A Root Cause Analysis Method for Preventing Erratic Behavior in Software Development: PEBA

Mr Hojat Mohammadnazar

PII: S0951-8320(18)30939-6
DOI: https://doi.org/10.1016/j.ress.2019.106565
Article Number: 106565
Reference: RESS 106565

To appear in: *Reliability Engineering and System Safety*

Received date: 30 July 2018
Revised date: 30 May 2019
Accepted date: 7 July 2019

Please cite this article as: Mr Hojat Mohammadnazar , A Root Cause Analysis Method for Preventing Erratic Behavior in Software Development: PEBA, *Reliability Engineering and System Safety* (2019), doi: https://doi.org/10.1016/j.ress.2019.106565

Highlights

- Preventing erratic behaviors could prevent faults from being introduced or going undetected.
- Mismatches between development context and practices could signal erratic behaviors.
- Proactive Erratic Behavior Analysis (PEBA) method as a proactive complement for existing RCA methods is developed.
- PEBA is resource-friendly, flexible and appropriate for SMEs as well as large organizations.

A Root Cause Analysis Method for Preventing Erratic Behavior in Software Development: PEBA

Mr Hojat Mohammadnazar
University of Jyväskylä
Faculty of Information Technology
P.O.Box 35 (Agora)
FI-40014 Jyväskylä
Finland
Phone: +358466150448
E-mail: homohamm@student.jyu.fi

Abstract
Measures taken to prevent faults from being introduced or going undetected can secure development of highly reliable software systems. One such measure is analyzing root causes of recurring faults and preventing them from appearing again. Previous methods developed for this purpose have been reactive in nature and relied heavily on fault reporting mechanisms of ogranizations. Additionally, previous efforts lack a defined mechanism for innovating corrective actions. In this study, we strive to complement the existing methods by introducing a proactive and qualitative method that does not rely on fault data. During the course of the research, in addition to an extensive literature search, an empirical field study is conducted with representatives of companies active in safety-critical and business-critical domains. Our proposed method relies on identifying mismatches between development practices and development context in order to predict erratic behaviors. Corrective actions in this method are innovated by resolving these mismatches. The use of the method is demonstrated in two safety-critical projects. Evaluation of the proposed method is done by two experts with respect to proactivity, resource-intensity, and effectiveness.
Keywords: Software Reliability, Fault Prevention, Fault Removal, Quality Assurance, Root Cause Analysis, Software Process Improvement

1 Introduction

With increasing presence of automated computation and networked communication, quality measures of systems responsible for delivering these services become critical. Reliability as the degree to which a system can continue to operate correctly in a specified duration of time has been a matter of concern in computer engineering from the early ages of computer evolution [1]. Over the past few decades, however, the intricacies of developing highly reliable software has come to the fore. Reliability of a system suffers with occurrences of service failures [2]. Unsatisfactory reliability might have catastrophic consequences on the user(s) and the environment especially in safety-critical [3] and business-critical [4] systems. Several instances of aircraft and spacecraft accidents due to software failures are presented in [5] and [6], respectively.

Even though it is a common practice, setting a numerical reliability target in terms of failures, time-between-failures or similar measures is not viewed as the most effective way to develop a highly reliable software system by all experts [7], [8]. The software reliability community has been challenged to leave the prevalent idea of software reliability modeling and provide *credible* methods for developing highly

reliable software systems [7]. Experts advocating the use of credible methods rather than reliability targets, argue that the rigor of practices and procedures could ensure development of highly reliable software. In fact, ECSS (i.e. European Cooperation for Space Standardization) software dependability and safety standard ECSS-Q-HB-80-03A [9], advised against using reliability models. Credible methods for developing highly-reliable software fall into four categories [2]: (1) fault prevention, (2) fault tolerance, (3) fault removal, and (4) fault forecasting [10]. Of these four, fault prevention and fault removal are primary means accessible to developers to stop faults from being introduced or to go undetected.

One well-known approach to fault prevention is to track down root causes of recurring faults and preventing them from appearing later on. This approach could be used to address faults in all the undertakings for quality improvement from inception to delivery. A myriad of methods have been introduced in the literature for this purpose, commonly known as Root Cause Analysis (RCA) [11]–[15]. Most RCA methods rely on statistical models of fault data for identifying recurring faults. To generate such models, the fault data should be collected in a formulated manner. Even though reliance on fault data is insightful [13], it comes at a high price for RCA methods. Fault data is difficult to collect [16]; and its collection needs upfront investment and personnel training [17]. Furthermore, RCA is a reactive practice by design.

In this study, realizing that erratic behavior is one of the main means of fault delivery, we set out to develop a method that proactively and without reliance on fault data could prevent erratic behavior. In the course of this research, we find that mismatches between development context and development practices provide a fertile ground for erratic behavior and consequently introduction of faults into software products. Resultantly, we develop *Proactive Erratic Behavior Analysis* (PEBA) method, aiming to prevent erratic behaviors by identifying such mismatches and resolving them. PEBA makes use of a taxonomy of contextual factors for mapping the development context. This taxonomy is developed in this research by conducting an extensive literature search and resembles situational factors reported by Clarke and O'Connor [18]. Using PEBA, introduction and non-detection of faults could be minimized. As such, PEBA complements the existing RCA methods.

2 Background

RCA is a structured investigation to identify the underlying causes of faults. By conducting an RCA, root causes of recurring faults are tracked down and resolved in order to prevent them from being introduced or going undetected. RCA can be performed both during the development and after product release. In the former case, RCA can result in in-process improvements [11], while in the latter, it helps create an organizational portfolio by which lessons learned from one project can be put into practice in later projects [19]. RCA can lead to improvements in artifacts of all stages of development.

Lehtinen et al. [15] identified three common steps to all RCA methods – (1) target problem detection, (2) root cause detection and (3) corrective action innovation. In target problem detection stage, recurring faults are identified. This is done either by qualitative analysis of faults by a team of experts [13], [15], [20] or by statistical analysis of fault reports [11]. When recurring faults are identified, their root causes should be discovered, hence, root cause detection stage. Several methods have been proposed for tracking down the root causes. Among them using fishbone diagrams

[21] and causal maps [22] are common. As soon as, one identifies recurring faults and their root causes, corrective measures could be undertaken. Therefore, in the last stage of RCA corrective actions are devised and undertaken to address the root causes of recurring faults. Unfortunately, not much is known about innovating corrective measures [15]. Previous literature often cites brainstorming, brainwriting, interviews, and focus group meetings [12], [14], [15] as approaches for innovating corrective measures.

Majority of RCA methods proposed in the literature rely on statistical analysis of fault reports in order to identify recurring faults. Fault reports, themselves, are formalized via a fault classification scheme. Reliance of the majority of RCA methods on fault reports makes them vulnerable to fault reporting mechanisms of organizations. Fault reports that are collected in organizations usually have comprehensibility and inaccuracy issues [23]. Mohagheghi et al. [16] have identified a number of problems in fault reporting processes. They reported ambiguous problem report fields as a source of confusion for developers. Definitions and terms might mean different things to different groups of stakeholders [16]. Lack of attention to product releases, changes in report fields between releases, coarse-grained information in reports, and different report formats and reporting tools are other issues that these researchers witnessed in fault reporting practices of organizations [16]. Furthermore, in practice, fault reports are usually collected just for fault removal and unfortunately are not further analyzed to gain process improvement insights [23], particularly in smaller organizations [24]. The considerable amount of upfront investment needed for collecting fault reports has made others to argue that RCA methods relying on fault data are inappropriate for SMEs [15]. Non-immediate visible gains, required customization, change in people's routines [17], and impractical assumption of full knowledge of faults [25] are other issues associated with RCA. These issues may be seen by SMEs as impediments to conducting RCA.

