A hybrid approach to assessing the maturity of Requirements Engineering practices in agile projects[☆]

Mirosław Ochodek[a,*], Sylwia Kopczyńska[a], Tomasz Jerzy Paszek[b]

[a] *Poznan University of Technology*
Faculty of Computing, Institute of Computing Science
ul. Piotrowo 2, 60-965 Poznań, Poland
Tel.: +48-61-6652944
Fax: +48-61-8771525
[b] *Unity Technologies — unity3d.com*

Abstract

Context: Requirements Engineering (RE) is one of the key processes in software development. With the advent of agile software development methods, new challenges have emerged for traditional, prescriptive maturity models aiming to support the improvement of RE process. One of the main problems is that frequently the guidelines prescribed by agile software development methods have to be adapted to a project's context to provide benefits. Therefore, it might be naive to believe that it is possible to propose a prescriptive method of RE process improvement that will suit all agile projects without any alteration.

Objective: The goal of this paper is to propose a hybrid approach to assessing the maturity of agile RE, which combines elements of prescriptive and problem-oriented improvement methods.

Method: We followed the Design Science Research paradigm to develop an artifact—a maturity model and assessment method. We based our proposal on literature studies and feedback from IT professionals.

Results: We proposed the Requirements Engineering Maturity Model for Agile software development (REMMA), which includes two main components: a maturity model for agile RE (a set of state-of-the-art agile RE practices) and an assessment method that makes it possible to evaluate how well the agile RE practices are implemented. The method takes into account dependencies between practices and the necessity to adapt them to a certain project context.

Conclusions: The proposed approach seems to be a useful tool to support the process of improving RE practices in agile projects.

Keywords: Requirements Engineering, process assessment, process maturity, process improvement, agile

1. Introduction

Requirements Engineering (RE) is one of the key processes in software development. It aims at identifying, analyzing, documenting and validating requirements for the system to be developed. It has been observed that when the RE process is orchestrated properly it can favorably influence the whole software development process [1, 2]. Conversely, problems related to requirements were often identified as main causes of famous IT projects' failures and software systems' disasters [3, 4, 5, 6].

With the advent of agile software development methods, such as eXtreme Programming (XP) [7], Scrum [8], or Crystal Clear [9], new challenges have emerged for RE.

The agile methods advocate for the iterative discovery of requirements rather than an exhaustive up-front elicitation. They aim at delivering software faster and with the focus on ensuring that it meets a customer's changing needs in environments where developing unambiguous and complete specifications is often impossible or even inappropriate [10].

In order to support evolving requirements and allow for introducing changes at every stage of the project, agile RE focuses on preserving continuous and intensive communication with the customer and advises producing only minimal, evolving requirements specifications [10, 11, 12]. As a result, the approach to planning in agile software development projects was changed from traditional—predictive to adaptive [11].

The need for adaptivity is deeply-rooted in the principles of the Agile Manifesto [13] that advise teams to regularly reflect on how to become more effective and make use of the lessons they learned. Although this principle makes

[☆]This project was funded by the Poznan University of Technology internal grant DS 91-518.

[*]Corresponding author
Email addresses: mochodek@cs.put.poznan.pl (Mirosław Ochodek), skopczynska@cs.put.poznan.pl (Sylwia Kopczyńska), tomasz@unity3d.com (Tomasz Jerzy Paszek)

the team responsible for improving the software development process, many agile software development methods introduce additional roles responsible for driving the improvement process, e.g., Coach in XP [7], or Scrum Master [8] (we will refer to all these roles as agile coaches). The agile coaches are supposed to have decent knowledge of agile methods to help teams find the best solutions to their problems, as well as be able to convince management to allocate necessary resources to support the improvement process.

Among the different tools that agile coaches can use to support the improvement process, they can employ one of the existing agile maturity models (e.g., AMM [14], SAMI [15]). These tools might be useful for discovering problems in their projects, as they *prescribe* sets of guidelines related to the proper usage of practices in agile projects.

Unfortunately, prescriptive approaches to improvement, based on existing maturity models, seem to have an important deficiency in the context of agile software development. The state-of-the-art agile practices proposed by the agile methods and maturity models might work out-of-the-box for software development projects fitting into what is sometimes called the "*agile sweet spot*" (i.e., small, co-located teams; an on-site or available customer representative; an emphasis on coding and testing early; and frequent feedback into updated requirements) [16, 17, 18]. Unfortunately, one can never be sure to what extent the specific characteristics of a particular project make it fit in or outside that spot. Therefore, the *problem* is that relying purely on maturity models as a means of driving the improvement process might be insufficient for many agile projects. What is needed here is the ability to combine prescriptive guidelines provided by agile methods and agile maturity models with the lessons learned from applying agile methods in specific contexts. The latter usually come as the results of following problem-oriented (inductive) approaches to improvement (i.e., team identifies a problem and tries to solve it taking into account specific context of the project).

In the paper, we would like to address that problem by proposing a maturity model and assessment method to support the improvement of the RE process in agile projects. The method is called the Requirements Engineering Maturity Model for Agile software development (REMMA). It is a hybrid method in the sense that it combines elements characteristic to both prescriptive and inductive approaches to process improvement.

REMMA was designed to be used by project teams, and especially agile coaches, to allow them to assess the implementation of RE practices in their projects with the use of the following components:

- a prescriptive component that provides a set of guidelines related to the appropriate implementation of the state-of-the-art agile RE practices (including guidelines related to preserving the synergetic nature of the agile practices [19]);
- an inductive component in the form of a model which explicitly allows incorporating information about how the specific context of a project affects the usage of agile RE practices (in the paper, we discuss four examples of such contexts: the need for adherence to XP and Scrum, globally distributed software development, and the development of safety-critical systems);
- the assessment method which allows evaluating the implementation of agile RE practices in a project, taking into account guidelines related to the implementation of state-of-the-art agile RE practices, preserving synergy between the practices, and assessing their appropriateness for the project context.

The paper is structured as follows. In Section, 2 we introduce the conceptual framework of maturity models and assessment methods. In Section 3, we present our research approach and the architecture of REMMA. In the following sections 4 and 5, we describe the REMMA model and the assessment method respectively. Related work is discussed in Section 6. Finally, we conclude the paper in Section 7.

2. Maturity models and assessment approaches

2.1. Maturity models

Software process assessment and improvement methods can be divided into two categories: inductive (problem-based) and prescriptive (model-based). The inductive methods regard existing problems and weaknesses as the driving force for improvement, while the prescriptive methods focus on the alignment with best practices [20].

Each prescriptive method proposes a set of best practices which when followed will contribute to the success of software development. The existing sets of best practices are dedicated either to software development as a whole, e.g., the CMM model family [21], Agile Maturity Model (AMM) [14], or to a certain project area or aspect of software development, e.g., Requirements Engineering: A Good Practice Guide (REGPG) [21].

The practices in the prescriptive methods are frequently divided into several categories and arranged hierarchically. For example, the aforementioned REGPG model proposes a set of 66 practices and two parallel classification schemes of practices. The practices are divided into 8 categories based on the stage of RE in which they are used, e.g., elicitation, documentation, and independently into 3 categories that reflect the level of RE adoption in a project: basic, intermediate, advanced.

Another example of a hierarchical model is the Requirements Engineering Process Maturity (REPM) proposed by Gorschek [22]. The topmost elements of the model are related to the main activities of RE (e.g., elicitation) and are

called main process areas. Below them are sub-process areas (e.g., stakeholder identification) and actions, which can be seen as equivalent to very specific practices (e.g., ask executive stakeholders).

Interestingly, most of the prescriptive methods unanimously use the term *practice* to refer to certain activities, however, the results of the literature study by Paivarinta and Smolander [23] show that the meaning of the term differs between studies. For instance, some authors understand practice as a set of thoroughly predefined procedures executed by humans while others perceive it as a set of less organized activities. Therefore, in order to avoid misunderstandings related to the meaning of the term in this paper, we would like to propose the following definition of practice:

Definition 1. *A **practice** is a pattern of actions or behavior that is recurrently executed by humans in order to achieve certain goals.*

Although the prescriptive methods propose sets of best practices to be followed, they usually allow for some flexibility in their usage. For instance, McMahon [24] left a place for alternative practices to be added to the proposed model if they aim at achieving the same goals. Niazi et al. [25] and Gorschek [22] allowed for ignoring certain practices if they seem to be inappropriate for an organization.

To indicate the degree to which the existing processes are institutionalized and effective [26] with respect to a certain model, the terms *maturity* and *maturity model* were introduced. The first term—maturity model is understood as a set of best practices with their classification schema (e.g., process area, adaptability level). It is expected that the more convergent the implementation of processes is with the maturity model, the higher its *maturity level*. Therefore, in addition to maturity models, most prescriptive methods propose corresponding, dedicated assessment methods for determining the level of maturity.

When discussing the idea of maturity, it is important to emphasize that maturity can be considered at different levels of an organization. For instance, the CMM family of methods introduced the concept of *organizational maturity*, which refers to the extent to which an organization has explicitly and consistently deployed processes that are documented, managed, measured, controlled, and continually improved [27]. In addition, each process in the organization can be investigated from the perspective of its *capability*. Capability characterizes the ability to achieve current or projected business goals [26]. Organizational maturity and process capability can be further combined into what is called *organizational capability* [28].

Looking at the concept of maturity from the perspective of agile software development methods, we can notice that agile methods focus on improvement mainly at the project level [29, 30, 31]. Therefore, although the usability of popular CMM/CMMI frameworks in the agile context has been demonstrated [24], the application of these methods, and the ability of agile organizations to achieve certain levels of maturity in particular, has become a widely debated concern [32, 33, 34, 35, 29]. In addition, several works suggest that agile has its own practices and goals [36], and some plan-oriented practices do not fit the agile context. Thus, some researchers proposed mappings between agile software development methods and the CMM family methods [37, 38, 39, 30, 40, 41, 42, 31].

Moreover, Fontana et al. [26] noticed that recently researchers have started searching for new definitions of maturity in the context of agile software development. Unfortunately, it seems that no substantial conclusions have been reached so far.

2.2. Assessment Approaches

The main goal of assessment approaches used with maturity models is to provide feedback on the maturity of an organization, process or project, which consequently can stimulate the organization to take some corrective actions [40].

The kind of feedback one receives depends on the outputs of the assessment method being used. For instance, some assessment methods provide output in the form of a list or quantity of (in)sufficiently implemented practices, e.g., Karlskrona test [43], while using other methods, such as CMMI SCAMPI^{SM} [44], results in comprehensive reports containing summaries of strengths and weaknesses documented for each process area in the scope of assessment, as well as information about the maturity level.

The maturity level is often determined with the use of questionnaires or checklists, based on a sum of the frequencies of answers to a series of Likert items or yes/no questions. This procedure originates from early approaches, such as CMM/CMMI [27], but it also appears in the methods proposed for agile software development, e.g, Agile Maturity Model (AMM)[14].

For instance, in AMM the assessor answers a series of multiple choice questions (Yes, Partially, No, N/A, Don't know) verifying if key practices are used. The result is a single measure that corresponds to the percentage of sufficiently implemented practices (a 'Yes' answer counts as one point, and 'Partially' counts as half a point). Based on the measure's value, it can be determined whether a given level of maturity has been reached (e.g., 86% to 100% — fully achieved). A similar mechanism was proposed in the previously mentioned REGPG.

In REGPG, the maturity of the RE process is based on a sum of points scored for practices in each of the three categories: basic, intermediate, advanced. However, the dependencies between practices are not considered. Therefore, it is possible to achieve a higher level of maturity by using a greater number of advanced practices without mastering the basic ones; this was reported as an issue by the REGPG authors during the method's validation [45].

Other authors propose determining the level of maturity based on the average number of points scored for the sufficient implementation of certain practices or actions.

For instance, Gorschek in REPM [22] proposed determining the level of maturity based on the average of sums of points received for each main process area.

Niazi et al. [25] in their RE Maturity Measurement Framework (REMMF) provided a more advanced assessment approach. They introduced a measurement instrument built upon 3 dimensions: Approach (concerning management support), Deployment (concerning depth and consistency of application) and Results (concerning depth and consistency of positive results over time). The evaluation of every possible practice in REMMF is represented by a description of the situation it denotes. For instance, for the Approach dimension equal *weak* (2) means "management begins to recognize need"[25]. The quotient for each practice is calculated as an average of the 3 scores (one for each dimension), and afterward an average of the quotients for all practices in the same category is calculated.

Finally, some of the existing assessment approaches can be used to determine a maturity level that a project can achieve. For example, Sidky's Agile Adaptation Framework (SAMI) [15] might indicate the readiness of an organization to adopt agile practices. This is based on the identification of indicators of possible problems and impediments to introducing agile practices.

3. Research methodology and solution design

The background information and discussion of problems contained in literature related to maturity assessment in software projects justifies conducting distinct research activities that intend to solve the identified practical problem of *assessing RE maturity in agile projects*.

Because the goal of our research is to create a method, we oriented our approach towards Design Science Research (DSR) [46], a problem-solving paradigm that focuses on creating and evaluating artifacts and solutions for practical purposes. In particular, we decided to follow the guidelines provided by Wieringa [47].

According to Wieringa, there are two types of research problems in DSR: practical problems and knowledge questions. A practical problem is one whose solution requires an artifact which, when introduced, changes the world so that it better contributes to the achievement of a given goal. Meanwhile, a knowledge question is about closing knowledge gaps about the world.

In our research approach to solve the identified problem, we systematically created an artifact—the REMMA method, which allows assessing the maturity of RE in agile projects. In order to create the artifact, we executed a series of engineering activities and answered several knowledge questions. Finally, we plan to validate our proposal in the field by executing what in DSR is called an empirical cycle. The research process and the resultant architecture of the proposed artifact are presented in Figure 1.

We began the research process by investigating the identified problem and defining the main quality attributes of the artifact under development that would allow for its acceptance by practitioners.

In order to identify these attributes, we referred to the theoretical framework of the Technology Acceptance Model (TAM) [48]. According to TAM, the behavioral intention of using a new technology is determined by its *perceived usefulness* (PU) and *perceived ease of use* (PEOU). We decided to focus on determinants of PU and PEOU that, by definition, are not bounded to computer systems. Therefore, among different determinants of perceived usefulness, we selected—job relevance, output quality, and result demonstrability. Looking from the perspective of perceived ease of use, we decided to focus on self-efficacy (an individual's belief that he or she can perform the maturity assessment in his or her project), and perception of external control, which is the degree to which an individual believes that organizational and technical resources exist to support the use of the artifact (e.g., management support). Finally, the method should be cost-effective in respect to the effort required to perform the assessment.

