</td>
<td>x/S/x/S</td>
<td>x/S/x/S</td>
<td>-/-/x/-</td>
<td>-/-/-/x</td>
<td>-/-/-/-</td>
<td>x/x/S/x</td>
</tr>
<tr>
<td>Description</td>
<td>Product Compliance / Change Requests / Cost Modification / Deadlines not met / Customer’s duty to cooperate / Undetected flaws due to complexity / Rights of use / Source code cession</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Settlement of Disputes</td>
<td>x/-/-/-</td>
<td>x/-/-/-</td>
<td>x/-/-/-</td>
<td>x/-/-/-</td>
<td>x/-/-/-</td>
<td>x/-/-/-</td>
<td>x/-/-/-</td>
</tr>
<tr>
<td>Description</td>
<td>settlement out of court without legal advice / settlement out of court with legal advice / arrangement after receiving order / court ruling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
development effort of the innovation cycle and of the coding required by the orders for tailor-made solutions. When the load on the developers is high, an external company with which Company C has developed a long-term relationship is subcontracted. Although Company C has currently room for growth, an expansion has been intentionally slowed down due to the perception of the CEO that it is risky to explore subcontracting possibilities with other enterprises. The management believes that, in general, it is hard to find competent development partners, and very hard to obtain appropriate compensation in case of failure of a development subcontract. The products offered by Company C usually require high quality standards since they are employed for critical applications. Additionally, many customers are subject to norms and regulations that have to be met by the automation systems installed at their facilities.

The requirements process at Company C operates in two different scenarios: for the development of the company’s product line, the requirements documents of public bidding requests together with actual regulations are analysed to determine the features included in each release cycle. For the tailor-made solutions offered, a separate contract is usually signed for the construction of the requirements specification and a feasibility study based on informal customer requirements. Upon completion of this contract, the customer decides whether to order the development based on the resulting recommendations and cost projections. For quality assurance, Company C has an own testing team that performs strict checking of the components acquired externally and applies a complete validation process on the own products which includes code quality checking, simulations, field testing, external audits, and certificate acquisitions. Interestingly, some commercially available code static analysis tools are utilised as part of the quality management activities. Acceptance tests at this enterprise consist of extensive field tests that may extend up to three months. During the tests, all features and failure scenarios are thoroughly checked. Due to the high complexity of the projects executed, the final product usually does not exactly fulfill the initial specifications. Specifications are refined and modified as the project progresses without thoroughly tracking their changes. Acceptance tests are conditioned mainly to the satisfaction of the customer and not to a systematic checking of the requirements agreed upon.

As stated by the company, fear that new contractors may complicate the own development process has prevented expansion. The reluctance to subcontract finds its roots in previous experience that lead to the perception of the risks involved as high. Company C expressed the wish to alleviate the perceived risks through legal agreements that prescribe certain aspects of the development process like creating glossaries, certification through independent testers, guidelines for electronic communication, bug management and disclosure of source code.

4) Company D: Company D finds its niche in the development and manufacturing of diverse electronic sensors and control units for the automotive industry. It is a medium-sized enterprise with a software development team of only four people plus a technical development team. In total, Company D employs 140 people. The company operates in a highly competitive branch that requires efficient development at relatively low prices, with the advantage of receiving high volume orders in many cases. The enterprise also generates value by producing intellectual property in the form of patents, specially in the sensors technology field. Most of Company D’s customers are big automotive corporations, they require from their suppliers compliance with safety and quality standards. Given the disproportionate economic power of the customers, most of the contracting terms are determined by the customer. According to the development manager, when the capacity of Company D’s development team is depleted, parts or whole projects are subcontracted. In order to acquire business, open bidding requests from vehicle manufacturers are analysed to produce offers that compete in a selection process mainly guided by prices and expected quality. In order to improve the perceived quality of their products, Company D’s development process is certified to ISO/TS standards.