RCA could support development of highly reliable software systems by preventing recurring faults from being introduced and from going undetected. However, considering the reactive nature of existing RCA methods, prevention might not occur after all. The underlying assumption in existing RCA methods is that a problem (recurring faults) already exists, root causes of which should be identified. This assumption reveals the reactive nature of RCA. On the other hand, our review of 18 studies in the literature on fault reporting and RCA[1] revealed further evidence that current RCA methods are reactive. Despite highlighting the significance of proactive rather than reactive prevention of faults [13][26], most studies conducted in the literature are conducted after product release (retrospective) and fall short in providing insights for in-process improvements (See Table 1). Furthermore, closer inspection of the recommended time for conducting RCA (after each phase, after each iteration and in exceptional cases) suggests that results of RCA are only useful for later phases or iterations of development. Table 1 suggests that RCA could prevent faults from being introduced and from going undetected in future phases, iterations or projects. Benefits of RCA, therefore, seem not to have a bearing on the current phase, iteration or project.

[1] RCA is also used in project management and for examining project failures. We have not included studies that use RCA for such purposes here.

Table 1 RCA approaches and timing

Method recommended	**Method demonstrated**	**Timing**	**Source**
In-process	In-process	No time recommended	[27]
		After each phase	[28]
		After each iteration	[29][26]
	Retrospective	No time recommended	[11], [19], [30]
	Retrospective and In-process	No time recommended	[15]
	NA	Right after each phase or in exceptional cases	[14]
Retrospective	Retrospective	NA	[31]
Retrospective and In-process	NA	No time recommended	[13]
NA	In-process	No time recommended	[32]
	Retrospective	NA	[33], [34], [35], [36], [24]
	Retrospective and In-process	No time recommended	[37]

Considering their reactive nature and reliance on fault data, it is arguable that benefits of existing RCA methods could be complemented with a method that looks forward to prevent faults proactively and without reliance on fault data in a lightweight manner. Such a method should also provide a systematic mechanism for innovating corrective actions.

In software development research and practice, proactive improvement of software quality is often sought by means of applying Software Process Improvement (SPI). Oftentimes, SPIs have a continuous and cyclic nature that is perhaps best epitomized in Shewhart–Deming's plan-do-check-act (PDCA) paradigm [38]. Petterson et al. [38] characterized SPIs into two groups: (1) prescriptive SPIs such as CMMI, and ISO/IEC 15504 that take a top-down approach suggesting a number of best practices and (2) inductive ones such as iFlap [38] that take a bottom-up approach suggesting improvements based on current state of affairs in a software organization. Experiences of applying prescriptive methods such as CMMI and evidence of their effectiveness in fault prevention has been reported in the literature [39]–[42]. RCA methods are often suggested as an improvement opportunity in prescriptive SPIs. For instance, one of the key process areas of the CMMI level 5 is 'Causal Analysis and Resolution'. RCA methods are, therefore, part of prescriptive SPIs and as such have a different scope and application. Recently, however in the light of increasing necessity for SPIs in SMEs [43], and in order to tackle the cumbersome nature of SPIs such as CMMI [38], lightweight inductive approaches have been proposed. SPIs such as iFlap [38], COMPETISOFT [44], ASPE-MSC [45] and FLEX-RCA [46], to name a few, represent this group. The goal of such SPIs is to provide a lightweight solution particularly for SMEs to increase software quality by identifying and addressing problem areas in their development processes. With the exception of Flex-RCA, SPIs in this group, however, address problem areas mostly at a high level [46]. RCA methods, in contrast, focus on faults and provide concrete ways to identify root causes of recurring faults and suggest corrective measures. These corrective measures could deliver improvements to

development processes as well as instigating changes in staff roles, tools, and equipment during development.

In this study, focusing on faults, we set out to develop a proactive and lightweight RCA method and provide proof of concept. Like other RCA methods, our proposed method would focus on faults, however, the proactive nature of the method would hold resemblence to lightweight inductive SPIs as it aims at resolving problem areas in processes as well other problem areas.

3 Research Approach

Our study is carried out in three phases. The sources used for collecting data are published scientific studies and semi-structured interviewees with engineers active in safety-critical and business-critical domains (see Table 2). The underlying research strategy in this study in Design Science Research Methodology [47].

In phase one, we set out to understand erratic human behavior as one of the main sources of fault introduction and non-detection. In this phase, 10 experts from six international, privately-owned companies operating in safety-critical and business-critical domains were interviewed (Interviews 1-10 in Table 2). These companies provided engineering services for their customers in their respective domains and ranged from small to large size. The interviews were conducted by two of the authors independent of each other. At the end of this phase, we find that mismatches between development practices and development context could signal erratic behaviors.

In the second phase, based on this finding, we develop the PEBA method as a proactive complement for existing RCA methods. PEBA makes use of the taxonomy of contextual factors affecting fault prevention and fault removal. This taxonomy is also developed in this phase. Development of the taxonomy was done using directed qualitative content analysis [48] on 142 studies. These 142 studies were analyzed after carrying out an extensive literature search on topics associated with software reliability, fault prevention, fault removal and RCA. The literature search process is detailed in Appendix F. The complete list of reviewed articles is provided as supplementary material. For details of taxonomy development, please refer to section 5.1 and Figure 1.

The third phase is proof of concept. First, we demonstrate the use of PEBA and later we evaluate the method. For demonstration, we show the use of PEBA in two ongoing small and high impact projects in a company active in avionics domain. Two interviews (Interviews 11 and 12) were conducted with representatives of these two projects in this phase. Evaluation is done by interviewing two quality assurance experts in two companies providing software services to energy and healthcare suppliers. Interview guides are provided in Appendix G.

Table 2 Interviews

ID	Role(s)	Experience (at the time)	Domain	Company
1	Team leader	9-10 years	Avionics	1
2	Head of department	22 years	Avionics	1
3	Team leader & software process owner	8-9 years	Automotive	2
4	Testing engineer	11 years	Healthcare	3
5	Solutions engineer	5 years	Avionics	4
6	Team leader	5 years	Telecommunication	5
7	Head of development	10 years	Healthcare &	6

			Telecommunication	
8	Software developer	9 years	Healthcare & Telecommunication	6
9	Senior software architect	NA	Healthcare & Telecommunication	6
10	Software developer	over 14 years	Healthcare & Telecommunication	6
11	Software engineer	over six years	Avionics	1
12	System engineer	over 20 years	Avionics	1
13	Lead quality assurance engineer	15-16 years	Energy	7
14	Senior quality assurance engineer	13 years	Healthcare	8

4 Phase 1: Erratic behavior

Looking at the studies that provided categories of root causes (Table 3), it becomes clear that individuals are the main actors who deliver faults; meaning that they can either introduce faults or they might fail in detecting and removing faults. The predominant role of human behavior in delivering faults suggests that by preventing erratic behaviors, one of the main avenues to fault introduction and undetection could be blocked. Therefore, we probed ten professionals (Interviews 1-10) about quality practices in their respective organizations and the difficulties they face to understand erratic behaviors.