In the same stage of our research, we performed a literature review of approaches to improvement and maturity assessment in software development so as to better understand the mechanisms of existing methods, as well as to identify their advantages and disadvantages. An additional result was the development of a conceptual framework for the proposed method.

One of the findings of the literature study was that the term agile maturity is used inconsistently [26, 49]. Therefore, we would like to state explicitly how it is going to be understood in the context of this paper. The proposed definition is based on the origins of the agile movement—the Agile Manifesto, with its four values and twelve principles.

Definition 2. *Agile maturity* is the degree to which a software development project follows the agile values and principles defined in the Agile Manifesto.

From the agile values and principles follow agile practices, as it is a practice-led paradigm [36, 50, 51]. In the paper, we will use the term *agile practice* to refer to a practice that supports achieving at least one agile value or principle, and does not interfere with achieving any other agile value or principle (for the definition of practice see Definition 1). Since agile practices answer the question of "*how* we can drive the software processes to obtain agility" [52], we formulated the following assumption:

Assumption 1. There exists a set of agile practices that, when used in a software development project, allows the achievement of agile maturity. This set will be referred to as the *agile maturity model*.

An agile maturity model can be used to support the prescriptive-oriented approach for improvement. Similarly to the findings of some other authors, e.g., [23], we also believe that some prescription is required in agile software development. In fact, agile development methods are to some

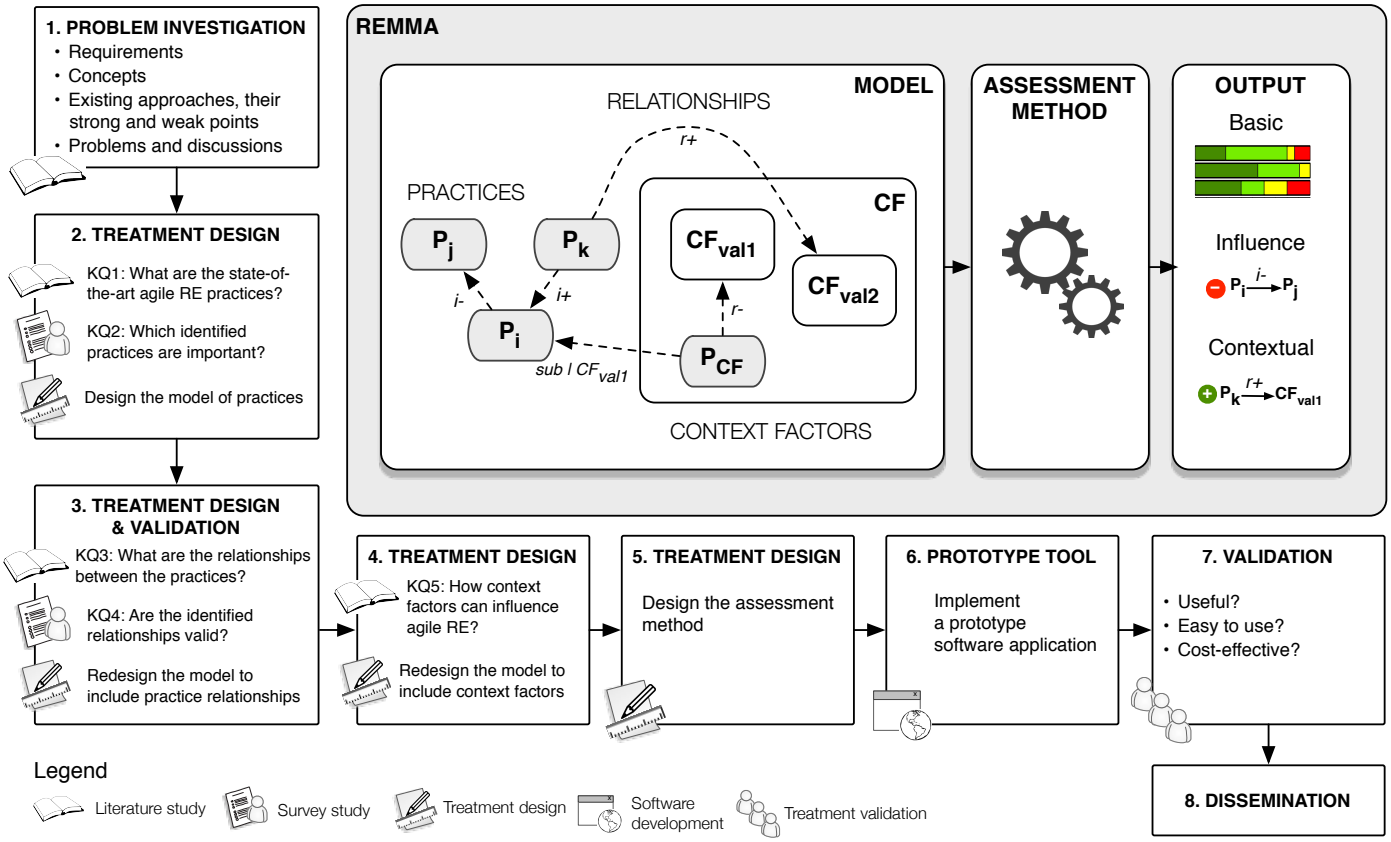

Figure 1: The architecture and process of designing the artifact—the REMMA method.

extent prescriptive, in the sense that they usually propose sets of best practices to follow. Considering that, it seems possible to think about the maturity of agile projects in terms of how well they implement the recommended state-of-the-art practices. Such an approach is also used in other maturity assessment approaches, e.g., the family of CMMI [27] methods, Agile Maturity Model (AMM) [14]. This led us to make the following assumption:

Assumption 2. *The level of agile maturity* can be determined as the *alignment* between the practices implemented in a software development project and the agile maturity model.

The term *alignment* might be understood as the degree to which the practices in a project allow achieving the same goals as those in agile practices. However, we are going to extend this definition by taking into account several specific characteristics of agile practices.

The first specific property of agile practices is that they seem to be valuable to a project if implemented standalone, however the true power of agile methods seems to lie in a synergistic combination of these practices [19, 53]. For example, using the XP practice of documenting requirements as lightweight user stories without an available customer, willing to closely collaborate with the development team and answer questions related to the requirements, constitutes a visible risk to the project's success.

Another important characteristic of agile practices is that they are usually proposed for those projects that are close to the "agile sweet spot." The result, as some authors have observed, is that in some environments it may be necessary to extend, drop or alter agile practices in order to successfully adopt agile methods [54, 55].

Considering these two properties of agile practices, we would like to propose the following extended definition of alignment to the agile maturity model.

Definition 3. *Alignment* is the degree to which the practices used in a software development project (in short, the practices) match the agile practices defined in an agile maturity model; in particular, this includes:

(a) basic alignment—the degree to which the practices allow achieving the same objectives as the agile practices prescribe;
(b) influence alignment—the degree to which synergy between the practices is preserved as it is preserved between agile practices;
(c) alignment to the context—the degree to which practices are appropriate for the environment in which they are used as the agile maturity model prescribes.

The definition of *alignment* is reflected in the architecture of the REMMA method presented in Figure 1. The

core component of the architecture is the REMMA model containing a catalog of state-of-the-art *agile RE practices*. In addition to agile practices, the model introduces different kinds of *relationships* between the practices in order to be able to capture the synergistic nature of agile practices. Moreover, the model enables including information about the project context (*context factors*) which, when present (take a specific value), might cause alterations in the agile maturity model, e.g., substitute a practice with another if lessons learned suggest that the original one does not fit the project context, or indicate which practices should be followed in specific project contexts to mitigate certain risks related to the adoption of agile methods.

Examining the alignment of the practices performed in a software development project to the agile maturity model makes it possible to combine the two improvement ideas: prescriptive and problem-based. The prescriptive approach is based on the examination of alignment with RE agile practices which constitute our agile maturity model (basic alignment) and investigating if synergy between the practices is preserved in the project (influence alignment). The problem-based approach might be realized by introducing gained experience as context factors adopting the initial agile maturity model (contextual alignment). Thus, our proposal might be seen as a hybrid approach, based on a model of agile practices and using an assessment mechanism to identify areas for improvement.

Having defined the aforementioned concepts and the idea behind our method, we proceeded to design the details of our solution in several iterations.

Our first iteration was dedicated to creating a model of practices by answering the knowledge question *What are the state-of-art agile practices for Requirements Engineering?* (KQ1) Thus, we performed a literature review to elicit agile practices, then analyzed them and proposed the catalog of Agile RE practices. In addition, we conducted a survey to determine the *importance of the proposed practices for agile projects* (KQ2). The two steps are described in the paper by Ochodek and Kopczyńska [56]. These resulted in a model of practices categorized according to their impact on the success of an agile project.

In the following iteration, we extracted information about relationships between agile practices defined in the literature and combined it with our experience and knowledge (KQ3). A list of the identified relationships was presented to agile practitioners. We asked them to state *if the proposed relationships are valid* (KQ4). This step refined the REMMA model, which at this stage consisted of practices and influence relationships among them.

In the third iteration of the research process, we designed the component of the model that allows modeling the influence of project context on the agile maturity model. This component, by definition, is supposed to be extended by users, based on their experience of using agile software development methods in different environments. However, in order to verify the proposed model, we decided to perform a literature review and propose four exemplary context factors.

Subsequently, we designed an assessment method for the model and implemented a prototype software tool[1] to support usage of the method and allow for its validation in an industrial setting.

Finally, we plan to execute an empirical cycle to validate our proposal. We want to carry out an exploratory case study in a realistic context to understand to what extent the method satisfies its requirements: is perceived as useful (i.e., provides results that correctly reflect the current state of implementation of RE practices in a project), easy to use (can be used by project team), and cost-effective (project team can afford using the method).

Ultimately, we believe that the dissemination of results might provoke further steps of DSR, namely the implementation and evaluation of the REMMA method by practitioners in various organizations.

4. The REMMA model

The REMMA model consists of three components: the catalog of agile RE practices, relationships between practices, and context factors. In the section, we present each of the components and discuss the steps we took to develop all of them.

4.1. The REMMA catalog of practices

In order to create a catalog of practices to be included in REMMA, we had to answer the knowledge question KQ1 (*what are the state-of-the-art agile RE practices?*) We decided to answer this question based on a literature review. The review resulted in 31 Agile RE practices that can grouped into the following six areas. For details about the review refer to the paper by Ochodek and Kopczyńska [56].

- PA1: Customer involvement
- PA2: Envisioning a product
- PA3: Knowledge sharing & communication
- PA4: Acceptance testing
- PA5: Requirements elicitation, analysis, documentation, and management
- PA6: Planning

For the sake of brevity, the definitions and discussion of 31 practices included in the REMMA practices catalog are presented in Section Appendix A. In addition, the catalog contains another outcome of the study—definitions of minimal quality requisites for the practices.

In addition, we wanted to characterize practices according to their importance in the context of agile software development projects (KQ2). We assumed that the

[1]The tool is available on-line: http://remma.cs.put.poznan.pl

importance of practices might be used to value identified problems.

Definition 4. ***Importance of a practice*** *reflects its potential influence on an agile project.* It is expressed using the following three-point ordinal scale:

- *critical*: a practice seems crucial in almost any agile project;
- *important*: a practice should be used in most agile projects;
- *additional*: an auxiliary practice that is recommended to be used.

We decided to ask practitioners how do they perceived the importance of REMMA practices for the success of agile projects. We decided to collect the opinions through a web-based survey, which is reported in the paper by Ochodek and Kopczyńska [56].

4.2. The REMMA relationships between practices

While performing the mapping of candidate practices onto theories of RE and agile software development, we noticed that studies related to agile RE frequently refer to practices that seem more suited to other areas of software development (e.g., project planning), or to practices that have a visibly much broader scope than RE (e.g., communication). This could be evidence that agile practices are interdependent. For instance, it is difficult to make a clear distinction between RE and planning activities—they seem intertwined and synergistic. An example is that the agile approach to planning short iterations that bring business value would be difficult to implement if the customer did not prioritize requirements.

In the REMMA model, we expressed this synergistic nature of agile practices through influence relationships between practices.

Definition 5. The existence of an ***influence relationship*** *between practices A and B ($A \xrightarrow{i} B$) implies that the (insufficient) implementation of practice A can influence (enforce or diminish) the effectiveness of the implementation of practice B. Practice A is called the* ***affecting practice****, while practice B is the* ***affected practice****.* We distinguish three types of influence relationships:

- *positive* ($A \xrightarrow{i+} B$) — the proper implementation of the affecting practice A can have a positive influence on the affected practice B;
- *negative* ($A \xrightarrow{i-} B$) — the insufficient use of the affecting practice A can have a negative influence on the affected practice B;
- *positive and negative* ($A \xrightarrow{i-} B$) — the affecting practice A can have both positive and negative influence on the affected practice B, depending on how well the affecting practice A is implemented.

The knowledge question (KQ3) that we had to answer at this stage was: *what relationships exist between practices in the REMMA catalog?*

In order to identify the influence relationships between the practices, we once again referred to literature study and our experience. As a result, we defined a set of 16 candidate relationships. We decided to validate the set by conducting a survey (KQ4).

We prepared a web-based questionnaire that consisted of 25 items—21 items related to candidate relationships as well as 4 demographic questions. Each item aiming to validate a relationship had the form of a statement indicating the existence of the relationship, e.g., *covering requirements with acceptance tests makes preparing automatic test cases more effective.* In the case of negative relationships, the sentences were negated. For the positive-negative relationships, we constructed two separate items. Respondents were asked to express their (dis)agreement with each item using a five-point Likert scale. Besides, they could state that they did not know.

In order to investigate whether respondents were committed to the task, we added three additional relationships to the survey questionnaire that we perceived as *fake* relationships.

We administered a web-based questionnaire directly to 139 IT professionals from software development companies located in the Greater Poland district in Poland and posted an invitation to participate in the survey on the LinkedIn page of Poznan Agile User Group[??]. In response, we received 33 filled questionnaires.

To characterize respondents, we can say that they had worked in IT projects for at least one year (1–17 years, median 6 years), and correspondingly in agile projects for at least half a year (0.5–14 years, median 2 years). They participated in 1–15 agile projects (median 5). Forty-five percent of respondents worked as Scrum Masters (15), forty-eight percent of them elicited or analyzed requirements (16), around sixty percent had experience in testing or quality assurance (20); and seventy percent played technical roles—programming or designing software (23).