Once an order is received, a structured requirements specification document is compiled based on the requirements made available by the customer and the list of standards that are to be satisfied. The requirements from which a specification is derived are usually kept at an abstract level that leaves a broad design space available for the developers to work on. According to the sales manager, this has the advantage of allowing diverse technical solutions, with the backside that the initial cost projections are often imprecise. Company D performs continuous testing during development as their main quality assurance activity. In some cases, external certification parties attest the compliance of the products with norms and customer requirements. Acceptance tests use the customer requirements as basis and are performed almost entirely by Company D. Once succeeded, series manufacturing starts and orders are filled in batches as required by the customer.

The company assumes high risks by performing development with such asymmetrical contract partners. On the one side, it is difficult to obtain feedback regarding requirements due to the lack of efficient communication channels with the technical counterparts on the customer’s side. On the other side, requirement changes usually lead to disputes over increased contract prices. Furthermore, ambiguities on the requirements have caused recall actions that prove very expensive. With them, questions and disputes over product liability may arise.

5) Company E: Company E is a small company that employs 10 people, of which most work directly on software development. Their main activity is the customisation
of an own, web-based ERP/CRM system that is offered as Software-as-a-service. The products are almost entirely developed in-house or integrate open-source components. Usually, the customers are companies with limited knowledge about software and in particular software requirements. Company E elaborates a requirements specification based on imprecise, informal high-level requirements that are refined iteratively. Changes are tracked by an internal log that links to mail messages or telephone call notes. Orders for which the requirements are not sufficient to estimate the required costs are sometimes rejected.

Early prototypes are discussed with the customers as means to refine the requirements specification. The success of the development projects depends greatly on an efficient involvement of the customer. Some effort is invested in transferring technical knowledge aimed at improving customer communication. A simple structured software development project tracking strategy is used based on word processor and spreadsheet documents. Since the products offered are web-based and hosted by the company, a reduced overhead results for delivering updates. The product is delivered in an early beta state and the customer reports any defects found. They are then classified as ordinary errors to be fixed or as changes to the specification that may require additional effort or costs. Different features of the delivered software are evaluated by the customer during live testing and acceptance is declared individually. Formal acceptance tests are not performed. Full acceptance is then implied when all features are positively evaluated by the customer.

Company E assumes certain risks due to static contractual agreements that are used without adaptation to the individual circumstances of single projects. Some disputes arise due to the heterogeneous backgrounds of developers and customers. Due to deficient requirements change tracking, it proves difficult to agree upon the nature of a change request. Customers insist on requests being simple requirement changes while the developers insist on them being new features, with their implied additional cost. Furthermore, those differences in technical knowledge lead to disagreements in the acceptance test phase. Company E additionally names implicit requirements and scope creep as source for disputes. Those problems are further exacerbated because the negotiation of the requirements and their changes is unstructured and the projects are usually not bounded by fixed deadlines.

6) **Company F:** Company F mainly develops software for embedded systems. They customise or port open-source software to proprietary platforms, develop platform-specific device drivers, and offer software maintenance to different customers. Customers range from small SMEs to major enterprises from the automotive branch. Company F employs 7 developers and does not subcontract for development. Their experience is that the focus on open-source software simplifies a number of legal issues, e.g., many of the programs that need to be ported are already released under the GNU Public Licence (GPL). Consequently, there may not be any additional specific licensing regulations or agreements regarding availability of the source code. Company F uses nevertheless an extensive contract for larger projects or for projects with new customers. They state that it is also not uncommon, depending on the age of business relations, that projects are executed without any written contract.

Company F does not use explicit requirements or design documents to capture the informal requirements supplied by the customer in natural language. The only artefacts are series of emails and series of prototypes exchanged with the customer. The CEO of Company F expressed a strong reluctance against structured development processes, or processes in general. The development process of Company F could be termed “chaotic”, and success seems to depend strongly on exceptionally experienced and skilled developers. Developers individually choose the approach for each project how they see fit. The resulting process could be categorised as a mixture between rapid prototyping and a classical waterfall model. Notably, there is typically no final documentation at all, even source code comments are rare. Quality assurance relies on static analysis, run-time analysis, and reviews from the open-source community. Acceptance tests are performed by the customer within a certain time-bound after completion. If no issues are reported from the tests, acceptance is assumed. In some cases, Company F defines the test procedures for the customer due to the customers “lack of technical understanding”.