Table 3 Root cause categories

Developed artifact	**Root cause categories**	**Source**
Taxonomy of software error causes	Consistency, Completeness, Communication, Clerical	[49]
Root cause scheme	Application Errors, Problem-Solution Errors, Semantics Error, Syntax Error, Environment Errors, Information Management Errors, Clerical Errors	[31]
Classes of root cause	Phase-related, Human-related, Project-related, Review-related	[19]
Most cited cause-categories in the literature	Tools, Input, People, Methods	[14]
Requirements common causes	Noncompliant Process, Lack of Understanding, Human Error	[34]
Requirement error taxonomy	People Errors, Process Errors, Documentation Errors	[50]
Root cause taxonomy for software defects	Human Error, Process Error, Tool Problems, Task Problems	[51]
Categories of error causes	Communication Failure, Oversight, Education, Transcription Error,	[20]
Error causes	Programmer Error, Language Misunderstood, Previous Fix, Communication Failure, Spec unclear, Clerical, Programming Language Bug, Specification Changed, Other, Unknown	[52]
Levels of programming error causes	Technological, Organizational, Historic, Group Dynamic, Individual, Inexplicable Causes	[53]

We used inductive thematic analysis [54] for capturing themes (i.e. important patterns related to a phenomenon) in interviews. We started analysis by highlighting parts of the text that indicated problematic or fault-prone exercises. Following this step, we looked for possible patterns within the text in a cyclic and iterative process which included revisiting our analysis multiple times. Doing this, three recurring themes emerged. Interviewees referred to (1) a development practice, (2) a contextual factor and (3) a mismatch between these when they addressed system quality or problems in development. For example, multiple interviewees explained a mismatch between time and resources and the practice of code reviewing. Such a mismatch could result in abandonment of reviews in favor of catching deadlines and eventually to ineffective fault removal. Another interviewee signaled a mismatch between communication mechanisms, culture and the international nature of the development team. He lamented that due to cultural differences in an international team, ideas are not challenged or communicated properly. Later on in the interview, he elaborated that because of this mismatch reviews of system documents might not take place. Mismatch between tool support and different development practices were commonly observed as well. Table 4 shows the mismatches referred to in interviews. Each row in the table refers to a mismatch (theme 3) between a development practice (theme 1) and a contextual factor (theme 2). The last column in the table indicates the interview in which each mismatch was signaled. Corresponding interview text from which these themes emerged could be found in Appendix D in order of the appearance in Table 4.

Table 4: Recurring themes and pattern in interview analysis

Theme 1 (practice)	**Theme 2 (context)**	**Interview**
Code review	Time and resources	1
Team communication channel of choice (Scrum's daily standup meetings in this case)	Organizational structure allowing a person to work in three projects	1
Audit practices	Schedule	1
Ideal testing practices	Project size	1
Developer's background and way of working	Task at hand	1
Planning practices	Tool support	2
Compliance with defined rules and practices	Organizational inertia	2
Senior management compliance	Defined practices	2
Quality management practices	Knowledge of standards	2
Requirement engineering practice	External stakeholders	2
Code review	Time	3
Consistent application of the process	Having defined processes	3
Communication mechanisms	Culture and the international nature of the development team	3
Iterative software development practice	Involvement of external stakeholders	3
Information management practices	Tool support	3
Traceability practices	Tool support	3
Manual testing	Number of code reviews per reviewer	4
Developers' coding	Tool support	4
Compliance coding rules	Quick development	4
Testing practices	Time	5
Compliance with guidelines	Legacy code	5
Refactoring practices	Evolving code	5

Transparent communication between team members (i.e during standup meetings)	Work pressure	6
Fault reporting practices	Tool support	6
Commenting practices	Market share of a product, and its complexity	6
Testing practices	Time	7
Testing practices	Budget	7
Testing and documentation practices	Quick delivery	8
Coding standard practiced in the project	Project members' coding styles	8
Use of (new) technologies	Product's projected software lifecycle	8
Coding standards practiced in the project	Number of tools (technologies) in a project	9
Code review	Schedule	9
Integration testing	Rapidly changing technology	10
Documentation practices	Staff changes	10
Compliance with development guidelines	Multiple contractors	10
Documentation practices	Rapidly changing technology	10

The important role of mismatches between development context and development practices have been touched upon by scholars as well. Fenton and Neil [55] claimed that while the mismatch between design effort and problem complexity leads to introduction of faults, the mismatch between design size and testing effort leads to ineffective detection of faults. Design effort and testing effort are development practices that are asked to be matched to design size, and problem complexity that are factors of the context. Similarly, the mismatch between design effort and functionality was argued by Avižienis et al. [2] as one of the prime causes of development failures. It is worth noting that Endres [53] considered a root cause of a programming fault to be a mismatch between problem difficulty and adequate practices applied. Furthermore, Leszak et al. [19] reported that mismatch between the skill-level needed in a project and individuals' skills can lead to introduction of faults.

Therefore, based on the analysis of responses from the interviews, and previous scholarly observations, there exists promising evidence that mismatches between development practices and context could result in erratic behavior. Erratic behaviors are prone to occur if circumstances to which they are vulnerable emerge. Therefore, erratic behaviors could be prevented by resolving mismatches between development practices and context. Consequently, by identifying mismatches, one could predict potential for erratic behaviors. In other words, mismatches can signal potential for erratic behaviors.

If this holds, it follows that erratic behaviors can be prevented, simply, by tailoring the development method so that the development practices fit the context. The idea of matching the software development method to the needs of the context is well-established in the research community [56]–[58]. In practice, there is indeed very little chance that a method is fully adopted and development methods are almost always subject to tailoring [59]. Following this line of reasoning, a group of experts have argued that there is no universally applicable software development method which suits all kinds of projects [60], [61]. Available methods must be tailored according to characteristics of software development context and needs of development teams [62], [63]. Data from the interviews confirmed the potential of tailoring as a corrective action.

> *So usually there is request for performance and criticality analysis of the system. This means, in this space standards, four levels of criticality that depends on the*

consequences of the failure of the systems or parts of the system. So once you perform this analysis and decide what is your criticality level, then, it drives what practices you have to follow per standard. You tailor the standard. (Interview 1)

In a way, it is perfectly normal, to have a difference, between what the book says, the standard says, and the actual practice. It's even part of the standards, or the methods to accept this idea of tailorage. The ECS standards have somewhere, it tells, you should tailor this standard. The […] project management thing, people think often that it's a huge bureaucratic thing. No, one of the key principal of the method tells, tailoring to meet environment. (Interview 2)

Therefore, a method could be developed that relies on identifying mismatches between development practices and context for predicting potential erratic behaviors. Consequently, resolving such mismatches could provide a means to systematically innovate corrective actions.

5 Phase 2: Taxonomy of contextual factors and PEBA

Based on the findings in the first phase of this research endeavor, we propose a method for proactively identifying and resolving erratic behaviors as the main avenue to fault delivery. The proposed method is PEBA, a proactive method that relies on the knowledge of individuals from the development arena for finding mismatches between development practices and context. In order to get a strong foothold for identifying mismatches, PEBA takes advantage of a taxonomy classifying contextual factors affecting fault prevention and detection. The taxonomy is an analyst's guiding light to finding mismatches between development practices and context.