Nearly all of the respondents correctly denied the existence of *fake* relationships (there were 2–3 confirmations of each fake relationship). Additionally, all but one relationship were confirmed by at least 76% (25) of respondents. Only one relationship was perceived as questionable (45% (15) approved and 21% (7) denied the existence of this relation, the others were not able to decide).

The descriptions of the identified influence relationships between practices, and the justification of their inclusion to REMMA are presented in tables 1 and 2. We tried, to the best of our ability, not to repeat information provided in Appendix A). While discussing the identified relationships in the tables, we will refer to the results of this survey using the following pattern (Yes%/No%/Do not know or hard to say%).

Table 1: Negative influence relationships between practices in REMMA (the *neutral level* column is used by the assessment method).

Affecting practice	Neutral level	Type	Affected practice
P1.1: Available / On-site customer available	normal use	–	P5.2: Write short, negotiable requirements
Justification: (91/0/9) Agile processes advocate for minimal documentation [57, 58, 59, 60, 10] in the form of short, negotiable requirements, such as user stories. A visible shift is made from documentation to communication [58, 61]. Cohn [57] calls this type of lightweight requirements a *two-way promise* between developers and customer. Both sides promise that they will be available and willing to discuss the details when the time comes. If the customer is not available, it means that the agreement is broken and the team's chances to succeed diminish.			
P2.3: Define project / product constraints	normal use	–	P5.5: Make requirements estimable
Justification: (94/3/3) The most frequently used effort estimation methods in agile projects are the expert-judgment methods [62, 63]. It is assumed that requirements are provided in such a form that developers can understand their scope, estimate effort, and make commitments. Project and product constraints can visibly influence the effort needed to implement a requirement [64, 65]. If they are not defined and communicated to developers, they will not have a complete understanding of the requirements they affect. Therefore, such requirements are not fully estimable. In addition, Beck [7] states that defining constraints make requirements clearer in respect to the cost and intended use of the product.			
P5.10: Assess implementation risks for requirements	normal use	–	P5.5: Make requirements estimable
Justification: (76/9/15) Similarly to the constraints, being unaware of risks related to implementing a requirement increases the level of uncertainty and makes accurate estimation more difficult. Drury-Grogan et al. [66] reported that some developers feel uncomfortable waiting for design decisions to emerge during iteration execution, hence they prefer to discuss technical issues during iteration planning. Risks are valuable in the context of how estimates are used for negotiating the scope of the release. Being able to incorporate risks into estimates creates a beneficial counterpoint to prioritizing only by the means of business value [57, 67].			
P4.5: Cover requirements with acceptance tests	normal use	–	P4.2: Prepare and maintain automatic acceptance tests
Justification: (85/6/9) Acceptance tests cannot be automated if they do not exist.			
P5.1: Make requirements independent	normal use	–	P5.5: Make requirements estimable
Justification: (82/9/9) Agile approaches to planning iterations assume that the customer and developers collaboratively set the scope of iteration. The customer prioritizes requirements while developers provide the estimates. As a result, the customer can flexibly select the most valuable features to be implemented first. However, if requirements depended on each other, it makes estimating a cumbersome task [59, 68]. It becomes especially difficult when the dependent requirements are not meant to be included within the scope of the iteration [53, 69].			
P5.3: Make complex requirements divisible	de facto standard	–	P5.1: Make requirements independent
Justification: (94/3/3) Decomposition of requirements is one of strategies to make requirements independent [68] (see Appendix A). Therefore, if complex requirements are defined in such a way that they are not ready to be decomposed, then splitting them into a set of independent requirements might be difficult.			
P5.4: Requirements should be valuable to purchasers or users	normal use	–	P5.8: Let customer prioritize requirements
Justification: (94/3/3) If the customer is not able to understand and assess the business value of a requirement, he or she will have problems deciding on how important they are for the success of the project or product and when the requirements should be implemented [70, 59, 68].			
P5.5: Make requirements estimable	de facto standard	–	P6.1: Negotiate release scope with customer
Justification: (88/9/3) Agile methods advocate for achieving a mutual agreement on iteration scope between the customer and developers. It is assumed that developers should commit themselves to implementing as many of the most important requirements as they believe is possible. However, if requirements are non-estimable, the level of uncertainty increases and the team may make an unrealistic commitment. It is frequently observed that developers underestimate effort in agile projects [71, 72].			
P5.8: Let customer prioritize requirements	normal use	–	P6.1: Negotiate release scope with customer
Justification: (88/6/6) Complementary to the previously discussed relationship, the customers' role during iteration planning is to optimize the business value of the requirements included within the scope of iteration. Therefore, if requirement priorities do not reflect customers' point of view, they would not be able to effectively negotiate the scope of iteration.			
P5.6: Make requirements testable	normal use	–	P4.5: Cover requirements with acceptance tests
Justification: (94/0/6) If it is impossible to define a reasonable set of acceptance criteria, it is also impossible to create acceptance test scenarios. The problem of non-testable requirements mainly relates to non-functional requirements [59].			
P5.6: Make requirements testable	de facto standard	–	P4.3: Prepare acceptance tests before coding
Justification: (91/0/9) The justification is similar to the previous relationship. Non-testable requirements make it more difficult or even impossible to create acceptance tests before coding.			
P5.9: Define requirements using notation and language that can easily be understood by all stakeholders	normal use	–	P5.4: Requirements should be valuable to purchasers or users
Justification: (85/9/6) The agile community emphasizes the importance of communication. The communication is ineffective if one side does not understand the language the other side uses. It is the same with the notation used to express requirements. If customers are not able to read and understand requirements, they will not be able to assess their business value.			
P5.9: Define requirements using notation and language that can easily be understood by all stakeholders	disc. use	–	P5.5: Make requirements estimable
Justification: (85/9/6) There are different reasons for calling a requirement non-estimable. The main issue relates to insufficient understanding of the requirement and the domain it belongs to [70, 59, 68, 73]. Therefore, if developers cannot understand language or notation used to express requirements, it will create a barrier to understanding the requirements.			

Table 2: Positive and positive-negative influence relationships between practices in REMMA (the *neutral level* column is used for assessment).

Affecting practice	Neutral level	Type	Affected practice
P4.2: Prepare and maintain automatic acceptance tests	normal use	±	P4.4: Perform regression acceptance testing
Justification: ([-] 76/15/9; [+] 85/9/6) Similarly to the constraints, being unaware of risks related to implementing a requirement increases the level of uncertainty and makes accurate estimation more difficult. Drury-Grogan et al. [66] reported that some developers feel uncomfortable waiting for design decisions to emerge during iteration execution, hence they prefer to discuss technical issues during iteration planning. Risks are valuable in the context of how estimates are used for negotiating the scope of the release. Being able to incorporate risks into estimates creates a beneficial counterpoint to prioritizing only by the means of business value [57, 67].			
P4.5: Cover requirements with acceptance tests	normal use	±	P5.2: Write short, negotiable requirements
Justification: ([-] 87/3/10; [+] 88/0/12) Acceptance tests cannot be automated if they do not exist.			
P6.3: Keep iteration length short to continually collect feedback	normal use	+	P5.2: Write short, negotiable requirements
Justification: (45/21/33) By relying on short, negotiable requirements, such as user stories, a customer can change the direction of product development. The short iterations provide valuable input for making such changes, and support what is called *iterative requirements* [57, 10]. In addition, a short feedback loop is beneficial for developers who are sometimes left by their customers with unclear requirements [74]. In such cases, short iteration cycles give them a chance to receive frequent feedback and clarification of the unclear requirements. The relationship was confirmed by only 48 percent of the survey respondents. Although, only 29 percent of the respondents disagreed with the existence of the relationship, it is a signal that the influence might be in question. This is also the only positive relationship. It might also be the case that it is easier to confirm the existence of negative relationships than positive ones.			

4.3. Context factors in REMMA

As we have already stated in Section 3, practices in agile projects should be aligned with the project and organizational contexts. The REMMA model addresses this issue by introducing the concept of *context factor*.

Definition 6. *A **context factor** describes a unique property of the environment, project or product that could have an effect on how the team operates. The intensity of the context factor is expressed by its **values**. Each context factor has at least two values, one indicating its presence in a project, and the other denying it.*

For instance, the *functional size* of a product could be considered a potential context factor in many projects. It might have three values such as *low*, *medium*, and *high*. Other example of context factor could be *adherence to Scrum Framework*, with two values, *yes* and *no*.

In an agile project, the project team responds to a context factor by implementing a set of (agile or non-agile) practices. If the practices are selected and executed properly, we say that they are correctly aligned with the context.

Therefore, a certain value of a context factor might require incorporating specific practices into the process. To address this issue, the REMMA model introduces the concept of *response relationship* between a practice and value of context factor.

Definition 7. *The existence of a **response relationship** between the practice P and the context-factor value CF_{val} ($P \xrightarrow{r} CF_{val}$) implies that (insufficient) implementation of the practice P can have influence on the project affected by the context-factor value CF_{val}. The practice P is called **response practice**.* We distinguish three types of response relationships:

- *positive* ($P \xrightarrow{r+} CF_{val}$) — the proper implementation of the response practice P can have a positive influence on the project affected by the context-factor value CF_{val};
- *negative* ($P \xrightarrow{r-} CF_{val}$) — the insufficient use of the response practice P can have a negative influence on the project affected by the context-factor value CF_{val};
- *positive and negative* ($P \xrightarrow{r\pm} CF_{val}$) — the response practice P can have both positive and negative influence on the project, accordingly to how well the response practice P is implemented.

In addition, a response relationship can have an *inverted* effect ($P \xrightarrow{\neg r} CF_{val}$). For instance, an inverted positive response relationship between the practice P and value of context factor CF_{val} ($P \xrightarrow{\neg r+} CF_{val}$) would imply that the proper implementation of the practice P can have a negative influence on the project if the context-factor value CF_{val} applies to the project.

As we already stated, the role of context factors is to enable an organization or agile community to model lessons learned regarding how to effectively operate in a given environment. Therefore, in some cases an organization might effectively incorporate practices that are not included in the REMMA catalog. In extreme cases, these practices might even be perceived as not being agile. For instance, if the goal of a project is to develop a safety-critical system, there is a justified need for preparing more comprehensive documentation [17]. As a result, in REMMA, context-factor values can *introduce practices* and *influence relationships between practices* into the model.

We identified three rationales for introducing new practices or influence relationships between practices:

- to use the practice coupled with other agile RE practices in order to address context-specific issues;
- to *compensate* the lack / insufficient implementation of other practices;
- to *substitute* for other practices that should not be used in a given context. Substitution means that a different practice or influence relationship should be used instead of the substituted one. Substitution may also be used if one would like to redefine an existing practice so that it is better suited to the context (e.g., introduce more restrictive minimal quality requisites).

Examples of how to model a) compensations and b) substitutions in REMMA are presented in Figure 2. Compensation between the practice P_X ❷, introduced by the context factor CF_{val} ❶, and the practice P_1 is modeled using a standard influence relationship ❸. A substitution between the practices is modeled using the *substitution relationship*.

Definition 8. *A **substitution relationship** between the practices A and B ($A \xrightarrow{sub|CF_{val}} B$) is defined for a context-factor value CF_{val}. It implies that the practice A should be used instead of the practice B if the context-factor value CF_{val} appears in the project. As a result, it implicitly triggers the existence of an inverted, positive response relationship between the practice B and CF_{val}:*
($A \xrightarrow{sub|CF_{val}} B \Rightarrow B \xrightarrow{\neg r+} CF_{val}$).

Similarly, a substitution relationship can be defined for two influence relationships having the same affected practice (e.g., $A \xrightarrow{i} C$ and $B \xrightarrow{i} C$). Again, it is defined in the context of a certain context-factor value. It also preserves a similar meaning: the substituting relationship replaces the substituted one.

A context factor in REMMA can describe a large variety of situations. Some of the factors can be specific to a given project, and thus it would be difficult to generalize them to other cases (e.g., responses to the need for cooperating with a certain customer, enhancing a certain product). These project-specific context factors should be modeled based on the knowledge and experience of particular organizations and project teams. Conversely, there are some very general and commonly agreed upon context factors, such as product size, distributed team, business domain, etc. The responses to these factors are more likely to be generalizable between different projects.

Therefore, taking into account a large number of possible context factors, it would be very difficult to propose a complete model of responses to the factors at hand. For instance, Kruchten [18] concluded that, in order to model

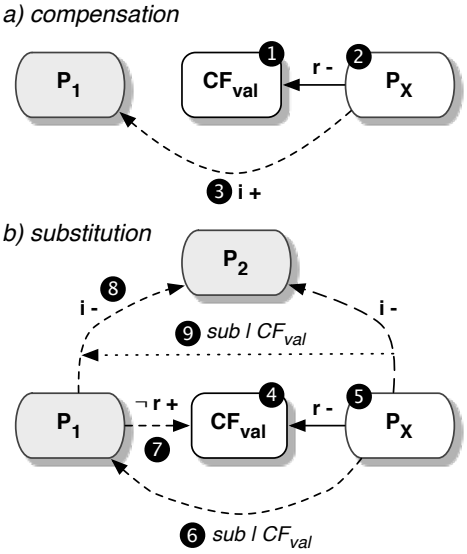

Figure 2: Examples of modeling a) compensation and b) substitution of practices and influence relationships in REMMA (a value of context factor ❶, ❹; a practice introduced by the context factor CF_{val} ❷, ❺; an influence relationship showing that the practice P_X can compensate the implementation of practice P_1 ❸; a substitution relationship between the practices P_X and P_1 ❼ with the implicitly implied inverted response relationship showing that the practice P_1 should not be used in the context CF_{val} ❻; an influence relationship between the practices P_1 and P_2 ❽, which is substituted by a relationship between P_X and P_2 ❾).

all relationships between 8 context factors that he proposed and most popular agile practices, he would have to investigate over a thousand cases. Taking that into consideration, we propose a tool that could be used to express such relationships, rather than trying to propose a complete set of context factors and adequate responses to them. However, we decided to illustrate how specific context factors can be added to REMMA by modeling four selected context factors.

We decided to select factors that seem to be very general and applicable to many projects. The first two factors model the need for adherence to recommendations of two agile software development methods: XP and Scrum. The remaining two factors are distributed teams and safety-critical systems. The two final factors represent some extreme cases for the possibility of adopting the agile methods. That is why they are sometimes given as arguments for discussing the limitations of agile methods [75].

XP and Scrum Modeling adherence to software development methods was based on the analysis of documents describing the methods. In the case of Scrum, we based the model mainly on the *Scrum Guide*™ [8], which defines the framework's core elements: roles, events, artifacts, and rules. For XP, we could not find a singular relevant and up-to-date reference point. Therefore, we decided to base the model mainly on Beck's book *Extreme Programming Explained: Embrace Change* [7] and the book by Stephens and Rosenberg [53] discussing weaknesses and strong points of XP.