The leaders at Company F see major problems with the traceability of requirements: Requirements are often unclear, e.g., by lacking expertise on the side of the customer and clarified between individual developers and representatives of the customer. These clarifications don’t follow a defined communication procedure such that changes in the specifications are often not sufficiently communicated to the rest of the team. Having an unregulated and unstructured requirements change mechanism, Company F appears specially prone to misunderstandings regarding the scope and the costs of development projects.

7) **Company G:** Company G is active in the branch of mechanical engineering; their software is mostly developed by contractors. Only one employee is responsible for in-house software development.

Requirements are termed very abstract, informal, and in natural language. They partly comprise use cases, in particular for validation purposes. They are normally communicated via email. In later project phases, prototypes are used to refine requirements further. Final acceptance tests are conducted in the form of field tests.

Company G sees their main problem in delays in software development. Those delays are attributed to unclear and volatile requirements. Further issues arise from test criteria that are derived from those unclear requirements; this can
lead to “forced” acceptance followed by a new project to implement the originally desired requirements.

B. Summary and Interpretation

All companies we visited for interviews faced problems with imprecise requirements. They relate those problems to the requirements process (Company A), implicit requirements (Company E), lacking expertise at the customer’s side (Company F). Even Company C, where separate contracts are signed for the construction of the requirements specification, faced disputes regarding product compliance.

In the response to the questionnaire (cf. Table I), only Company B always regulates the acceptance testing procedure. In the interviews, most companies we visited report according problems. Company A and Company E face problems due to differences in technical knowledge of the contract partners, Company G experienced “forced” acceptance. Only Company F seems to be content with the acceptance test procedure, yet their response to the questionnaire indicates that the acceptance test procedures are not fixed in the contract but just best practice.

Although all interviewed companies have experience with disputes (cf. Table I), none of the disputes has been brought to court. Only one of the companies has experience with settlements where legal advice was utilised. In addition, Company A stated the feeling that they do not fully understand all aspects of the arrangement made by their contracts. For Company D, most of the contracting terms are determined by the customer. That is, due diligence, understood as “the seller needs to fully understand all aspects of the arrangement with the buyer(s)” [7], can actually not be assumed for SME.

III. TOWARDS (LEGAL) CERTAINTY

Our approach to solve the legal and technical problems faced by SMEs involved in contracting for software consists of a new, modular software development contract and the new notions of checkable requirement and checking tool.

The contract we propose assumes that the contracting parties agree to designate a subset of the requirements as checkable requirements. A checkable requirement is a requirement for which there is a procedure to derive software specifications from it and a checking tool, i.e. a tool or procedure which soundly determines whether a software satisfies that software specifications. The contract codifies that the outcome of the checking tool applied to the delivered software is — with few and exactly specified exceptions due to mandatory regulations — binding for both parties, i.e., the checking tool provides a standardised acceptance test procedure for the checkable requirements. Thereby we address all three sources of uncertainty identified in Section I and thereby mitigate the risks of subcontracting for software: A checkable requirement is inherently exact because there exists a precisely determined checking procedure; for a checkable requirement, the checking tool provides a standardised acceptance test procedure, and the modular software development contract provides legal certainty.

This section is structured as follows. Checkable Requirements and checking tools are in turn defined in terms of backends, hence we firstly introduce the new notion of backends in Section III-A. We illustrate the wide scope of our notion of backends with a collection of examples in Section III-B. In Section III-C, we formally define checking tools, checkable specifications, and checkable requirements. In Section III-D, we discuss how to obtain checkable requirements for functional requirements of software using Interface between Requirements and Specification (IRS table).

Checkable Requirements and checking tools can be seen as an interface between computer science methods and tools, and the legal domain. Section III-E outlines how we use this interface to codify standardised acceptance test procedures in a modular software development contract. In Section III-F, we report on preliminary experiences with our approach.