5.1 Taxonomy of contextual factors

Development of the taxonomy was done using directed qualitative content analysis [48] on 142 studies. These studies were analyzed after carrying out an extensive literature search on topics associated with software reliability, fault prevention, fault removal and RCA. More information on the literature search is provided in Appendix F. Qualitative content analysis[2] is used to understand or explain a phenomenon through a systematic process of coding and identifying patterned regularities in text [48]. In directed qualitative content analysis previous research findings or theory is used to initialize a set of predetermined categories [48]. In order to define the initial set of code categories for developing taxonomy of contextual factors, it was found necessary to determine the best way a development context could be understood. According to Sjøberg et al. [64], in a typical software engineering situation "an actor applies technologies to perform certain activities on an (existing or planned) software system". From this statement, four key elements of a typical software engineering situation are understood to be (1) actor, (2) technology, (3) activity and (4) software system. Additionally, the development context can be described from different perspectives. We examined the context from (1) region, (2) organization, (3) project and (4) team perspectives. To summarize, the four perspectives of context and the four key elements of software engineering were considered two dimensions by which the context could be analyzed. For coding, a factor

[2] Qualitative content analysis (Hsien, & Shannon, 2005) is different from content analysis that is normally described as quantitative analysis of qualitative data

was defined as any phenomena, stimulant or circumstance that can be characterized as part of the context. Figure 1 depicts the taxonomy development process.

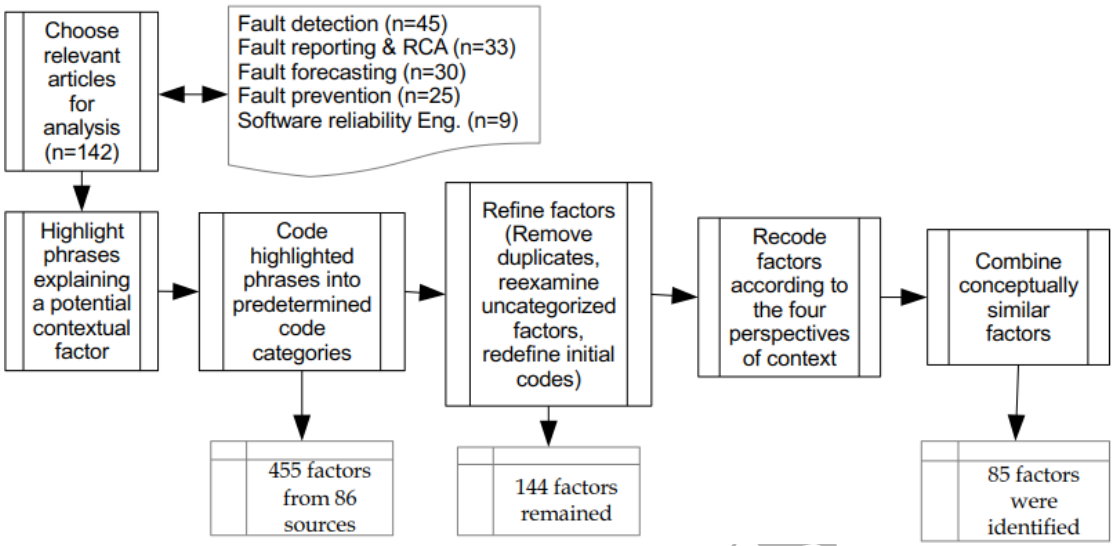

Figure 1 Taxonomy development process

After comparison and analysis of all factors, including the uncategorized ones, the key elements dimension was extended to human, environment, activity and artifact. Environment factors, as the name implies, refer to phenomena or stimulants in the surroundings of the people involved, the practices and deliverables. Factors related to high-level strategies and supporting technologies are included as environment factors. Human factors are those relating to individual's characteristics, behaviors, duties and their interactions with other individuals. The activity factors characterize the context in terms of the practices carried out and the processes followed to develop a product. It is important to emphasize that these factors do not refer to technicality of activities and how they are done, rather the existence and quality of activities that are known to affect the development practices. Finally, artifact factors address characteristics of any deliverable produced during development. The artifact could be a document that is the outcome of requirement analysis, or design. It could be the source code or the whole software system.

Overall, 85 factors were identified and taxonomy of contextual factors was developed. Figure 2 depicts the taxonomy and its structure. On the innermost circle, the perspectives of the context, and on the second layer, the key elements of context are visible. Some contextual factors are presented as examples in the outermost circle, however, due to space limitations the complete taxonomy of contextual factors is presented in Appendix A. The taxonomy will aid an analyst in identifying mismatches between development practices and context.

Figure 2 Taxonomy of contextual factors

5.2 PEBA method

PEBA is a proactive method, complemenatry to the existing RCA methods that are reactive in nature. PEBA does not rely on fault data. Based on the finding that mismatches between context and development practices can signal erratic behavior, PEBA is developed comprising three steps:

(1) Context mapping
(2) Erratic behavior mapping
(3) Corrective action innovation

In the first step, the development context is mapped. This task can be completed using the taxonomy of contextual factors developed in section 5.1. In the second step, mismatches between the context and development practices are identified and using causal maps [22] the relationship between mismatches and erratic behaviors are mapped. In the last step, corrective actions will be introduced. These corrective actions are derived from mismatches mapped in the previous step.

Two roles are defined for carrying out PEBA: the participant and the facilitator. The distinction between the participant and the facilitator roles is in the logical design of the method, and in reality the facilitator can take the role of the participant, as well, and vice versa. Such a design allows logical distribution of responsibilities between the participant and the facilitator while allowing the responsibilities to be assigned to individuals flexibly with respect to available project resources and structure. The role of

facilitator is similar to that of moderator in inspection [65]. The facilitator guides and controls the analysis. The participant, on the other hand, has the knowledge and know-how of the context and practices of development. The participant can be any of the stakeholders in the development. Project managers, quality managers, analysts, designers, developers, testers, reviewers, team leaders could all take the role of participant. The decision of who actually becomes a participant depends on the resources available and is up to the facilitator or management.

Acquiring a good understanding of the context is necessary for identification of mismatches. In step one of PEBA, the goal is to map the context to aid identification of mismatches later on. To this end, the facilitator selects the participants and outlines meetings. The meetings can be in the form of qualitative interviews, focus group meetings, or any other form according to available resources. The number of meetings is also a decision for the facilitator to make. If deemed sufficient for mapping the whole context, one meeting will wrap this step. Otherwise, further meetings are held. During the meeting(s), the facilitator and the participant(s) use the taxonomy of contextual factors, to map the context. As soon as the facilitator and/or the participants reach a consensus that the context is well understood, step one is complete.