The context factors describing adherence to XP and Scrum are presented in tables 3 and 4. Our strategy was to first identify practices directly proposed by the methods and make them obligatory (use negative response relationship). Then, we analyzed which REMMA practices could support implementing the methods, and based on that we added additional response relationships to the models. Finally, we identified the practices related to RE which are distinct to these methods and are not included in the REMMA catalog. As a result, we added three additional practices to the context factor related to Scrum: *product backlog*, *sprint backlog*, and *decisive product owner*. The practices are briefly discussed in Appendix A.1.

The Global Software Development (GSD) is probably one of the most frequently investigated context factors when it comes to agile software development methods. The reason for that is that agile methods focus on direct face-to-face communication, which is much more difficult when a project team is physically distributed.

In order to construct a context factor for GSD, we decided to conduct a literature study to find out how agile teams respond to problems stemming from physical distribution. We based our study on two systematic literature studies—the first by Hossain et al. [54] and the second by Jalali and Wohlin [77]. In addition, we analyzed the papers referred to in those studies to extract more specific insights. The resulting model of the context factor is presented in Table 5.

From the literature study, it seems that there are three major areas that are challenging for GSD, agile projects: communication, knowledge management, and trust. The main response to these challenges is to focus on maximizing the opportunities to communicate with members of distributed teams. On these grounds, it is frequently advised to promote different types of meetings, such as daily team meetings, iteration reviews, and retrospectives. The last one in particular helps to build trust among the project's members. However, organizing such meetings might be challenging, especially when sub-teams operate in different time zones. Thus, it is proposed that working hours be synchronized (or at least overlapping hours are found between different sites). When it comes to communication and knowledge sharing, it is important to introduce proper tools for distant communications and knowledge management, such as e-mails, video conferences, wikis, etc. Multiple communication modes should also be introduced in order to increase communication bandwidth. The role of effective communication with customers is crucial as well. Finally, it is recommended that frequent visits of members among different sites are organized to help to build stronger relationships between project members and reinforce the feeling of unity.

In the proposed context model of the agile GSD, practices related to knowledge sharing and communication were made obligatory. In addition, three new practices were introduced: *frequent visits*, *synchronized work hours*, and *multiple communication modes*. These practices are briefly described in Appendix A.1.

Safety-critical systems We decided to model *safety-critical systems* as another example of an extreme context factor in agile software development. In fact, safety-critical systems are often given as an example of products that might not fit well into the agile software development methods [75, 80, 81].

We once again decided to base the model on a literature study. Unfortunately, we have not found relevant secondary studies. Therefore, we decided to perform a limited literature review using common search engines (Google Scholar, IEEE eXplore, ACM Digital Library) and the backward snowballing technique. While performing the search, we were especially interested in empirical studies and studies that investigated the conformance of agile practices with standards for developing safety-critical systems. The results of the literature study did not provide us with sufficient evidence to allow us to model more than two values (*yes* and *no*) of the context factor (e.g., different types of safety-critical systems).

As a result of the study, we identified two main concerns related to agile RE in the context of developing safety-critical systems. These are the minimalistic approach to requirements documentation and a lack of an up-front analysis of requirements. No up-front analysis constitutes a problem because the design of a safety-critical system should be subjected to safety analysis before the system is implemented [82, 83, 84], e.g., Safety Impact Analysis, Functional Failure Analysis (FFA), Hazards and operability analysis (HAZOP). An added problem is that whenever a request for change appears, its impact on safety has to be investigated. Hence, allowing for emerging requirements might increase the costs of software development.

There are two solutions proposed to these problems. The first is to prepare in advance a (sufficiently) comprehensive requirements specification and architecture design that will allow performing necessary safety analyses and enable high-level planning [82, 85]. The second is to prepare descriptions of requirements that are more detailed than typical user stories during iterations (e.g., create use-case models) [55, 86].

Another issue concerns the feasibility of short iteration cycles and emerging requirements. Recently published papers provide some empirical evidence suggesting that short, consistent iteration cycles are beneficial for safety-critical projects. For instance, VanderLeest and Buter [87] reported that they were able to implement fixed weekly iteration cycles (avionic sector, system compliant with DO-178B). They also reported that a fixed length of iterations "*adds consistency to the planning as well as helping to prevent 'feature creep' because once an iteration's tasks have been set, they should not be changed.*" Similarly, Trimble [88], while sharing his experience from developing The Mission Control Technologies (MCT) in NASA, emphasizes the need for regular deliveries of working software (even without incomplete requirements). In addition, he

Table 3: Context factor: C1. eXtreme Programming (the column *neutral level* is used by the assessment method).

Response practices	Type	Neutral level	Justification
P4.1: Let customer define acceptance tests	–	de facto standard	Test-First Programming [53, 7]
P4.2: Prepare and maintain automatic acceptance tests	–	de facto standard	Test-First Programming [53, 7]
P4.3: Prepare acceptance tests before coding	–	de facto standard	Test-First Programming [53, 7]
P4.5: Cover requirements with acceptance tests	–	de facto standard	Test-First Programming [53, 7]
P5.6: Make requirements testable	–	de facto standard	Test-First Programming [53, 7]
P3.4: Provide and maintain informative workspace	–	de facto standard	Informative Workspace [7]
P3.5: Provide easy access to requirements	–	de facto standard	Sit together, Informative Workspace [7]
P5.2: Write short, negotiable requirements	–	de facto standard	Stories [7]
P5.4: Requirements should be valuable to purchasers or users	–	de facto standard	Stories [7]
P5.5: Make requirements estimable	–	de facto standard	Stories, Planning Game [7]
P6.4: Define a fixed iteration length	–	de facto standard	Small releases, cycles, Sustainable pace [53, 7]
P5.8: Let customer prioritize requirements	–	de facto standard	The Planning Game [53, 7]
P6.1: Negotiate iteration scope with customer	–	de facto standard	The Planning Game [53, 7]
P6.3: Keep iteration length short to continually collect feedback	–	de facto standard	Small releases, cycles [53, 7]
P5.9: Define requirements using notation and language that can easily be understood by all stakeholders	–	de facto standard	Stories, Informative Workspace [7]
P5.1: Make requirements independent	±	normal use	Stories, The Planning Game [7]
P5.3: Make complex requirements divisible	±	normal use	Stories, The Planning Game [7]
P3.1: Organize everyday team meetings	±	normal use	Stand-up meetings, Sustainable pace [7, 76]

Table 4: Context factor: C2. Scrum (the column *neutral level* is used by the assessment method).

Response practices	Type	Neutral level	Justification
P3.1: Organize everyday team meetings	–	de facto standard	Daily Scrum [8]
P3.2: Organize review meetings	–	de facto standard	Sprint Review [8]
P3.3: Organize retrospective meetings	–	de facto standard	Sprint Retrospective [8]
P6.1: Negotiate iteration scope with customer	–	de facto standard	Sprint Planning [8]
P6.2: Avoid changing increment scope after it is agreed upon	–	de facto standard	Sprint [8]
P6.3: Keep iteration length short to continually collect feedback	–	de facto standard	Sprint [8]
P5.8: Let customer prioritize requirements	–	de facto standard	Product Owner [8]
P5.4: Requirements should be valuable to purchasers or users	–	de facto standard	Product Owner, Sprint Planning [8]
P3.5: Provide easy access to requirements	–	de facto standard	Product Owner [8]
P5.9: Define requirements using notation and language that can easily be understood by all stakeholders	–	de facto standard	Product Owner [8]
P5.5: Make requirements estimable	–	de facto standard	Product Owner [8]
P5.6: Make requirements testable	–	de facto standard	Product Owner, understanding of "done" [8]
P4.1: Let customer define acceptance tests	–	de facto standard	Product Owner, understanding of "done" [8]
P4.5: Cover requirements with acceptance tests	–	de facto standard	Product Owner, understanding of "done" [8]
C2.P1: Product Backlog	–	de facto standard	Product Backlog [8]
C2.P2: Sprint Backlog	–	de facto standard	Sprint Backlog [8]
C2.P3: Decisive Product Owner	–	de facto standard	Product Owner [8]
P6.4: Define a fixed iteration length	±	normal use	Sprint [70]
P3.4: Provide and maintain informative workspace	±	normal use	Product Owner, Product Backlog, Spring Backlog, Sprint [70, 8]
P1.1: Available / On-site customer	±	normal use	Product Owner [70]
P1.2: Involve different stakeholders	±	normal use	Product Owner [70]
P5.2: Write short, negotiable requirements	±	normal use	Product Backlog, User stories [70]
P5.1: Make requirements independent	±	normal use	Product Backlog, User stories [70]
P5.3: Make complex requirements divisible	±	normal use	Product Backlog, User stories [70]

Table 5: Context factor: C3. The Global Software Development (the column *neutral level* is used by the assessment method).

Response practices	Type	Neutral level	Justification
P1.1: Available / On-site customer	±	de facto standard	Communication, Knowledge sharing, Trust [54, 77]
P3.1: Organize everyday team meetings	−	de facto standard	Communication, Knowledge sharing, Trust [54, 77]
P3.2: Organize review meetings	−	de facto standard	Communication, Knowledge sharing, Trust [54, 77]
P3.3: Organize retrospective meetings	−	de facto standard	Communication, Knowledge sharing, Trust [54, 77]
P3.4: Provide and maintain informative workspace	−	de facto standard	Communication, Knowledge sharing [54]
P3.5: Provide easy access to requirements	−	de facto standard	Communication, Knowledge sharing [54]
P5.9: Define requirements using notation and language that can easily be understood by all stakeholders	−	de facto standard	Communication [78]
P6.3: Keep iteration length short to continually collect feedback	±	normal use	Communication [79, 77]
C3.P1: Frequent visits	+	never used	Communication, Knowledge sharing, Trust [54]
C3.P2: Synchronized work hours	−	de facto standard	Communication, Knowledge sharing [54]
C3.P3: Multiple communication modes	±	discretionary used	Communication, Knowledge sharing [54]

states that although only verified safety-critical software can be deployed in an operational environment, it does not prevent the development team from providing increments to the customer in order to receive feedback. A similar view on agile, iterative development was presented by Gary et al. [89]. They emphasize the value of frequently delivering working software in the context of developing safety-critical systems. They additionally reported a successful implementation of the approach to handling emerging requirements, using continuous verification and prioritization of the product backlog.

When it comes to customer involvement, agile projects developing safety-critical systems seem to be by no means different from other agile projects. For instance, VanderLeest and Buter [87] reported that they were able to achieve customer involvement on both a daily basis and on iteration boundaries. Moreover, they emphasize the importance of consistent customer involvement, including at subcontracting levels. The possibility of having an on-site customer in safety-critical projects was also confirmed by Jonsson et al. [86]. Bowers [90] emphasizes that the customer's role is crucial in agile projects developing safety-critical systems.

The two further findings that we made during the study were unanimously presented in most of the analyzed papers. The first relates to the necessity of preserving traceability between code, requirements and test cases, which is imposed by safety standards [86, 89, 85]. The second one relates to the importance that agile methods place on testing (testable requirements, automated tests, regression tests, acceptance tests prepared by customers) [75, 86, 87, 89, 90, 91].

Based on the presented findings, we proposed three new practices for the context factor: *prepare an up front requirements specification that is sufficient for safety analysis*, *traceability*, and *comprehensive, negotiable requirements*. The last practice substitutes *writing short, negotiable requirements*. In addition, the practices related to customer involvement, testing, and iterative development were added to the model, as they seem to be especially beneficial for the projects affected by this context factor. The resulting model of the context factor is presented in Table 6.

The presented four context factors illustrate the use of the context-factors component of the REMMA method. In the next section, we are going to explain how they are used to assess practice alignment with the context.

5. Assessment of practice alignment

The assessment method in REMMA enables the appraisal of practice alignment on three levels:

- *Basic assessment* — the alignment of a project's practices with the agile RE practices is assessed based on the *frequency* of their *appropriate* usage.
- *Influence assessment* — the assessment of synergy between practices is added to the basic assessment.
- *Contextual assessment* — influence assessment is augmented with the evaluation of how appropriate the implementation of practices is in the project with respect to its environment.

The assessment is always performed for an agreed upon period, which is called *the time span of assessment*.

Definition 9. *The **time span of assessment** is a period for which the assessment is performed. For practical reasons, the time span of assessment should not be shorter than the duration of a single iteration in a project, because the usage of some of the practices, by definition, can be seen only once per iteration (e.g., an iteration review meeting).*

The assessment is performed by a person called the *assessor*. The assessor handles collecting evidence regarding the usage of practices, together with applying the rules of

Table 6: Context factor: C4. Safety-critical system (the column *neutral level* is used by the assessment method).

Response practices	Type	Neutral level	Justification
C4.P1: Prepare an up front requirements specification that is sufficient for safety analysis	−	de facto standard	[82, 85]
C4.P2: Write comprehensive, negotiable requirements (substitutes P5.2)	−	de facto standard	[55, 86].
C4.P3: Traceability	−	de facto standard	[86, 89, 85].
P1.1: Available / On-site customer	−	de facto standard	[87, 86, 90]
P4.1: Let customer define acceptance tests	±	normal use	[91]
P4.2: Prepare and maintain automatic acceptance tests	−	de facto standard	[86, 89, 90, 91]
P4.3: Prepare acceptance tests before coding	−	de facto standard	[75, 86, 87, 91]
P4.5: Cover requirements with acceptance tests	−	de facto standard	[75, 86, 87, 89, 90, 91]
P5.6: Make requirements testable	−	de facto standard	[86, 87]
P6.3: Keep iteration length short to continually collect feedback	±	normal use	[87, 88, 89]
P6.4: Define a fixed iteration length	±	normal use	[87]

the assessment method to produce and interpret its outcomes. Steps of the assessment can be performed manually or with the use of our prototype software tool, whose goal is to make the application of REMMA assessment rules less time consuming.

5.1. Basic assessment

The goal of basic assessment is to assess the basic alignment of practices, which according to Definition 3 is the degree to which the practices in a software development project make it possible to reach the same objectives as the agile practices defined in the agile maturity model (the agile RE practices defined in the REMMA catalog of practices).