A. Backends

Let \( D \) be a set of programs (called domain). Let \( \varphi \) be a property of programs from \( D \), i.e. there is a notion of property satisfaction between programs and \( \varphi \). We use \( p \models \varphi \) to denote that program \( p \in D \) satisfies property \( \varphi \). A (possibly non-deterministic) tool or procedure \( T \) which maps programs from \( D \) to a result from \{Yes, No, Unknown\} is called backend with respect to \( \varphi \) if and only if:

- \( T(p, \varphi) \) yields Yes only if the program satisfies the specification, i.e. only if \( p \models \varphi \) (or if the reason for yielding Yes is a bug in backend \( T \)).
- \( T(p, \varphi) \) yields No only if the program does not satisfy the specification, i.e. only if \( p \not\models \varphi \) (or if the reason for yielding No is a bug in \( T \)).

In other words, a tool or procedure \( T \) is a backend if and only if, whenever \( T \) “says Yes”, then this provides evidence for \( p \models \varphi \), e.g. by the algorithm implemented in \( T \), and whenever \( T \) “says No”, then this provides evidence for \( p \not\models \varphi \), e.g. in form of a counterexample. Let \( T_1, T_2 \) be backends with respect to program property \( \varphi \). \( T_2 \) is said to cause more positive certainty\(^1\) than \( T_1 \) in domain \( D \), denoted by \( T_2 \succeq T_1 \), if and only if \( T_2 \) yields Yes for some programs where \( T_1 \) yields Unknown, i.e. if \( T_1^{-1}(\text{Yes}) \subseteq T_2^{-1}(\text{Yes}) \).

B. Backend Examples

In the following, we elaborate on examples for backends ordered by increasing sophistication, that is, possibly increased effort needed to obtain positive results. Some of the backend examples we discuss below can be supported by manual effort. For instance, developers can often choose

\(^1\)The definition of causing more positive certainty reflects the observation that the contractors only gain satisfaction by Yes: if a backend yields Yes, then the customer obtains good software and the developer obtains the money (cf. Section III-E).
programming idioms which support a backend in confirming the absence of problems. In the case of static checkers (see below), developers can add annotations such as pre-/post-conditions and loop invariants to the code in order to enable the backend to confirm that the program satisfies a given invariant. The developing party has to fully understand their problem domain in particular the capabilities of the backends which are offered to customers. If the parties agree on a more sophisticated backend, which requires additional effort, this effort will be reflected in the overall cost estimation.

1) The Program Compiles (Trivial Example): A program property \( \varphi \) can be as simple as the requirement that a certain compiler \( C \) in a particular version \( V \) produces a non-empty executable if applied to the developed software. Possibly with the additional property, that no warnings are issued.

The corresponding backend is then a wrapper which applies compiler \( C \) in version \( V \) to the developed program and checks whether an executable was produced and yields

- Yes, if there is a non-empty executable, and
- No otherwise.

Note that in this case any termination of the compiler which does produce an empty executable leads to No, including abnormal termination caused by, e.g., compiler bugs or stack overflows in the parser. The domain in this case would be the set of all C programs.

2) Test Coverage: The percentage of branches covered by unit tests is also a program property \( \varphi \).

A backend corresponding to the property of complete coverage is then a wrapper which applies unit tests to the developed program in an appropriate framework, e.g., using gcov from the GCC suite, and yields

- Yes, on normal termination and indication of 100% branch coverage,
- No, on normal termination and indications branch coverage below 100%,
- Unknown otherwise.

3) Absence of Generic Errors: Static analysis tools such as frama-C [8] or static checkers such as VCC [9] can certify the absence of a set of generic errors such as invalid pointer dereferences or undefined program behaviour in the sense of the C standard. Such tools typically do not provide a simple yes/no output, but, as in the case of frama-C, assign each program statement a status. Statuses include surely valid and surely invalid, but also never tried, which applies to originally unreachable code or code which is unreachable because it is only reached via a generic error [10], [11].

So one can define the program property absence of generic errors as all assertions related to safe memory access have status surely_valid or never Tried. The corresponding backend would run frama-C with the plugins RTE and value analysis [10], [11] and yield

- Yes, if all assertions related to safe memory access have status surely_valid or never_tried,
- No, if at least one assertion related to safe memory access has status surely_invalid, and
- Unknown otherwise.

That is, the backend implements the decision about what is a problem report by mapping the output of the bug-finding tool to Yes or No.