The prerequisite of step two is a good understanding of the context and practices. So far, as a result of completing step one, the context has already been understood and wise selection of participants has ensured a good knowledge of development practices. The second step is carried out with the purpose, firstly, to identify mismatches between the context and practices and, secondly, to map the relationship between mismatches and erratic behaviors. To this end, the facilitator plans and holds meetings with the participant(s) similar to step one. During the meetings, the participant(s) and the facilitator, identify mismatches. Next, the relationship between mismatches and erratic behaviors will be mapped using a causal map [22] to potential erratic behaviors. The potential erratic behaviors, coupled with other mismatches, can lead to other erratic behaviors. Both the facilitator and the participant(s) can draw upon their experience and knowledge to map mismatches to erratic behaviors. The facilitator should promote discourse at this stage. The participant should convince the facilitator and other participants that an erratic behavior would occur due to a certain mismatch or combination of mismatches using reasonable arguments. If the participant manages to convince others of the possibility of an erratic behavior, the facilitator draws an arrow between the mismatch and that erratic behavior. This process will be iterated until the facilitator and participant(s) conclude that all mismatches are mapped to possible erratic behaviors.

The final step of PEBA is innovation of corrective actions. In PEBA, innovating corrective actions is done in a straightforward manner by deriving corrective actions from mismatches leading to erratic behaviors. Corrective actions should prevent the emergence of circumstances that give rise to erratic behaviors by resolving relevant mismatches. To this end, meetings are held by the facilitator, in accordance with the resources available. Yet again, the form and the number of the meetings are for the facilitator to decide. During the meetings, the participant(s) prioritizes erratic behaviors. This prioritization could be done in different manners, for instance, with respect to severity of the erratic behavior or by identifying bottlenecks in causal maps. The prioritization of erratic behaviors drives the agenda of the meeting and the innovation of corrective actions. After prirotization, the facilitator and the participants innovate corrective actions by proposing solutions that could resolve mismatches leading to an erratic behavior.

It is note-worthy, that the three steps need not be conducted in separate meetings. In one single qualitative interview or focus group meeting all the steps could be completed. The facilitator should plan according to the available resources, and participants' schedules. This makes PEBA a flexible method to be used in SMEs as well as large enterprises. As regards the appropriate time for carrying out PEBA, it is recommended to be conducted before each major stage of development or iteration.

6 Phase 3: Proof of concept

In this phase, we attend to proof of concept. Initially, we demonstrate the use of PEBA in two small and high-impact projects. Later, we evaluate the method by interviewing two independent quality assurance experts.

6.1 Demonstration

In Project 1, a system including both hardware and software was under development for the domain of avionics. This system was a replacement for an onboard system that was already operational on a well-known spacecraft at the time of carrying out this research, hence, a low level of tolerable risk and necessity for backward compatibility. Project 2 was also in the domain of avionics, however, in this project onboard software system prototypes were being developed to be used in future spacecrafts. In both cases, the software developer had close contact with the team leader and participated in meetings with their respective clients. At the time of conducting PEBA, Project 1 was still in the early stages, while Project 2 was in late stages of development and mainly validation activities was taking place. Therefore, proactive RCA was very much relevant to Project 1.

For mapping the development context, the facilitator held two online interviews with the main software developer in each project (Interviews 11-12). In this step, the researcher took the role of the facilitator and the main developer in charge of each project was the participant. The choice of online interviews over focus group meetings or face-to-face meetings was made based on the availability of participants and the geographical distance between the researcher and the participants.

The interviewee for Project 1 (Interview 11) was responsible for design and development of the software and selection of the hardware. The interviewee for Project 2 (Interview 12) had been working for the company for four years at the time of the interview and was the fourth engineer assigned to this project in three years. Project 1 was running for over two years. In order to map the context, the facilitator analyzed interview data using the taxonomy of contextual factors. At the end of this step, for each project a file was created, holding the key-value pairs of contextual factors (see Appendix B). It is important to note that since this was only proof of concept the analysis was limited to the project and team perspectives of the taxonomy and did not cover region and organization perspectives.

In step two of PEBA, erratic behaviors are mapped using directed graphs. Before the mapping starts, mismatches between practices and the context must be identified. The researcher took both facilitator and participant roles to find mismatches and map potential erratic behaviors. This is possible since the distinction between the roles are logical. Using the file containing key-value pairs of contextual factors and with the knowledge of practices discussed during interviews mismatches were identified. As soon as no further mismatches could be identified, mapping the erratic behaviors

started. Figure 3 and Figure 4 demonstrate the map of erratic behaviors and their relationship with mismatches and other erratic behaviors in the Project 1 and 2, respectively. Due to space limitations, here we only provide the description of mismatches and erratic behaviours of Project 1 and provide full description of mismatches and erratic behaviors for Project 2 in Appendix C.

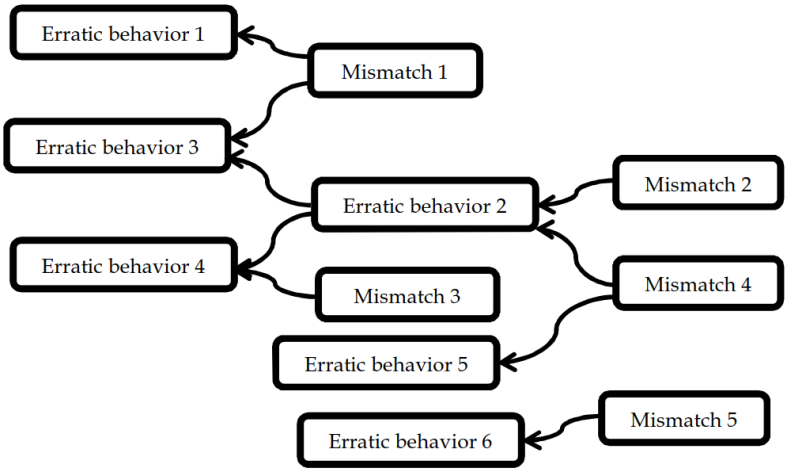

Figure 3: Project 1 causal map

The causal map of Project 1 (Figure 3) shows two separate paths. The first path depicted on the top shows the interconnections between 'Mismatches 1, 2, 3, 4' and 'Erratic behaviors 1, 2, 3, 4, 5'. The second path, visible on the bottom of the figure, shows the potential cause-effect relationship between 'Mismatch 5' and 'Erratic behavior 6'. Description of mismatches for Project 1 is provided in Table 5.

Table 5 Description of mismatches for Project 1

Mismatch #	**Between**	**Description**
Mismatch 1	Necessity of backward compatibility and concurrency of development	Part of the system needs to hold backward compatibility with scripts developed by experiment container developers. Since these developers are working in parallel to the team, no such script is provided to the development team. This could result in faults in the form of unsupported previous behavior.
Mismatch 2	Selection of fault detection practices and reliance on customer feedback	Reviews are performed in order to detect faults. However, late reviews held with the customer and reliance on such reviews for feedback, results in long time-span between updates to documents and late delivery.
Mismatch 3	Reliance on documentation and time-span between updates to documents	Considering that the developer relies heavily on documentation, long time-span between updates to documents might lead to fault introduction or nondetection
Mismatch 4	Availability and quality of documentation and tool support	The interviewee wished for better tool support for documentation. In case the tool is difficult to use and handle, considering that high quality and availability of documentation is necessary for this project and considering that the developer relies heavily on documentation, this inconsistency might lead to improper

		handling or update of the document and eventually a fault introduction or nondetection.
Mismatch 5	Availability of feedback with number of project members	No one inside the company is reviewing the works of the developer, this means that the point of departure is meetings and reviews with the customer. These meetings might be too little, too late.

Descriptions of erratic behaviors for the Project 1, presented in Figure 3, are provided in Table 6. The last column of the table includes the cause of each erratic behavior.