According to the Definition 1, each practice defines its objectives, whose achievement can be examined by investigating the *output* produced while using the practice. For instance, if we consider the practice whose goal is to make the acceptance tests automatic, then the expected output of using the practice are automated acceptance test cases. Therefore, to assess the practice we have to verify if such test cases were created. Considering that, we make the following assumption:

Assumption 3. *The presence of the practice's expected output in a project is the evidence that the objectives of a practice were achieved.*

As presented above, the usage of a practice often results in producing certain artifacts (e.g., acceptance test cases). However, it is worth emphasizing that the output can have other forms as well, e.g., it could be an event or achieving a certain object state (e.g., the answer to a question related to requirements, or the fact that stakeholders understand the vision of a project).

Therefore, it is necessary to define the smallest piece of evidence that allows verifying whether a single act of using the practice resulted in producing its expected output. To address this issue, we introduced the concept of the practice *assessment unit*.

Table 7: The assessment units of practices described in the paper.

Practice	Assessment unit	Practice	Assessment unit
P1.1	request for requirements clarification	P5.4	requirement
P1.2	∅	P5.5	requirement
P2.1	∅	P5.6	requirement
P2.2	functional requirement	P5.7	∅
P2.3	∅	P5.8	requirement
P3.1	each day	P5.9	requirement
P3.2	iteration	P5.10	requirement
P3.3	iteration	P6.1	iteration
P3.4	∅	P6.2	iteration
P3.5	∅	P6.3	iteration
P3.6	∅	P6.4	∅
P3.7	∅	C2.P1	sprint
P4.1	acceptance criterion	C2.P2	sprint
P4.2	acceptance test	C2.P3	∅
P4.3	requirement	C3.P1	∅
P4.4	requirement implemented	C3.P2	∅
P4.5	requirement	C3.P3	∅
P5.1	requirement	C4.P1	∅
P5.2	requirement	C4.P2	requirement
P5.3	requirement	C4.P3	requirement

Definition 10. *The **assessment unit** of a practice is an atomic piece of evidence that allows determining if a single act of using the practice resulted in its expected output. A null assessment unit (∅) is used for practices that must always be assessed in the scope of a whole project (e.g., they refer to the state of a project).*

The assessment units of practices described in the paper are presented in Table 7.

Taking the above into consideration, in order to perform the basic assessment, the assessor has to examine the available instances of the assessment units of each practice to determine how often (*frequency*) its expected outputs

were produced (*quality*). The assessor does this by verifying whether *all* of the minimal quality requisites defined for the practice have been met by the *instances* of the assessment unit being investigated. For example, the assessment unit of the practice *write short, negotiable requirements* (P5.2) is a requirement. Therefore, each requirement in the project that meets all minimal quality requisites of the practice P5.2 provides a single piece of evidence of the practice's correct usage. Conversely, each requirement that does not meet all the requisites is evidence against the correct usage of that practice.

In the case of practices with null assessment units ($\emptyset$), the assessment is always performed for the whole project. For instance, for the practice *establish project's shared vision* (P2.1), the assessor needs to verify whether the project's vision is established, documented, contains up-to-date information, and is understood by all stakeholders.

Considering that the time span of assessment should cover at least the duration of a single iteration in a project (see Definition 9), the assessor has to examine how the usage of a practice changed within that period (i.e., assess the frequency of using the practice that resulted in its expected outputs).

An example of how the assessor might combine the assessment of frequency and quality into a single, combined assessment of practice is presented in Figure 3. The practice considered in sub-figure a) has a null unit of assessment. As it is presented, the minimal quality criteria of the practice were met for one and a half out of two iterations covered by the time span of assessment. This allows us to state that the practice was appropriately used 75% of the time. The practice considered in sub-figure b) has a specific unit of assessment; as an example, let us assume it is acceptance test. The number of instances of the assessment unit (here, the number of acceptance test cases) that met the minimal quality criteria changed over time, from 40% during the first half of the first iteration to 80% for the remaining period of time. When averaged over time, this gives the final assessment of 70%.

To conclude this part, we could state that in order to perform the basic assessment, the assessor has to collect evidence of how often and how well each practice was used within the agreed time span of assessment. The evidence can be collected either by analyzing project data (e.g., for the practice P1.1 *available / on-site customer*, the assessor could analyze how the requests for requirements clarification stored in the task management system were handled) or by interviewing project team members. In the latter case, interviewees are asked to reflect on how a practice was used within a given period of time (e.g., how often was the project vision kept up to date within the last two increments, or what was the average percentage of requirements covered with acceptance test cases within the last two iterations).

Assuming that the main method of collecting data would be to less or more formally interview project members, we decided to introduce a fuzzy interval scale of assessment in-

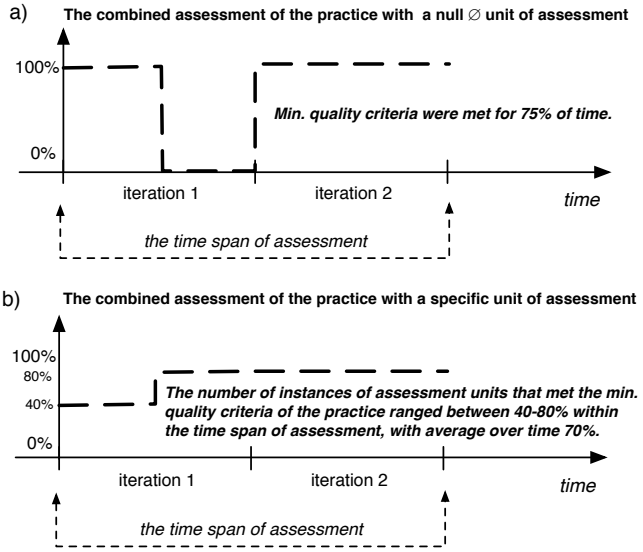

Figure 3: An example of assessing the usage of practices in time: practices with binary a) and qualitative b) types of assessment results.

stead of forcing respondents to provide percentage values. We believe that this makes the assessment faster and prevents situations where interviewees are confused because they do not feel confident enough to provide answers with the demanded level of precision (in this case, we value accuracy over precision).

The resulting four-point assessment scale is similar to the one proposed by Sommerville and Sawyer [21]:

- *De facto standard* (level=3): a practice is commonly agreed upon in a project and used *appropriately* for no less than 75% of time.
- *Normal use* (level=2): a practice is commonly agreed upon in a project and is used *appropriately* for no less than 50% of time.
- *Discretionary use* (level=1): a practice is not commonly agreed upon and is occasionally used *appropriately* in a project (for no less than 25% of time).
- *Never used* (level=0): a practice is hardly ever used *appropriately* in a project (for less than 25% of time).

Depending on the information needs, the results of the basic assessment can be reported in various forms. However, after Sommerville and Ranson [45], we propose presenting the coverage of the practices visually relative to their importance. A similar analysis can be performed for each of the RE areas. An example of analysis results is presented in Figure 4. It indicates that most of the critical practices were frequently used (80% of critical practices are at least normally used). However, it also reveals that all of the neglected critical practices belong to the area of release planning.

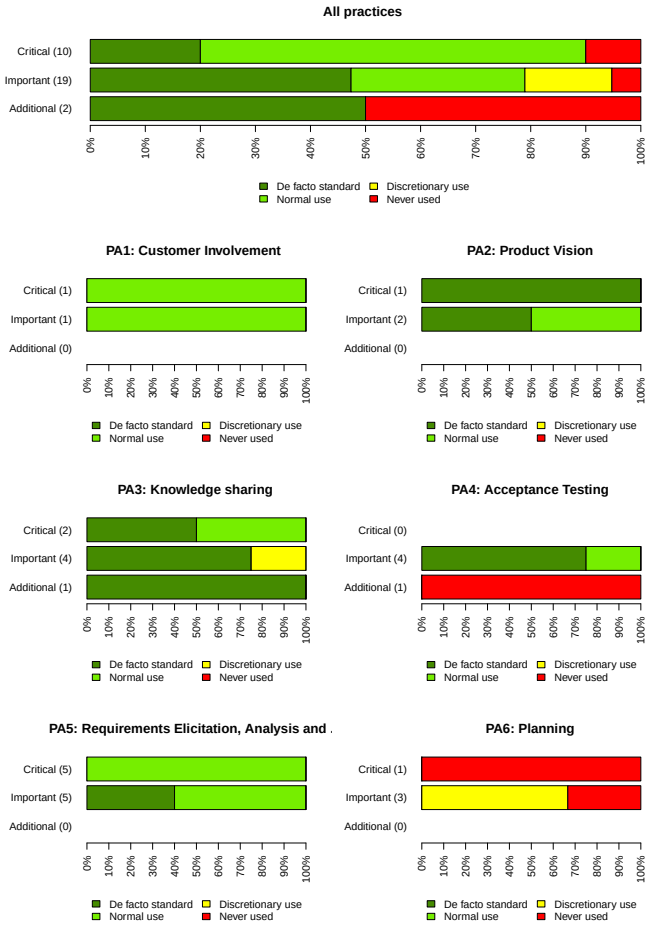

Figure 4: A chart presenting exemplary results of basic assessment.

In addition to providing visual presentation of the assessment results, we propose calculating two simple measures: *percentage of practice implementation* (PIP), and *total number of levels* (TL) according to equations 1 and 2. The TL measure represents the total number of assessment levels obtained for all practices. It therefore depends on the number of practices considered in the assessment. Both measures can be calculated for the entire catalog of practices, the practices belonging to a certain area, or the practices of a certain level of importance.

$$TL = \sum_{i=1}^{n} Assessment(Practice_i) \tag{1}$$

$$PIP = \frac{TL}{3 \times n} \times 100\% \tag{2}$$

Where:

- n is the number of practices in the catalog (in the case of contextual assessment discussed in Section 5.3, the default catalog is extended by practices introduced by the values of context factors);
- $Assessment(Practice_i)$ gives the number of implementation levels for the practice $Practice_i$ obtained during the assessment {never used = 0; discretionary use = 1; normal use = 2; de facto standard = 3}.

5.2. Influence assessment

The goal of influence assessment is to evaluate the degree to which synergy between the practices in a project is preserved. It is performed on top of basic assessment results.

Before we proceed to describe the procedure of performing the influence assessment, let us first introduce the concept of the *neutral level* of a relationship.

Definition 11. ***Neutral level*** *is defined for an influence and response relationship. It indicates the point at which the affecting practice does not have any meaningful influence on the affected practice or context-factor value. Depending on the relationship type (positive, negative, positive-negative), if the assessed level is greater or lesser than the neutral level, the relationship becomes meaningful for the project and contributes to the results of influence or contextual assessment.*

The assessors perform the influence assessment according to the steps of Algorithm 1. The algorithm consists of three stages. The goal of the first stage is to initialize variables (lines 1–5). In the second stage, all influence relationships are investigated (lines 6–14). Firstly, the influence level of a relationship is calculated by comparing the basic assessment level of the affecting practice and the neutral level defined for the relationship. Then, in the line 10, it is checked whether the relationship is triggered, based on the previously calculated influence level and the type of relationship. If the relationship is triggered, the influence is stored as the number of minus or plus signs equal to the influence level of relationship. The goal of the third stage of the algorithm (lines 15–23) is to calculate the results of the assessment for each practice as the sum of all influence levels and the basic assessment. Additionally, the information about the triggered relationship is stored to enable the traceability of relationships contributing to the influence assessment and the practice.

An example of analyzing the results of an influence assessment from the perspective of a single practice is presented in Figure 5. It demonstrates a project team claiming that they almost always correctly use (*de facto standard*) the practice *write short, negotiable requirements* (P5.2). From the perspective of basic assessment, this indicates the highest possible alignment with the state-of-the-art agile RE practice. Unfortunately, this assessment may not show the entire truth, as it is apparent that the team does not pay enough attention to getting the customer's representative involved in clarifying requirements, which limits the strength of practice P5.2. Fortunately, the team tries to mitigate the problem of low customer availability by incorporating other practices. They predominantly pay attention to defining acceptance tests for the requirements

Data:
P — a set of REMMA practices;
IR — a set of REMMA influence relationships;
$p.basic\_assessment$ — the result of basic assessment (number of levels) for the practice p;
$rel.neutral\_level$ — neutral level defined for the influence relationship rel;
Result:
$p.influence\_in$ — a set of tuples (lvl, rel) such that lvl is the level of influence another practice has on the practice p, triggered by the relationship rel;
$p.influence\_out$ — a set of tuples (lvl, rel) such that lvl is the level of influence the practice p has on some other practice, triggered by the relationship rel;
$p.total\_influence\_in$ — is the sum of all influence levels the other practices have on the practice p;
$p.total\_influence\_out$ — is the sum of all influence levels the practice p has on the other practices;
$p.influence\_assessment$ — is the influence assessment of the practice p including the basic assessment and the sum of all influence levels the other practices have on the practice p;

```
1  foreach p ∈ P do
2      p.influence_in ← {};
3      p.influence_out ← {};
4      p.total_influence_in ← 0;
5      p.total_influence_out ← 0;
6  foreach rel: a --i type--> b ∈ IR do
7      lvl ← a.basic_assessment − rel.neutral_level;
8      if (lvl < 0 and type ='−')
9        or (lvl > 0 and type ='+')
10       or type ='±' then
11         b.influence_in ←
12           b.influence_in ∪ {(lvl, rel)};
13         a.influence_out ←
14           a.influence_out ∪ {(lvl, rel)};
15 foreach p ∈ P do
16     foreach in ∈ p.influence_in do
17         p.total_influence_in ←
18           p.total_influence_in + in[0];
19     foreach out ∈ p.influence_out do
20         p.total_influence_out ←
21           p.total_influence_out + out[0];
22     p.influence_assessment ←
23       p.basic_assessment + p.total_influence_in;
```

Algorithm 1: Algorithm of conducting influence assessment for all practices.

(*de facto standard*). It might help to clarify imprecise requirements. Another way of dealing with a lack of on-site customer presence is to minimize the release length, so as to increase the chance of receiving feedback. However, it seems that the team did not seize this opportunity. To summarize, the presented analysis shows that the implementation of practice P5.2 is exposed to an important risk related to low customer availability, and thus the team's claim about the perfect usage of the practice seems overoptimistic.

If the results of influence assessment are interpreted in the form of a causal analysis as presented above, each of the triggered influence relationships provides some feedback about the alignment of practices. That is why we need to preserve traceability between the results (influence levels) and relationships.

Another way of interpreting the result is to calculate the aggregated influence assessment for a practice as it is presented in line 23 of Algorithm 1. The aggregated assessment result enables calculating the previously introduced TL and PIP measures at the level of influence assessment. For example, for the case presented in Figure 5, the final influence assessment of practice P5.2 would be equal to 3 (basic assessment) – 2 + 1.