The contract parties have to be aware of the exact definition of the program property. Programs, which are safe on a specific platform (e.g. because signed overflows may have defined behaviour as on the Intel x86) may not be detected correct by frama-C due to implementation defined behaviours. Frama-C makes assumptions about the C implementation used. Thus the results may not correspond exactly to the implementation used in reality. Nonetheless, successful verification of the program property provides additional strong evidence of the quality of the program in the sense of Jackson [12].

4) Invariant Satisfied: In addition to absence of generic errors, static checkers such as VCC [9] and static analysis tools such as frama-C [8] can also certify that a program satisfies a functional specification, i.e. a relation between program variables, given as an invariant.

The corresponding backend would run VCC and yield Yes if the VCC output indicates that the invariant has been proven to hold, and Unknown otherwise.

Note that there are two classes reasons for the latter outcome. Firstly, the checking procedure may simply run out of memory or be terminated after an upper bound on computation time is reached. Secondly, the program may not be sufficiently annotated, e.g., the given loop invariants may not be strong enough to conclude that the invariant is satisfied by the program. As static checking is not a complete method, i.e. there is no guarantee that, if the program is correct wrt. a given invariant then the invariant can be proven. In both cases, the developer can support the backend as discussed above.

5) Certification: The definition of backends is not limited to tools. The contract parties can also agree on an authority which offers expert reviews of programs. The result of such a review just needs to be mapped to Yes, No, or Unknown.

C. Checkable Specifications and Checking Tools

A checkable specification is a finite set of pairs

\[ \Phi = \{(\varphi_1, T_1), \ldots, (\varphi_n, T_n)\}, n \in \mathbb{N}_0, \]

where \( T_i \) is a backend with respect to property \( \varphi_i \).

A checking tool result of program \( p \) with respect to a checkable specification \( \Phi \) is an assignment of a status to each program property in \( \Phi \), i.e. a set

\[ \{(\varphi_1, s_1), \ldots, (\varphi_n, s_n)\} \]

where \( s_i \in \{\text{Yes, No, Unknown}\} \) such that \( s_i \) coincides with a result of the backend \( T_i \) applied to \( p \), i.e. \( T_i(p, \varphi_i) \).
yielded $s_i$. A **checking tool** is a (possibly non-deterministic) tool $S$ which maps checkable specifications to checking tool results. In other words, a checking tool groups multiple backends together and applies them to the same program.

### D. Checkable Requirements

We call a requirement a **checkable requirement** if and only if it is already a checkable specification or if there exists a method or procedure to derive a checkable specification from it. That is, the properties discussed with the backends in Section III-B are examples for checkable requirements.

When software for systems that interact with a physical environment is to be developed, that is, at the time when a software development contract is negotiated, the requirements are typically **system requirements** and only to a limited amount checkable specifications. A system requirement is a requirement which is stated in terms of domain phenomena, whose presence or absence at a given point in time can be measured in the physical environment. They express that the software to be developed is supposed to control a physical system in a certain way, how this behaviour is realised is typically left to the contractor. For instance, when a software shall realise the monitoring of the backup battery of a system, one requirement could be

“For each point in time, the battery-low warning light is on if and only if the battery is low, i.e. the current battery voltage is below 6.6 V”,

or, formalised using LTL,

$$G \text{(battery-low warning light is on} \iff \text{battery is low (} V_{\text{bat}} < 6.6 V \text{)}). \quad (1)$$

Here, “battery-low warning light is on” and “battery is low” are domain phenomena.

To establish whether a software realises system requirements without executing it on the target platform, for instance, because the target platform is still under development, we face the problem that the program only operates on program variables, not on domain phenomena. On programs, we can only evaluate and analyse **software specifications**, i.e. properties of the evolution of program variables over time, but not system requirements.

Following the approach presented in [13], software specifications can be derived from system requirements and an **Interface between Requirements and Specification (IRS-table)** provided by the programmer. If the obtained software specification is an invariant, it is a checkable specification with the backend discussed in Section III-B4. Otherwise it may still be a backend if certification in the sense of Section III-B5 is available as a backend.