Table 6 Description of erratic behaviors for Project 1

Erratic behavior #	**Description**	**Cause**
Erratic behavior 1	Development without regards to requirements	Mismatch 1
Erratic behavior 2	Late update of documents	Mismatch 2 or Mismatch 4
Erratic behavior 3	Delayed delivery	Erratic behavior 2 or Mismatch 1
Erratic behavior 4	Development based on incorrect information	Erratic behavior 2 and Mismatch 3
Erratic behavior 5	Non-compliance with documentation procedure	Mismatch 4
Erratic behavior 6	Not noticing self-mistakes	Mismatch 5

The causal map of Project 2 is shown in Figure 4 below. Descriptions of mismatches and erratic behaviors for Project 2 are provided in Appendix C.

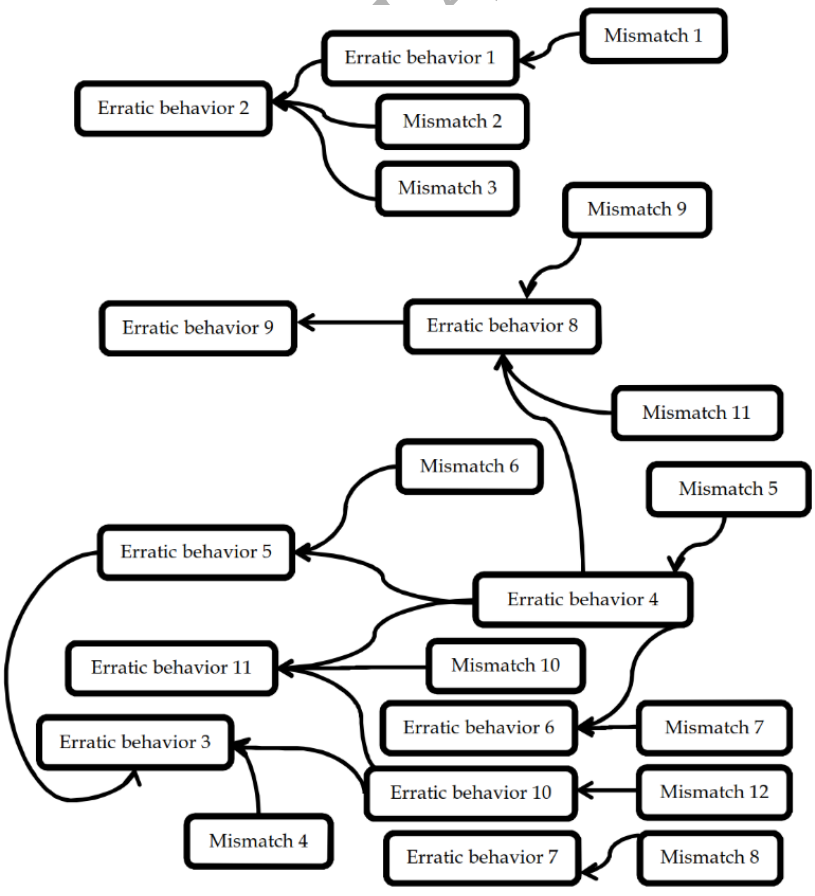

Figure 4: Project 2 causal map

The goal in the last step of PEBA is innovation of corrective actions. The corrective actions can be derived from mismatches in order to prevent erratic behaviors. In this step, corrective actions could be prioritized so that a sudden change of routines does not distress development. An example of possible corrective actions are innovated in this step for the purpose of demonstration. Consider 'Erratic behavior 2' in Project 1. This erratic behavior represents a bottleneck in Figure 3 as it could lead to 'Erratic behavior 1, 3 and 4'. Since either 'Mismatch 2' or 'Mismatch 4' could lead to 'Erratic behavior 2', solutions should address both of these mismatches. Corrective action for 'Mismatch 2' could be adding extra review sessions with internal reviewers. On the other hand, 'Mismatch 4' could be resolved by introducing new documentation tools or recruiting new members into the project to care for and handle the documentation.

6.2 Evaluation

Evaluation of PEBA is done by interviewing two quality assurance experts (Interview 13 and 14). Evaluators were provided with a description of the method and the findings presented in the demonstration phase. Corresponding interview text for evaluation could be found in Appendix E.

PEBA is designed to be proactive. This feature of PEBA is in stark distinction to the existing RCA methods in the literature that are reactive in nature. One of the evaluators stressed the complementary nature of PEBA and its necessity.

> *I think the method works as a complementary to the, sort of, rigorous mathematical models and I do think that [a conventional RCA model] also needs complementary methods, in the sense that, if you just focus on, sort of, these are the faults that we have identified and these are the weak areas because they have most faults; it's analysis that is not easy to do either (Interview 13)*

Resource-wise, one of the evaluators (Interview 13) was rather concerned about the resources needed, particularly, about scaling the method up from the two projects demonstrated to larger, more agile methods such as Scrum. She drew an analogy with retrospective meetings in Scrum and concluded that retrospective meetings in Scrum get to the problem quicker but if a project requires completeness then she would opt for PEBA. We believe this is a legitimate concern but it does not undermine the method. In agreement with the evaluator who mentioned completeness, we remind the reader that PEBA is developed as a credible method for development of highly reliable software systems and, therefore, it could be as lightweight or heavyweight in the scope of such projects. In essence, PEBA is not fixated on any number of interviews or even any particular data collection method. For that matter, one might decide on conducting a focus group meeting rather than several interviews. This is because we wish NOT to prescribe a one-size-fits-all solution. We believe the method is open for customization to the needs and resources of each organization or project. Secondly, we emphasize the proactive nature of the method as opposed to Scrum's retrospective meetings. When we mentioned this difference, the evaluator agreed and clarified that there is no such approach in Scrum.

Regarding needed resources, the second evaluator (Interview 14) noted that the proposed method is highly reliant on the competence of participants and the facilitator. In particular, the evaluator underlined the need for knowledge of the domain for achieving fruitful results. We acknowledge that the qualitative nature of PEBA makes correct selection of participants and facilitator essential to the success of its application.

When addressing effectiveness of the method, one of the evaluators (Interview 13) highlighted the value of the method in forcing individuals, and particularly management, into honesty. She stressed that knowing the mismatches and the potential erratic behavior that could follow them would help decision-makers in making conscious decisions. Furthermore, both evalutors recommended adding follow-ups to the method as an improvement that could highlight the effect of corrective actions. One evaluator (Interview 14) in particular made the recommendation stressing the necessity of reporting tangible results and the difficulty in doing so. According to him, participants, mostly developers in this case, might be unwilling to continue applying the method if the outcomes are not very tangible. He then continued that it is particularly difficult to see the results unless it is applied every few months to see if mismatches still exist or not. The comments of the evaluators address an important issue with regard to PEBA or any other preventive method for that matter. The proactive nature of such methods bars them from reporting "would-be" results such as number of faults that would have been detected if the method was not applied. However, PEBA is recommended to be conducted before each major stage of development or iteration. This would allow the team to assess mismatches identified in previous iterations and see whether they still exist or not as the evaluator suggested. In this manner, the next application of PEBA could act also as the follow-up to the previous application.