Finally, the analysis of triggered relationships might help to find practices that we call *negative influencers*. These are those practices that are used insufficiently, and as a result have a negative influence on other practices. These practices can be found by analyzing the triggered relationships in which the practice appears as affecting practice, and the influence result is negative.

5.3. Contextual assessment

The goal of contextual assessment is to investigate whether practices are aligned with the context. As it was stated in Section 4.3, the context is described by a set of context factors. A context factor corresponds to a single feature of the environment, e.g., recommendations for a software development methodology, characteristics of the customer, application type, etc. It is assumed that the set of context factors should reflect the lessons learned by a team or organization. Therefore, it might be very specific to a given team or organization.

The first step of contextual assessment is to describe the environment by selecting appropriate context factor values. Once those values are selected, the assessor performs the contextual assessment according to the steps of Algorithm 2.

The algorithm requires, as the first step, that basic and influence assessments are performed for any new practices and influence relationships introduced by the context-factors model (lines 1–2). In the second step, the variables are initialized (lines 3–5 and line 7). In the third step, all context-factor values *characterizing project* are investigated by analyzing how the practices implemented in the project respond to the context-factor value (lines 6–19). This is done by calculating the response level of the relationship by comparing the level of basic assessment of the practice with the neutral level defined for the relationship. Then, in the line 13, the triggering conditions are checked for the relationship. If the relationship is triggered, the influence is stored as the number of minus or plus signs equal to response level of the relationship. It is worth mentioning that for inverted relationships the response level is multiplied by minus one. The goal of the

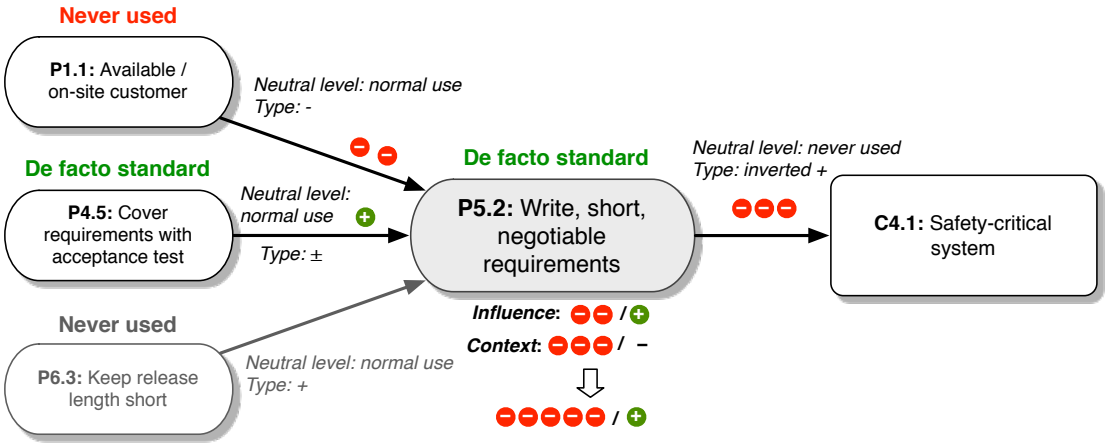

Figure 5: An example of influence and contextual assessments.

final stage of the algorithm (lines 20–26) is to calculate the results of the assessment for each practice as the sum of all response levels and its influence assessment. Additionally, information about triggered relationships is stored to enable the traceability of relationships contributing to the contextual assessment.

Let us now go back to the example presented in Figure 5 to illustrate how the results of contextual assessment enhance the results of influence assessment. Based on the results of influence assessment, we have already observed that the practice *write short, negotiable requirements* (P5.2) is claimed to be correctly used by the project team (*de facto standard*), however there is a visible threat to its usage in the form of a poorly cooperating customer representative. Let us now extend the assessment by including a context-factor value that appeared in the project, stating that the project aims to develop a safety-critical system.

The context factor value substituted practice P5.2 with the more restrictive practice *write comprehensive, negotiable requirements* (C4.P2). As a result, the inverted, positive response relationship introduced by the context-factor value was triggered and resulted in a negative contextual assessment of practice 5.2. The conclusion from the analysis of this case is that the team should consider introducing more comprehensive requirement documentation. Firstly, it seems that they are unable to encourage the customer to cooperate closely, and secondly, they are developing a safety-critical system.

Similarly to the influence assessment, the result of the contextual assessment of a practice can also be aggregated to a single number (see line 26 of Algorithm 2). The aggregated assessment result enables calculating the previously introduced *TL* and *PIP* measures at the level of contextual assessment. For example, for the case presented in Figure 5, the final influence assessment of the practice P5.2 would be equal to 2 (influence assessment) – 3 + 0.

Finally, influence and contextual assessments help to identify one more anomaly in the implementation of practices. We call it a *false-positive practice*. This is a practice that was indicated as *correctly and frequently* used during basic assessment, but is either unsupported by other related practices or remains ill-fitted to the context (i.e., received a large number of '−' signs as a result of influence and contextual assessments).

6. Related Work

In this section, we would like to complement the conceptual framework of maturity models and assessment methods presented in Section 2 by focusing on assessing the maturity of Requirements Engineering. We would also like to discuss how these methods relate to our work.

The simplest approach to probe and assess agile software development processes is to use one of the available checklists (e.g., [92]) or simple, survey-based tests (e.g., Karlskrona test [43]). The advantage of such approaches is that they are easy and cost-effective to apply. Unfortunately, when looking from the perspective of assessing the maturity of RE process, they seem over-simplified and too general. We use elements of checklist-based assessment in REMMA primarily to evaluate the fulfillment of the minimal quality requisites of practices.

When considering maturity assessment and improvement methods in the area of RE, the REGPG framework proposed by Sommerville and Sawyer [21] seemed to be one of the most recognized approaches. It is a framework dedicated to assessing the maturity of traditional RE. Therefore, it is not entirely suitable for assessing the agile RE process [93]. In REMMA, we use the practices-assessment scale proposed by REGPG (de facto standard, normal use, discretionary use, never used). However, we decided to provide more specific definitions of its items to ensure that it could be treated as an interval scale (see Section 5.1 for details).

Having had some experience with applying REGPG, Niazi et al. [25] proposed the RE Maturity Measurement Framework (REMMF). They built their framework upon

Data:
P_C — a set of practices introduced by the context-factors model;
P' — a set containing all practices, i.e., introduced by REMMA and context-factors models;
IR_C — a set of influence relationships introduced by the context-factors model;
RR — a set containing response relationships introduced by the context-factors model;
C — a set containing context-factors values that appeared in the project under assessment;
$p.basic\_assessment$ — the result of basic assessment (number of levels) for the practice p;
$p.influence\_assessment$ — the result of influence assessment (number of levels) for the practice p;
$rel.neutral\_level$ — neutral level defined for the response relationship rel;
Result:
$p.context\_responses$ — a set containing tuples (lvl, rel) such that lvl is the level of response the practice p had on a context-factor value, triggered by the response relationship rel;
$p.total\_context\_responses$ — is the sum of all response levels the practice p has on context-factor values;
$ctx\_value.responses$ — a set containing tuples (lvl, rel) such that lvl is the level of response a practice had on the context-factor value ctx_value, triggered by the response relationship rel;
$p.context\_assessment$ — is the contextual assessment of the practice p;

```
perform basic assessment for all practices in P_C;
perform influence assessment (Algorithm 1) for the
  practices in P' and influence relationships in IR_C;
foreach p ∈ P' do
    p.context_responses ← {};
    p.total_context_responses ← 0;
foreach ctx_value ∈ C do
    ctx_value.responses ← {};
    foreach rel: a --inv r type--> ctx_value ∈ RR do
        lvl ← a.basic_assessment
          − rel.neutral_level;
        if (lvl < 0 and type = '−')
          or (lvl > 0 and type = '+')
          or type = '±' then
            if inv = True then
                lvl ← −lvl;
            a.context_responses ←
              a.context_responses ∪ {(lvl, rel)};
            ctx_value.responses ←
              ctx_value.responses ∪ {(lvl, rel)};
foreach p ∈ P' do
    foreach resp ∈ p.context_responses do
        p.total_context_responses ←
          p.total_context_responses + resp[0];
    p.context_assessment ←
      p.influence_assessment
      + p.total_context_responses;
```

Algorithm 2: Algorithm of performing contextual assessment for all practices.

REGPG's strong point—a well-defined set of practices, and tried to overcome its weaknesses, e.g., one-dimensional evaluation, lack of indicators for practice evaluation. To address these issues, they proposed a new multi-criterion measurement instrument (see Section 2.2). An evaluation in two organizations demonstrated that it was considered effective in assessing the maturity level as well as identifying weak and strong RE practices. Finally, the authors of this method concluded that REMMF is general enough to be applied in various types of organizations. However, considering that the method was created to assess traditional RE, its applicability to appraising agile RE seems doubtful.

In order to make one of the most popular Software Process Improvement frameworks—the CMM family—focused on RE, R-CMM [94] was proposed. The main goal was to introduce a mechanism that could support organizations in selecting appropriate strategies for implementing their RE processes. To achieve their objectives, the authors modified the GQM (Goal-Question-Metric) notation into the Goal-Question-Practice format (GQP). They claimed that introducing GQP made it possible to provide an explicit rationale and explanation for using a specific set of practices. Later, the model was adapted to CMMI. As a result, a new R-CMMI model was presented [95], which further evolved into the REPAIM [96] model. The latest version, which resigned from the GQP format in favor of the goals' definition notation used in CMMI, was validated by a panel of experts.

Another idea for constructing a maturity model for RE was proposed by Gorschek [22]. In his REPM model, he introduced the concept of actions (activity) in place of practices. He also suggested presenting the results in tables and line graph charts summarizing the number of actions (total, completed, inappropriate), as well as in natural language descriptions. We decided to follow these guidelines in REMMA and propose aggregating assessment results as numerical indicators (i.e., TL and PIP), graph charts (e.g., Figure 4), and descriptive causal analyses of relationships between practices and context factors.

From the REMMA perspective, the approaches depending mainly on assessing the implementation of practices, such as CMMI [27] or the Agile Maturity Model (AMM) [14], are capable of performing assessment at the basic assessment level. As a result, obtaining highly positive assessment results could lead to a false-positive conviction about the maturity of agile RE process, because these methods do not consider the needs of adapting to the environment and preserving the synergistic nature of agile practices.

Despite this limitation, the aforementioned approaches are still valuable sources of information with respect to practices applied generally in Software Engineering (e.g., CMMI [27]), important for RE specifically (e.g., REPM [22], REPAIM [96], REMMF [25]) and agile software development (e.g., SAMI [15], AMM[14]).

7. Conclusions

In the paper, we proposed a method called REMMA that allows assessing the maturity of Requirements Engineering in agile projects based on the concept of alignment between practices. We call the approach a hybrid because it combines elements known from prescriptive and problem-oriented approaches to process improvement.

We developed the method based on the literature concerning agile software development and opinions of IT professionals. The resulting REMMA method consists of two main components: the *maturity model* consisting of state-of-the-art agile RE practices, relationships between practices, and context factors; and the *assessment method*. The assessment method allows appraising the implementation of practices also from the perspective of their appropriateness for the context of a project.

Appendix A. Practices catalog

In the appendix, we present the default catalog of Agile RE practices in Table A.8.

Appendix A.1. Context-specific practices

In the appendix we present the practices introduced by the context factors described in Section 4.3.

C2: Scrum

C2.P1: **Product Backlog** *(Critical)* (PA5)

Product backlog is one of the artifacts defined by Scrum Framework. It is "*an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product.* " [8]

The role responsible for managing product backlog is product owner. He or she is responsible for managing content of product backlog, making it available to the stakeholders, and order backlog items according to their importance. This work is done continuously because product backlog constantly evolves during the project.

Attributes of a properly managed product backlog might be summarized using the acronym DEEP proposed by Pichler [120]: Detailed appropriately, Estimated, Emergent, and Prioritized.

Minimal quality requisites:

- backlog items that will be done soon are sufficiently well described so that they can be implemented in the coming sprint,
- backlog items are estimated,
- backlog is refined at least once a sprint (items are added, removed or reprioritized as needed),
- backlog items are sorted with the most important (valuable) items at the top and the least important at the bottom.

C2.P2: **Sprint Backlog** *(Critical)* (PA6)

Sprint backlog is "*the set of product backlog items selected for the sprint, plus a plan for delivering the product Increment and realizing the sprint goal.* " [8] It is a tool supporting planning and monitoring progress created during sprint planning that is owned by the development team.

Minimal quality requisites:

- backlog is created as a result of spring planning session,
- backlog is updated at least once a day to show current progress of a sprint.

C2.P3: **Decisive Product Owner** *(Critical)* (PA1)

Product owner is "*the sole person responsible for managing the Product Backlog.*" [8] Among these management responsibilities the most important one is to optimize the value of the work that Development Team performs.

By introducing the role of product owner, Scrum tries to mitigate the problems related to making decisions and taking responsibility for actions when there are many stakeholders in a project. That it is why product owner is one person, not a committee, and to work effectively he or she needs to be made accountable for making decisions regarding the product development.

Minimal quality requisites:

- the role of product owner is assigned to a single person,
- the person is accountable for making decisions regarding the product development.

C3: The Global Software Development

C3.P1: **Frequent visits** *(Important)* (PA3)

Agile methods advocate for relying on face-to-face communication and building self-organizing teams. Achieving both of these qualities might be difficult when a project team is distributed between different locations. Therefore, it is recommended to promote visits of team members to help them build trust, personal relationships and better understand the vision of a product [54].

Minimal quality requisites:

- cross-location visits are organized with such frequency that they enable to build trust among team members and help them establish a single understanding of product vision (a certain frequency might depend on cultural differences, offshore/inshore development, etc.).

C3.P2: **Synchronized work hours** *(Important)* (PA3)

Taking into account that a distributed team may have different working hours between the different locations, it is recommended to establish a time-window in which all team members are available, and direct communication is possible [54].

Minimal quality requisites:

- a common time-window is agreed when all team members should be available and can directly communicate,
- all team members are available within the agreed time window.

C3.P3: **Multiple communication modes** *(Additional)* (PA3)

The practice ensures that a project team with distributed project stakeholders is supported with different options of communication tools, e.g., phone, web camera, teleconference, video conference, web conference, net meeting, email, shared mailing list, Instant Message (IM), Short Message Service (SMS), or Internet chat [54].

Minimal quality requisites:

- at least two alternative communication modes are available for project stakeholders.