For example, assume that the state of the battery warning light is represented by the variable $SCL$ and the battery status by $ADCR$. Further assume that the battery warning light is on if the former variable has value 1, and a low battery is characterised by a value below 33152 of the latter variable. Then the programmer would provide the IRS-table shown in Figure 1.

#### E. Contract

Sections III-C and III-D can be seen as an interface between methods and tools from computer science and the legal domain. A legal contract can refer to the precisely defined notions of checkable requirement and checking tool. Using this interface, a standardised acceptance test procedure has been codified in a modular software development contract [14]. The contractual basis consists of a framework contract which contains general provisions including definitions as well as clauses relating the collaboration of the parties and the software development contract. By using several contractual modules the development contract can be customised to the needs of the respective project. In particular, checkable requirements are defined as a subset of the overall requirements specification which the contract parties agree upon and whose contract compliant realisation is checkable by a (fixed) checking tool.

To allow customers and contractors to decide the latter property, we provide a prototype tool **TESA** which offers requirements such as absence of generic errors (cf. Section III-C) to choose from, and means to construct functional requirements. For the requirements that can be chosen or constructed with TESA, it is known that the contract compliant realisation is a checkable requirement and hence checkable by the checking tool. The TESA tool can export checkable requirements into a PDF with an electronic fingerprint and embeds into the PDF a machine readable XML version of the requirements. In this way, checkable requirements can become part of the overall requirements specification and are automatically marked as such.

In addition, for each software development project, an individual contract is signed which refers to the framework contract. The individual contract in particular defines the software which is to be delivered. Next to development of the software, the developer is now always obliged to provide an IRS-table (cf. Section III-D).

Checking compliance with the contract is regulated in two clauses, one for checkable requirements and one for other requirements. The contract states that the realisation of checkable requirements is tested with and only with the checking tool. The checking tool is supposed to be applied according to its usage instructions to the program which

<table>
<thead>
<tr>
<th>Domain Phenomenon</th>
<th>$Expr_{\text{S}}(\text{Var})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>battery low warning light is on</td>
<td>$SCL == 1$</td>
</tr>
<tr>
<td>battery low (&lt;6.6V)</td>
<td>$ADCR &lt; 33152$</td>
</tr>
</tbody>
</table>

Figure 1. The IRS-table for the requirement (1) which maps the domain phenomena to boolean expressions over program variables $\text{Var}$. [13]
is provided by the developer. The checking tool result is declared to be binding.

As backends may yield the result “Unknown”, there is an exit option in the contract. The checking tool result is not binding for those requirements $\varphi$ where

1) one of the parties provides evidence that the result provided by $\varphi$’s backend is erroneous,
2) the parties agree and state that they want to consider that result erroneous, or
3) if the result provided by $\varphi$’s backend is “Unknown” and the checking tool result consists only of statuses “Yes” and “Unknown”.

In these cases, the clause for other requirements applies. That is, the customer determines the testing procedure for those checkable requirements for which the checking tool result is no longer binding within the conditions stated by the development contract.

Note that, by the construction of the contract, in particular the developer has a strong interest in obtaining a positive result from the checking tool. If the checking tool yields the positive result “Yes” for each checkable specification (which in turn are obtained from the checkable requirements (cf. Section III-D)), then (given that all other requirements have been met) the contract regulates that the payment is due, i.e. that the contract is satisfied. Thus there is a strong motivation for investing the (possibly additional) effort which is necessary to enable the checking tool to yield the positive result. Furthermore, given the requirements are realisable and given adequate capabilities at the contractor, our approach ensures successful contracting for both sides.

F. Proof of Concept

We have applied the proposed approach a posteriori, i.e. not in the context of an actual software development contract, to a real-world version of the battery control example referenced in Section III-D and to the software for the nodes of a preliminary version of a wireless sensor network which detects shock waves hitting a wall [15]. In both cases, errors in the software were detected. After fixing the errors and providing necessary annotations manually, the software was found to be correct, that is, we obtained a positive checking tool result.

We are currently developing a domain definition, i.e. a characterisation of C programs, which allows for the fully automatic application of frama-C.