Evaluators made two recommendations. One evaluator (Interview 13) stated that, unlike risk analysis methods, it is not clear how one should prioritize tasks in PEBA. Prioritizing could be done in different manners for instance with respect to severity of the erratic behavior that follows a mismatch or by identifying bottlenecks in causal maps (see 'Erratic behavior 2' in Figure 3). In companies with fewer resources who cannot manage to resolve all issues, we suggest identifying and resolving bottlenecks since in this manner the route to several different erratic behaviors could be blocked. The other evaluator (Interview 14), on the other hand, suggested leaving some room for unforeseen problems that might occur later. Drawing from his experience in applying Scrum, he suggested that extra resources including man-power and time are considered for application of PEBA to account for such problems. While we find this suggestion intriguing, we also find it necessary to point out that SMEs might not have the resources to do so. The evaluator admitted that his experience comes from working in relatively large organizations and as such this recommendation might be useful for such settings.

The taxonomy of contextual factors developed in this research endeavor is instrumental to conducting PEBA. However, it is by no means exhaustive or finalized. Based on the knowledge and experience of the staff in the context, the taxonomy can be customized in a way to represent the context in the best possible way. One evaluator (Interview 13) refused to make a strong comment about the taxonomy on the grounds that it really needs to be evaluated in action. However, she did mention the large number of factors as an inhibiting factor for adoption of the method and recommended that different presets of factors be provided for different types of organizations or projects. This task however, is beyond what we could possibly do in this research effort. Our taxonomy development method simply does not allow for such an action, even though we find it useful. By contrast, the other evaluator (Interview 14) viewed the large number of contextual factors as an indication of the comprehensiveness of the taxonomy. For instance, he mentioned that one source of trouble in software projects is accounting for customer demands. He then added that several contextual factors in the taxonomy such as 'involvement of different stakeholders' could capture such demands.

7 Discussion, Limitations and Future research

We acknowledge that this research endeavor is very much informed by, and extends the works of Lanubile et al. [66], Lehtinen et al. [15], and Clarke and O'Connor [18]. The PEBA method, while benefiting from the merits of the ARCA method [15] is designed to be proactive. It does not assume the existence of problems. Similar to the "Error abstraction" method proposed by Lanubile et al. [66], in this study the main underlying theme is identification of common individual errors. However, while the error abstraction method relies on abstracting common errors from a set of already existing faults, in this research the goal is prediction of erratic behaviors. It is arguable that this difference between the two studies marks a difference in a reactive approach and a proactive approach. Another point of departure between the two is the scope of application. Lanubile et al. [66] focused solely on requirement faults; however, fault prevention should be extended to all stages of development. Identification of erratic behaviors in this study is done proactively for all development stages.

As regards mapping the context, Clarke and O'Connor [18] developed a reference framework of situational factors that can be used as a tool for defining software development processes or delivering improvements. The taxonomy of contextual factors developed in this research effort is similar to the situational factors reference framework of Clarke and O'Connor [18] in providing a tool for mapping the context of development. However, in doing so, the taxonomy of contextual factors, presented in this research, limits factors to those that can affect fault introduction and fault nondetection. Narrowing down the scope of the taxonomy improves its utilization for finding mismatches between the development context and practices. The reference framework of Clarke and O'Connor [18] has 8 factor classifications, 44 factors, and 172 sub-factors. The large scale of this framework compared to 85 factors and two dimensions presented in our taxonomy of contextual factors may render it inapplicable for the purposes of PEBA. Even though 85 contextual factors might still be too many to handle in practice, since the taxonomy is presented in two dimensions, practitioners can focus on the dimensions that they find most important. Furthermore, our taxonomy of contextual factor is flexible and could be customized based on the context in which PEBA is conducted.

Like other RCA methods, our proposed method focuses on faults. However, the proactive nature of the method resembles lightweight inductive SPIs such as iFlap [38] and FLEX-RCA [46] as it aims to resolve problem areas in processes as well as other problem areas. FLEX-RCA [46] is particularly of interest as, unlike other inductive SPIs, it provides detailed suggestions for practitioners rather than high-level advice and it relies on the traits of the participants. The main difference between FLEX-RCA [46] and PEBA, however, is in approaches for identifying problem areas themselves. FLEX-RCA [46] suffices to suggest methods such as brainstorming for identifying problems in process areas. Our proposed method goes one step further to suggest identifying mismatches between development context and development practices for that matter.

Additionally, PEBA holds similarities to risk management methods. Seeking a similar goal as risk management methods which aim to ensure the integrity of software development processes [67] and avoiding unsatisfactory or unwanted outcomes [68], PEBA enables development teams to prevent erratic behaviors by identifying corrective actions suitable for a context. Risk management frameworks often involve both reactive and proactive elements. While risk identification, and assessment comprise the proactive elements in risk management frameworks, contingency planning forms their

reactive nature. This dual character is observable in frameworks such as ProRisk [67], PRAM [69], Boehm's risk management framework [68] and KBRM [70]. PEBA overlaps risk management frameworks in their proactive elements and perhaps complements them. PEBA, by means of enabling identification of mismatches, could systematize the identification of risks in risk management frameworks, thereby, providing a concrete basis for risk mitigation and contingency planning. A systematized way for identification of mismatches is often missing in risk management frameworks. For instance, Boehm [68] suggests using checklists for risk identification. Interestingly, many of the risk factors identified by Boehm [68] and Schmidt et al. [71] such as personnel shortfalls and unrealistic schedules and budgets were identified as mismatches in this research.

7.1 Limitation

This study has a number of limitations. The reliability of the taxonomy of contextual factors developed in this study is subject to vulnerabilities. The directed qualitative content analysis conducted for developing the taxonomy might suffer from coder bias. In such a situation, a contextual factor might be missed or wrongly included. It is arguable, however, that the large number of factors coded alleviates the problem of missing a factor by increasing the chance of covering it in the analysis of other studies. The wrongly included factors are likely to have been dropped during the later steps of the development of the taxonomy.

Other limitations were faced in the phase 3. The scale and scope of the two projects for which the use of PEBA was demonstrated might not allow all the potential difficulties of the method to be surfaced. In Project 1, with a low level of tolerable risk, the focus was primarily on hardware rather than software. Project 2 was a prototype project for which the level of tolerable risk was fairly high. However, both project 1 and 2 were high-impact projects affecting space-exploration efforts and satisfied the goal of phase 3 which was proof of concept for a proactive preventive method that could be used in SMEs as well as large organizations.

Lastly, the data collection took place within projects in safety-critical and business-critical domains where software reliability is a sensitive topic. Therefore, in addition to the normal limitations attributed to interview data, some information might have been withheld from the researchers.

7.2 Future research

This research is mainly based on the finding that mismatches between context and development practices could signal potential erratic behaviors. There is a need for further assessment of this fresh reading of erratic behaviors. Consequently, we encourage future research to undertake this burden.

Furthermore, it is paramount to note that much research has been conducted on human behavior and there already exists substantial theories on human error [72], [73]. However, these theories are primarily addressing operator errors while we focus on developer errors. While Walia and Carver [50] explicitly differentiate between these, we refrain from drawing such distinction until further evidence is provided by future research and note that our finding does not challenge any of these theories. In fact, it corroborates with Reason's underlying assumption that human errors fall into recurring patterns [74].