C4. Safety-critical system

C4.P1: **Prepare up front a requirements specification that is sufficient for safety analysis** *(Important)* (PA5)

An up-front analysis of a problem should be performed to identify the requirements related to reliability and safety, as well as key requirements that could potentially affect the safety and reliability characteristics of the system. The prospered requirements specification should be sufficient to propose an architecture of a system and perform required safety analyses [82, 83, 84], e.g., Safety Impact Analysis, Functional Failure Analysis (FFA), Hazards and operability analysis (HAZOP).

Minimal quality requisites:

- an up-front analysis of requirements has been performed that is sufficient to prepare for required safety analyses.

Table A.8: Agile RE Practices with their descriptions.

ID	Description with references to sources
P01	**Available / On-site customer.** The customer/user is available and able to answer questions regarding requirements in such a way that the response time does not negatively impact the work of the development team (in particular, works on-site with developers) [97, 26, 98, 53, 7, 99, 100, 101, 102, 103, 104].
P02	**Involve different stakeholders.** The stakeholders in the project are identified and have their roles and responsibilities defined. The analysis of requirements is performed by taking into account similarities (overlaps), inconsistencies and contradictions between different stakeholders' viewpoints, functional areas, and quality expectations [103, 104, 105, 67, 106, 70].
P03	**Establish project's shared vision.** The problem to be solved (goal of the project) and the proposed solution are well defined and kept up-to-date[98, 107, 70, 108, 109, 110].
P04	**Create prototypes to better understand requirements.** A prototype is created for functional requirements (e.g., wireframes, mockup, storyboard, etc.). The customer validates the created prototypes and provides feedback[97, 11, 107, 111, 112, 113, 59].
P05	**Define project / product constraints.** Project and product constraints are defined and kept up-to-date[106, 70].
P06	**Organize everyday team meetings.** The team discusses the progress, evaluate the feasibility of iteration, plan and adjust the plan (if it is required) (e.g., each team member states what he/she was able to accomplish on the previous day, what he/she is going to work on that day and informs about impediments)[98, 107, 114, 8].
P07	**Organize review meetings.** A meeting is conducted after an iteration during which the team presents its goal/scope, the work completed, the key decisions and a demo of the completed work. The feedback from stakeholders shall be received[97, 11, 26, 98, 107, 8].
P08	**Organize retrospective meetings.** After each finished iteration, a meeting is organized to discuss all issues that appeared during the iteration and to define improvement actions[97, 98, 107, 8].
P09	**Provide and maintain informative workspace.** Along with face-to-face communication, team members have constant access (either in the workplace or through a software system) to information such as project status. e.g., a burndown chart, completed tasks, currently developed tasks, awaiting tasks, requirements, reference materials such as law regulations, business documents etc.[115, 98, 58].
P10	**Provide easy access to requirements.** The requirements are stored in a place agreed upon by all project stakeholders, e.g., in a software system, that allows easy access for the development team, customer, and all authorized stakeholders.[116].
P11	**Maintain information about 'bad smells' and best practices related to requirements.** The information about problems related to defects (or misunderstandings) in requirements and to requirement process (elicitation, analysis, documentation, verification) and their impact on a project is stored, kept up-to-date and available to team members [59, 117].
P12	**Perform the 'elevator test'.** All team members are able to pass the 'elevator test', i.e., to explain the idea of a product in the time it takes to ride up in an elevator[118, 119, 120, 121].
P13	**Let customer define acceptance tests.** Acceptance criteria and the corresponding acceptance test scenarios are defined by the customer, or at least the customer accepts them[97, 7, 70, 59, 122, 123, 124, 125].
P14	**Prepare and maintain automatic acceptance tests.** An automatic version of each acceptance test is implemented and kept up-to-date[97, 7, 70, 59, 58, 122, 123].
P15	**Prepare acceptance tests before coding.** Acceptance criteria and acceptance tests for each requirement are defined before a requirement is included into the scope of the iteration [97, 11, 26, 107, 114, 126, 127, 128, 129, 130].
P16	**Perform regression acceptance testing.** Acceptance tests for previously implemented functions are executed in the following iteration to verify if they are still meeting requirements [107, 131, 132].
P17	**Cover requirements with acceptance tests.** Acceptance criteria and acceptance test scenarios are defined for requirements (including non-functional requirements)[97, 70, 59].
P18	**Make requirements independent.** The requirements identified as dependent are grouped together into one or more independent requirements [97, 11, 26, 133, 68].
P19	**Write short, negotiable requirements.** Each requirement's description includes the most important information from the user's perspective, any additional information (extra precision) is separated — attached to the core requirement as annotations [97, 11, 26, 133, 70, 59, 68].
P20	**Make complex requirements divisible.** All sub-requirements of a compound requirement are identified in order to ease its decomposition, and the requirements identified as difficult to implement are augmented with the description of issues, concerns to help the development team extract some spike solution tasks [97, 11, 26, 101, 59, 68, 79].
P21	**Requirements should be valuable to purchasers or users.** Each requirement is defined by the customer or at least the customer accepts the requirement proposed by the development team [97, 11, 26, 133, 134].
P22	**Make requirements estimable.** Size of each requirement is small enough to enable its effort estimation by the development team, and the domain vocabulary used for its description is clear to them [97, 26, 11, 59].
P23	**Make requirements testable.** It is possible to verify that the software meets the acceptance criteria in a reasonable time and using a reasonable amount of resources [97, 11, 26, 133, 59].
P24	**Follow the user role modeling approach.** An identification of the user roles is performed (e.g., brainstorming session), the dependencies between user roles and user classes are identified, and attributes, properties of each user role and associated user classes are defined (e.g., the frequency of system usage by user class members; users' level of expertise in domain) [59, 135, 136].
P25	**Let customer prioritize requirements.** The priority of each requirement is provided by the customer or at least accepted by him [97, 26, 107, 8, 68].
P26	**Define requirements using notation and language that can easily be understood by all stakeholders.** The notation used to document a requirement can easily be understood by all stakeholders [97, 53, 36, 81, 137, 138].
P27	**Assess implementation risks for requirements.** The implementation risks of each requirement are analyzed and documented [97, 11, 58].
P28	**Negotiate iteration scope with customer.** The scope of each iteration is negotiated between the customer and the development team until a consensus is achieved. The customer understands justifications for estimates of requirements included in the scope of the iteration and the development team understands the business rationale behind these requirements [97, 26, 107].
P29	**Avoid changing increment scope after it is agreed upon.** The scope of each iteration does not get changed after it has been started unless there is a mutual agreement between the development team and the customer (the scope is renegotiated) [97, 70, 8].
P30	**Keep iteration length short to continually collect feedback.** The length of iteration is no longer than 4 weeks [97, 11, 107, 70, 8, 79].
P31	**Define a fixed iteration length.** The length of each iteration is defined, justified, and does not change (excluding external causes like national holidays, etc.) [97, 70, 68].

C4.P2: **Write comprehensive, negotiable requirements** *(Important)* (PA5)

In some cases, writing user stories might be insufficient to document requirements for safety-critical systems. Therefore, it is recommended to use more detailed techniques (e.g., use cases) whenever it seems rational [55, 86].

Minimal quality requisites:

- functional requirements that seem important from the perspective of meeting safety and reliability requirements are analyzed and documented,
- non-functional requirements are analyzed and documented.

C4.P3: **Traceability** *(Important)* (PA5)

It is required for projects aiming at the development of a safety-critical system to preserve traceability between code, requirements, and test cases [86, 89, 85].

Minimal quality requisites:

- traceability between each requirement and its implementation is preserved,
- traceability between each requirement and its test cases is preserved.

References

[1] D. Damian, D. Zowghi, L. Vaidyanathasamy, Y. Pal, An Industrial Case Study of Immediate Benefits of Requirements Engineering Process Improvement at the Australian Center for Unisys Software, Empirical Software Engineering 9 (1) (2004) 45–75.

[2] J. G. Brodman, D. L. Johnson, Return on Investment (ROI) from Software Process Improvement as Measured by US Industry, Software Process: Improvement and Practice 1 (1) (1995) 35–47.

[3] L. May, Major Causes of Software Project Failures, CrossTalk–The Journal of Defense Software Engineering 11 (7) (1998) 9–12.

[4] R. N. Charette, Why software fails, IEEE spectrum 42 (9) (2005) 36.

[5] J. Verner, K. Cox, S. Bleistein, N. Cerpa, Requirements Engineering and Software Project Success: an industrial survey in Australia and the US, Australasian Journal of Information Systems 13 (1).

[6] L. A. Kappelman, R. McKeeman, L. Zhang, Early warning signs of it project failure: The dominant dozen, Information systems management 23 (4) (2006) 31–36.

[7] K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 2000.

[8] K. Schwaber, J. Sutherland, The Scrum Guide™. The Definitive Guide to Scrum: The Rules of the Game. Scrum.org (2013).

[9] A. Cockburn, Crystal clear: a human-powered methodology for small teams, Pearson Education, 2004.

[10] L. Cao, B. Ramesh, Agile requirements engineering practices: An empirical study, Software, IEEE 25 (1) (2008) 60–67.

[11] H. Elshandidy, S. Mazen, Agile and traditional requirements engineering: A survey, International Journal of Scientific & Engineering Research 9.

[12] S. Adikari, C. McDonald, J. Campbell, Little design up-front: a design science approach to integrating usability into agile requirements engineering, in: Human-computer interaction. New trends, Springer, 2009, pp. 549–558.

[13] K. Beck, et.al, The Agile Manifesto. `http://agilemanifesto.org`, last access 28/08/2015.

[14] C. Patel, M. Ramachandran, Agile maturity model (amm): A software process improvement framework for agile software development practices, Int. J. of Software Engineering, IJSE 2 (1) (2009) 3–28.

[15] A. Sidky, A Structured Approach to Adopting Agile Practices: The Agile Adoption Framework, Ph.D. thesis, Virginia Polytechnic Institute and State University (2007).

[16] D. J. Reifer, F. Maurer, H. Erdogmus, Scaling agile methods, Software, IEEE 20 (4) (2003) 12–14.

[17] R. Hoda, P. Kruchten, J. Noble, S. Marshall, Agility in context, in: Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '10, ACM, New York, NY, USA, 2010, pp. 74–88.

[18] P. Kruchten, Contextualizing agile software development, Journal of Software: Evolution and Process 25 (4) (2013) 351–361.

[19] F. Grossman, J. Bergin, D. Leip, S. Merritt, O. Gotel, One xp experience: introducing agile (xp) software development into a culture that is willing but not ready, in: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 2004, pp. 242–254.

[20] N. P. Napier, L. Mathiassen, R. D. Johnson, Combining perceptions and prescriptions in requirements engineering process assessment: An industrial case study, IEEE Transactions on Software Engineering 35 (5) (2009) 593–606.

[21] I. Sommerville, P. Sawyer, Requirements engineering: a good practice guide, John Wiley & Sons, Inc., 1997.

[22] T. Gorschek, A method for assessing requirements engineering process maturity in software projects, Ph.D. thesis, Department of Software Engineering and Computer Science Blekinge Institute of Technology (2002).

[23] T. Päivärinta, K. Smolander, Theorizing about software development practices, Science of Computer Programming 101 (0) (2015) 124 – 135, towards general theories of software engineering.

[24] P. E. MaMahon, Integrating CMMI and Agile Developement, Addison-Wesley, 2011.

[25] M. Niazi, K. Cox, J. Verner, A measurement framework for assessing the maturity of requirements engineering process, Software Quality Journal 16 (2) (2008) 213–235.

[26] R. Fontana, I. Fontana, P. da Rosa Garbuio, S. Reinehr, A. Malucelli, Processes versus people: How should agile software development maturity be defined?, Journal of Systems and Software 97 (0) (2014) 140–155.

[27] CMMI Product Team, CMMI® for Development, Version 1.3, Tech. rep., Carnegie Mellon University (November 2010).

[28] M. Paulk, A taxonomy for improvement frameworks, in: presented at the World Congr. Softw. Qual., 2008.

[29] T. Kähkönen, P. Abrahamsson, Achieving cmmi level 2 with enhanced extreme programming approach, in: Product Focused Software Process Improvement, Springer, 2004, pp. 378–392.

[30] M. Fritzsche, P. Keil, et al., Agile methods and cmmi: compatibility or conflict?, e-Informatica 1 (1) (2007) 9–26.

[31] M. C. Paulk, Extreme programming from a cmm perspective, Software, IEEE 18 (6) (2001) 19–26.

[32] C. Vriens, Certifying for cmm level 2 and is09001 with xp@ scrum, in: Agile Development Conference, 2003. ADC 2003. Proceedings of the, IEEE, 2003, pp. 120–124.

[33] D. J. Anderson, Stretching agile to fit cmmi level 3-the story of creating msf for cmmi® process improvement at microsoft corporation, in: Agile Conference, 2005. Proceedings, IEEE, 2005, pp. 193–201.

[34] M. Pikkarainen, A. Mäntyniemi, An approach for using cmmi in agile software development assessments: experiences from three case studies, in: SPICE 2006 conference, Luxemburg, 2006, pp. 4–5.

[35] S. W. Baker, Formalizing agility: an agile organization's journey toward cmmi accreditation, in: Agile Conference, 2005. Proceedings, IEEE, 2005, pp. 185–192.

[36] B. Meyer, Agile!: The Good, the Hype and the Ugly, Springer Science & Business Media, 2014.

[37] J. A. H. Alegrıa, M. C. Bastarrica, Implementing cmmi using a combination of agile methods, CLEI Electronic Journal 9 (1).

[38] v. n. p. y. p. Marçal, Ana Sofia C and de Freitas, Bruno Celso C and Soares, Felipe S Furtado and Furtado, Maria Elizabeth S and Maciel, Teresa M and Belchior, Arnaldo D, journal=Innovations in Systems and Software Engineering, Blending scrum practices and cmmi project management process areas.

[39] J. Diaz, J. Garbajosa, J. A. Calvo-Manzano, Mapping cmmi level 2 to scrum practices: An experience report, in: Software Process Improvement, Springer, 2009, pp. 93–104.

[40] R. Turner, A. Jain, Agile meets cmmi: Culture clash or common cause?, in: Extreme Programming and Agile Methods—XP/Agile Universe 2002, Springer, 2002, pp. 153–165.

[41] J. Nawrocki, B. Walter, A. Wojciechowski, Toward maturity model for extreme programming, in: Euromicro Conference, 2001. Proceedings. 27th, IEEE, 2001, pp. 233–239.

[42] A. Omran, Agile cmmi from smes perspective, in: Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on, IEEE, 2008, pp. 1–8.