IV. RELATED WORK

There exists a number of studies on requirements engineering and software development practices in (very) small to medium enterprises [3], [5], [6], [1]. Our survey of the current situation (cf. Section II, for more details see [2]) does not disagree with their general findings. But contrary to those studies, we focus on subcontracting for software and the associated legal and technical issues. Notably, in [1] two enterprises wished for more research regarding subcontracting for software in SMEs, both from the customer and from the contractor point of view.

In the body of work on legal requirements in requirements engineering (see, e.g., [16]), we think the work closest to ours is [7] although the scope of the paper is limited to the time after the contract has been awarded and in particular assumes that due diligence has been applied. They identify problems in a much broader scope including tracing, penalty clauses, and incentive payments. Regarding requirements analysis, they focus on risk assessment and communication among contractors, and regarding contract compliance, they limit the discussion to the case of trace matrices “that show how each requirement in the contract is met in design, during execution, and in the finished product”.

In [17], an approach to check whether business processes described in BPMN are compliant to the conditions of a contract is outlined. They propose to formalise contract conditions with FCL (formal contract language). If the subject of a contract is the development of a set of business rules which comply with existing other contracts, then FCL could be used to formalise the requirements on the business rules. A tool which decides compliance would then be an automated acceptance checking procedure in our setting.

An approach “to help assure that operational practices comply with standards and regulations” is described in [18]. In their terms, we consider top-level obligations and verification sets without delegation. Our notions of checkable specification and backend refine verification sets for system requirements by also considering inconclusive results. Their approach lacks a discussion of contractual regulations in case of negative evaluation of their verification predicates. Requirements verification as described in, e.g., [19] is not subject of our work but a premise for successful subcontracting. A formalisation of requirements for the purpose of their verification can directly be re-used in our approach. Furthermore, the creation of a requirements document is often the subject of a contract on its own where consistency is a requirement on the requirements document. Our contractual framework could support this case with requirements verification procedures serving as automated acceptance checking procedure.

V. CONCLUSION AND FURTHER WORK

We tackle a main challenge that arises when subcontracting for software development in SMEs: legal uncertainty. We identify three sources of uncertainties faced by SMEs and outline a possible approach to resolve them. A modular legal contract codifies the mutual agreement that checkable requirements are verified by the agreed checking tool exclusively. In Project Salomo [2], researchers in computer science and law cooperate to elaborate this approach.

By construction of the contract, both, the contractor and the customer have a strong interest in obtaining positive
results for checkable requirements from the checking tool since positive results mean certainty. Our contract is well-suited for a gradual introduction of formal methods into the practice of SMEs because any choice of backends is supported as long as both parties agree. Examples of backends for formal methods are very scalable (or light-weight) formal methods like test generators and static analysis, sophisticated methods such as static checking, and even proofs by external reviewers. The increased effort for formal methods promises a significantly increased confidence in the quality of the delivered software. We are currently looking for partners for an a priori evaluation of our approach in practical subcontracts for software development.

Our approach as presented here is a starting point for successful subcontracting for software development in SMEs and as such contains a number of challenges that are also independently important: On the computer science side, there is much more research needed regarding the automation of formal analysis and then the interpretation of analysis results that is, the construction of checking tools from backends. Furthermore, the IRS approach needs to be extended to support a broader range of embedded software.

The integration into existing software development processes needs to be improved. It seems that current requirements management tools do not appeal to SMEs, although the resulting problems seem severe. Especially managing changes in requirements and tracing them through the product’s life-cycle is problematic for SMEs. As the prevalent change management technique seems to be the exchange of word processor or spreadsheet documents, new practical methods need to take this into account. For the IRS approach those methods also need to provide a machine-readable, formal version of the checkable requirements.

On the jurisprudence side, further work in particular needs to integrate into the modular contract means to legally support traceability, i.e. to trace refined requirements against top level requirements [7] but also to trace (refined) requirements against dynamic customer requests, which only appear during actual software development, as assessed in a survey [2] and in interviews. Also, considering a concept of delegation [18] in the contract could allow for more flexible acceptance test procedures.

Additional challenges arise from the management of change-requests. Especially defining a process which requires as little as possible additional overhead in the development process and which ensures that changes in requirements are recognised not only between partners but also in court seems challenging.

REFERENCES