[43] M. Seuffert, Agile Karlskrona test. A generic agile adoption test. `http://mayberg.se/learning/karlskrona-test`, last visit: 12th September 2014 (2009).

[44] SCAMPI Upgrade Team, Standard CMMI® Appraisal Method for Process Improvement (SCAMPI^{SM}) A, Version 1.3: Method Definition Document, Tech. rep., Carnegie Mellon University (March 2011).

[45] I. Sommerville, J. Ransom, An empirical study of industrial requirements engineering process assessment and improvement, ACM Transactions on Software Engineering and Methodology (TOSEM) 14 (1) (2005) 85–117.

[46] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS quarterly 28 (1) (2004) 75–105.

[47] R. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, 2014.

[48] V. Venkatesh, H. Bala, Technology acceptance model 3 and a research agenda on interventions, Decision sciences 39 (2) (2008) 273–315.

[49] T. Schweigert, D. Vohwinkel, M. Korsaa, R. Nevalainen, M. Biro, Agile maturity model: analysing agile maturity characteristics from the spice perspective, Journal of Software: Evolution and Process 26 (5) (2014) 513–520.

[50] P. Agerfalk, B. Fitzgerald, Flexible and distributed software processes: Old petunias in new bowls?, Communcations of the ACM 49 (10).

[51] D. Turk, R. France, B. Rumpe, Assumptions underlying agile software development processes, Journal of Database Management (JDM) 16 (4) (2005) 62–87.

[52] F. Navarrete, P. Botella, X. Franch, An approach to reconcile the agile and cmmi contexts in product line development, in: Proceed. of the 1st Internat. Workshop on Agile Product Line Engineering (APLE'06), 2006.

[53] D. Rosenberg, M. Stephens, Extreme programming refactored: the case against XP, Apress, 2003.

[54] E. Hossain, M. A. Babar, H.-y. Paik, Using scrum in global software development: A systematic literature review, in: Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on, Ieee, 2009, pp. 175–184.

[55] J. Grenning, Launching extreme programming at a process-intensive company, IEEE Software (6) (2001) 27–33.

[56] M. Ochodek, S. Kopczyńska, Perceived importance of agile requirements engineering practices–a survey, Journal of Systems and Software 143 (2018) 29–43.

[57] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, S. Shamshirband, A systematic literature review on agile requirements engineering practices and challenges, Computers in human behavior.

[58] M. Cohn, Succeeding with agile: software development using Scrum, Pearson Education, 2010.

[59] M. Cohn, User stories applied: For agile software development, Addison-Wesley Professional, 2004.

[60] M. Fowler, J. Highsmith, The agile manifesto, Software Development 9 (8) (2001) 28–35.

[61] D. Carlson, P. Matuzic, Practical agile requirements engineering, in: Proceedings of the 13th Annual Systems Engineering Conference, 2010.

[62] M. Kassab, An empirical study on the requirements engineering practices for agile software development, in: Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, IEEE, 2014, pp. 254–261.

[63] M. Kassab, C. Neill, P. Laplante, State of practice in requirements engineering: contemporary data, Innovations in Systems and Software Engineering 10 (4) (2014) 235–241.

[64] S. McConnell, Software estimation: demystifying the black art, Microsoft press, 2006.

[65] P. Hill, et al., Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration, McGraw Hill Professional, 2010.

[66] M. Drury, K. Conboy, K. Power, Obstacles to decision making in agile software development teams, Journal of Systems and Software 85 (6) (2012) 1239–1254.

[67] M. Daneva, E. Van Der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar, R. Wieringa, Agile requirements prioritization in large-scale outsourced system projects: An empirical study, Journal of systems and software 86 (5) (2013) 1333–1353.

[68] M. Cohn, Agile estimating and planning, Pearson Education, 2005.

[69] B. Victor, N. Jacobson, We didn't quite get it, in: 2009 Agile Conference, IEEE, 2009, pp. 271–274.

[70] K. S. Rubin, Essential Scrum: A practical guide to the most popular Agile process, Addison-Wesley, 2012.

[71] E. Bjarnason, K. Wnuk, B. Regnell, A case study on benefits and side-effects of agile practices in large-scale requirements engineering, in: Proceedings of the 1st Workshop on Agile Requirements Engineering, ACM, 2011, p. 3.

[72] L. Cao, Estimating agile software project effort: an empirical study, AMCIS 2008 Proceedings.

[73] M. Lang, K. Conboy, S. Keaveney, Cost estimation in agile software development projects, in: Information Systems Development, Springer, 2013, pp. 689–706.

[74] J. Cho, Issues and challenges of agile software development with scrum, Issues in Information Systems 9 (2) (2008) 188–195.

[75] D. Turk, R. B. France, B. Rumpe, Limitations of agile software processes, in: Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002, 2002, pp. 43–46.

[76] H. Sharp, H. Robinson, An ethnographic study of xp practice, Empirical Software Engineering 9 (4) (2004) 353–375.

[77] S. Jalali, C. Wohlin, Agile practices in global software engineering-a systematic map, in: Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on, IEEE, 2010, pp. 45–54.

[78] H. Holmström, B. Fitzgerald, P. J. Ågerfalk, E. Ó. Conchúir, Agile practices reduce distance in global software development, Information Systems Management 23 (3) (2006) 7–18.

[79] S. Berczuk, Back to basics: The role of agile principles in success with an distributed scrum team, in: Agile Conference (AGILE), 2007, IEEE, 2007, pp. 382–388.

[80] B. Boehm, Get ready for agile methods, with care, Computer 35 (1) (2002) 64–69.

[81] E. Hochmüller, R. T. Mittermeir, Agile process myths, in: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral, ACM, 2008, pp. 5–8.

[82] X. Ge, R. F. Paige, J. McDermid, et al., An iterative approach for development of safety-critical software and safety arguments, in: Agile Conference (AGILE), 2010, IEEE, 2010, pp. 35–43.

[83] A. Sidky, J. Arthur, Determining the applicability of agile practices to mission and life-critical systems, in: Software En-

gineering Workshop, 2007. SEW 2007. 31st IEEE, IEEE, 2007, pp. 3–12.
[84] A. Garg, Agile software development, DRDO Science Spectrum (2009) 55–59.
[85] M. McHugh, F. McCaffery, V. Casey, Barriers to adopting agile practices when developing medical device software, in: Software Process Improvement and Capability Determination, Springer, 2012, pp. 141–147.
[86] H. Jonsson, S. Larsson, S. Punnekkat, Agile practices in regulated railway software development, in: Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on, IEEE, 2012, pp. 355–360.
[87] S. H. VanderLeest, A. Buter, Escape the waterfall: Agile for aerospace, in: Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th, IEEE, 2009, pp. 6–D.
[88] J. Trimble, C. Webster, Agile development methods for space operations, in: The 12th International Conference on Space Operations, 2012.
[89] K. Gary, A. Enquobahrie, L. Ibanez, P. Cheng, Z. Yaniv, K. Cleary, S. Kokoori, B. Muffih, J. Heidenreich, Agile methods for open source safety-critical software, Software: Practice and Experience 41 (9) (2011) 945–962.
[90] J. Bowers, J. May, E. Melander, M. Baarman, A. Ayoob, Tailoring xp for large system mission critical software development, in: Extreme Programming and Agile Methods—XP/Agile Universe 2002, Springer, 2002, pp. 100–111.
[91] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, M. Zelkowitz, Empirical findings in agile methods, in: Extreme Programming and Agile Methods—XP/Agile Universe 2002, Springer, 2002, pp. 197–207.
[92] C. R. Jakobsen, K. A. Johnson, Mature agile with a twist of cmmi, in: Agile, 2008. AGILE'08. Conference, IEEE, 2008, pp. 212–217.
[93] J. Nawrocki, M. Jasiñski, B. Walter, A. Wojciechowski, Extreme programming modified: embrace requirements engineering practices, in: Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on, IEEE, 2002, pp. 303–310.
[94] S. Beecham, T. Hall, A. Rainer, Defining a requirements process improvement model, Software Quality Journal 13 (3) (2005) 247–279.
[95] B. Solemon, S. Shahibuddin, A. Ghani, Re-defining the requirements engineering process improvement model, in: Software Engineering Conference, 2009. APSEC'09. Asia-Pacific, Ieee, 2009, pp. 87–92.
[96] B. Solemon, S. Sahibuddin, A. A. A. Ghani, A new maturity model for requirements engineering process: an overview, Journal of Software Engineering and Applications 5 (5) (2012) 340–350.
[97] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, S. Shamshirband, A systematic literature review on agile requirements engineering practices and challenges, Computers in human behavior 51.
[98] P. Diebold, M. Dahlem, Agile practices in practice: a mapping study, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2014, p. 30.
[99] R. Hoda, J. Noble, S. Marshall, The impact of inadequate customer collaboration on self-organizing agile teams, Information and Software Technology 53 (5) (2011) 521–534.
[100] M. Korkala, M. Pikkarainen, K. Conboy, A case study of customer communication in globally distributed software product development, in: Proceedings of the 11th International Conference on Product Focused Software, ACM, 2010, pp. 43–46.
[101] P. Abrahamsson, J. Koskela, Extreme programming: A survey of empirical data from a controlled case study, in: Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on, IEEE, 2004, pp. 73–82.
[102] A. Martin, J. Noble, R. Biddle, Being jane malkovich: A look into the world of an xp customer, in: Extreme Programming and Agile Processes in Software Engineering, Springer, 2003, pp. 234–243.
[103] S. Mohammadi, B. Nikkhahan, S. Sohrabi, An analytical survey of" on-site customer" practice in extreme programming, in: Computer Science and its Applications, 2008. CSA'08. International Symposium on, IEEE, 2008, pp. 1–6.
[104] S. Mohammadi, B. Nikkhahan, S. Sohrabi, Challenges of user involvement in extreme programming projects, International Journal of Software Engineering and Its Applications 3 (1).
[105] K. Pohl, Requirements engineering: fundamentals, principles, and techniques, Springer Publishing Company, Incorporated, 2010.
[106] W. Karl, Software requirements, Microsoft Press.
[107] P. Abrahamsson, O. Salo, J. Ronkainen, Agile software development methods: review and analysis, Technical report, VTT Electronics and University of Oulu (2002).
[108] K. Vlaanderen, S. Jansen, S. Brinkkemper, E. Jaspers, The agile requirements refinery: Applying scrum principles to software product management, Information and software technology 53 (1) (2011) 58–70.
[109] J. E. Hannay, H. C. Benestad, Perceived productivity threats in large agile development projects, in: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2010, p. 15.
[110] K. Conboy, X. Wang, B. Fitzgerald, Creativity in agile systems development: A literature review, in: Information Systems–Creativity and Innovation in Small and Medium-Sized Enterprises, Springer, 2009, pp. 122–134.
[111] R. Muñoz, H. H. Miller-Jacobs, In search of the ideal prototype, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, 1992, pp. 577–579.
[112] J. Arnowitz, M. Arent, N. Berger, Effective prototyping for software makers, Elsevier, 2010.
[113] T. Memmel, F. Gundelsweiler, H. Reiterer, Agile human-centered software engineering, in: Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 1, British Computer Society, 2007, pp. 167–175.
[114] A. Begel, N. Nagappan, Usage and perceptions of agile software development in an industrial context: An exploratory study, in: 1st Intl Symp. on Empirical Software Engineering and Measurement (ESEM), 2007.
[115] T. Chow, D.-B. Cao, A survey study of critical success factors in agile software projects, Journal of Systems and Software 81 (6) (2008) 961–971.
[116] T. Chau, F. Maurer, Knowledge sharing in agile software teams, in: Logic versus approximation, Springer, 2004, pp. 173–183.
[117] K. Power, Definition of ready: An experience report from teams at cisco, in: Agile Processes in Software Engineering and Extreme Programming, Springer, 2014, pp. 312–319.
[118] G. A. Moore, Crossing the Chasm: Marketing and selling high-tech goods to mainstream customers, New York, HarperBusiness, 1991.
[119] J. Highsmith, Agile project management: creating innovative products, Pearson Education, 2009.
[120] R. Pichler, Agile product management with scrum: Creating products that customers love, Addison-Wesley Professional, 2010.
[121] L. Morris, M. Ma, P. Wu, Agile Innovation: The Revolutionary Approach to Accelerate Success, Inspire Engagement, and Ignite Creativity. 2014, New York: Wiley, 2014.
[122] R. Mugridge, W. Cunningham, Fit for developing software: framework for integrated tests, Pearson Education, 2005.
[123] I. Dees, M. Wynne, A. Hellesoy, Cucumber Recipes: Automate Anything with BDD Tools and Techniques, Pragmatic Bookshelf, 2013.
[124] B. Haugset, G. K. Hanssen, Automated acceptance testing: A literature review and an industrial case study, in: Agile, 2008. AGILE'08. Conference, IEEE, 2008, pp. 27–38.
[125] N. B. Harrison, A study of extreme programming in a large

company, Avaya Labs.
[126] V. Massol, T. Husted, Junit in action, Manning Publications Co., 2003.
[127] L. Koskela, Test Driven: Practical TDD and Acceptance TDD for Java Developers, Manning Publications Co., 2007.
[128] C. Solís, X. Wang, A study of the characteristics of behaviour driven development, in: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, IEEE, 2011, pp. 383–387.
[129] D. C. Gause, G. M. Weinberg, Exploring requirements: quality before design, Dorset House New York, 1989.
[130] R. C. Martin, G. Melnik, Tests and requirements, requirements and tests: A möbius strip, Software, IEEE 25 (1) (2008) 54–59.
[131] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability 22 (2) (2012) 67–120.
[132] C. Lowell, J. Stell-Smith, Successful automation of gui driven acceptance testing, in: Extreme programming and agile processes in software engineering, Springer, 2003, pp. 331–333.
[133] B. Wake, INVEST in good stories and SMART tasks. `http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/`, last access: 08/07/2015 (2003).
[134] K. Boness, R. Harrison, Goal sketching: Towards agile requirements engineering, in: null, IEEE, 2007, p. 71.
[135] L. L. Constantine, L. A. Lockwood, Software for use: a practical guide to the models and methods of usage-centered design, Pearson Education, 1999.
[136] J. P. Djajadiningrat, W. W. Gaver, J. Fres, Interaction relabelling and extreme characters: methods for exploring aesthetic interactions, in: Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques, ACM, 2000, pp. 66–71.
[137] M. Kajko-Mattsson, Problems in agile trenches, in: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, ACM, 2008, pp. 111–119.
[138] D. M. Berry, Ambiguity in natural language requirements documents, in: Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs, Springer, 2008, pp. 1–7.