Another area of exploration for future research is measurement of improvement in preventive methods such as PEBA. In this case, one of the evaluators suggested that revisiting previous mismatches in later stages of development could bring about the improvement potential of the method. However, still, a universal approach for preventive methods that could lay out their improvement potential could make it easier to convince senior management as well as developers into adoption.

8 Conclusion

In this research, the task of developing a proactive RCA method was undertaken. PEBA as the outcome of this endeavor is flexible and proactive, and relies on identification of mismatches between the development context and practices in order to predict erratic behaviors. Preventing these erratic behaviors could then prevent faults from being introduced or going undetected. Even though, development of a system that is completely free from faults is far from reality, this could make considerable contributions to the development of highly reliable systems.

REFERENCES

[1] A. Goel, "Software reliability models:Assumptions, limitations, and applicability," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 12, pp. 1411–1423, 1985.

[2] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[3] P. Bishop, "Does software have to be ultra reliable in safety critical systems?," *Int. Conf. Comput. Safety, Reliab. Secur.*, pp. 118–129, 2013.

[4] J. A. Børretzen, T. Stålhane, T. Lauritsen, and P. T. Myhrer, "Safety activities during early software project phases.," in *Norwegian Informatics Conference*, 2004.

[5] F. M. Favarò, D. W. Jackson, J. H. Saleh, and D. N. Mavris, "Software contributions to aircraft adverse events: Case studies and analyses of recurrent accident patterns and failure mechanisms," *Reliab. Eng. Syst. Saf.*, vol. 113, no. 1, pp. 131–142, 2013.

[6] N. G. Leveson, "Role of Software in Spacecraft Accidents," *J. Spacecr. Rockets*, vol. 41, no. 4, pp. 564–575, 2004.

[7] R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Trans. Softw. Eng.*, vol. 19, no. 1, pp. 3–12, 1993.

[8] B. Littlewood and L. Strigini, "'Validation of ultra-high dependability…' – 20 years on Bev," *Saf. Syst. Safety-Critical Syst. Club Newsl.*, vol. 20, no. 3, 2011.

[9] ECSS, "Space product assurance–Software dependability and safety ECSS-Q-HB-80-03A:2012." European Cooperation for Space Standardization, 2012.

[10] M. R. Lyu, "Software Reliability Engineering : A Roadmap," *Futur. Softw. Eng.*, pp. 153–170, 2007.

[11] R. Chillarege *et al.*, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 943–956, 1992.

[12] D. N. Card, "Learning from our mistakes with defect causal analysis," *IEEE Softw.*, vol. 15, no. 1, pp. 56–63, 1998.

[13] R. Grady, "Software failure analysis for high-return process improvement decisions," *Hewlett Packard J.*, no. August, pp. 1–12, 1996.

[14] M. Kalinowski, G. H. Travassos, and D. N. Card, "Towards a defect prevention based process improvement approach," *EUROMICRO 2008 - Proc. 34th EUROMICRO Conf. Softw. Eng. Adv. Appl. SEAA 2008*, pp. 199–206, 2008.

[15] T. O. A. Lehtinen, M. V. Mäntylä, and J. Vanhanen, "Development and evaluation of a lightweight root cause analysis method (ARCA method) - Field studies at four software companies," *Inf. Softw. Technol.*, vol. 53, no. 10, pp. 1045–1061, 2011.

[16] P. Mohagheghi, R. Conradi, and J. A. Børretzen, "Revisiting the problem of using problem reports for quality assessment," *Proc. 2006 Int. Work. Softw. Qual. - WoSQ '06*, pp. 45--50, 2006.

[17] G. Carrozza, R. Pietrantuono, and S. Russo, "Defect analysis in mission-critical software systems: a detailed investigation," *J. Softw. Evol. Process*, vol. 27, no. 1, pp. 22–49, 2015.

[18] P. Clarke and R. V. O'Connor, "The situational factors that affect the software development process: Towards a comprehensive reference framework," *Inf. Softw. Technol.*, vol. 54, no. 5, pp. 433–447, 2012.

[19] M. Leszak, D. E. Perry, and D. Stoll, "Classification and evaluation of defects in a project retrospective," *J. Syst. Softw.*, vol. 61, no. 3, pp. 173–187, 2002.

[20] R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski, "Experiences with Defect Prevention," *IBM Syst. J.*, vol. 29, no. 1, pp. 4–32, 1990.

[21] K. Ishikawa, *Guide to quality control: industrial engineering and technology.* Asian Productivity Organization., 1976.

[22] F. O. Bjørnson, A. I. Wang, and E. Arisholm, "Improving the effectiveness of root cause analysis in post mortem analysis: A controlled experiment," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 150–161, 2009.

[23] J. A. Børretzen and J. Dyre-Hansen, "Investigating the Software Fault Profile of Industrial Projects to Determine Process Improvement Areas: An Empirical Study," *Eur. Conf. Softw. Process Improv.*, pp. 212–223, 2007.

[24] A. Raninen, T. Toroi, H. Vainio, and J. J. Ahonen, "Defect Data Analysis as Input for Software Process Improvement," in *International Conference on Product Focused Software Process Improvement*, 2012, pp. 3–16.

[25] N. Mellegård, M. Staron, and F. Törner, "A light-weight defect classification scheme for embedded automotive software and its initial evaluation," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 261–270, 2012.

[26] M. Soylemez and A. Tarhan, "Challenges of software process and product quality improvement : catalyzing defect root-cause investigation by process enactment data analysis," *Softw. Qual. J.*, vol. 26, pp. 779–807, 2018.

[27] B. Freimut, C. Denger, and M. Ketterer, "An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management

Bernd Freimut," *Softw. Metrics, 2005. 11th IEEE Int. Symp.*, no. Metrics, pp. 10–19, 2005.

[28] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, and R. Chillarege, "A case study of software process improvement during development," *IEEE Trans. Softw. Eng.*, vol. 19, no. 12, pp. 1157–1170, 1993.

[29] P. Jalote and N. Agrawal, "Using defect analysis feedback for improving quality and productivity in iterative software development," *Proc. Inf. Sci. Commun. Technol.*, pp. 701–713, 2005.

[30] A. A. Shenvi, "Defect prevention with orthogonal defect classification," *Proc. 2nd India Softw. Eng. Conf.*, pp. 83–88, 2009.

[31] V. R. Basili and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," in *Proceedings of the 9th international conference on Software Engineering*, 1987, pp. 345–357.

[32] G. Y. Hong, M. Xie, and P. Shanmugan, "A Statistical Method for Controlling Software Defect Detection Process," *Comput. Ind. Eng.*, vol. 37, pp. 137–140, 1999.

[33] N. Bridge and C. Miller, "Orthogonal Defect Classification Using Defect Data to Improve Software Development Norm Bridge Motorola Corporate Software Center Motorola GSM Products Division Arlington Heights , Illinois," *Softw. Qual.*, vol. 3, no. 1, pp. 1–8, 1998.

[34] J. H. Hayes, I. Raphael, E. A. Holbrook, and D. M. Pruett, "A case history of International Space Station requirement faults," *11th IEEE Int. Conf. Eng. Complex Comput. Syst.*, 2006.

[35] R. Lutz and I. C. Mikulski, "Empirical analysis of safety-critical anomolies during operations," *IEEE Trans. Softw. Eng.*, vol. 30, no. 3, pp. 172–180, 2004.

[36] W. D. Yu, "A Software Fault Prevention Approach in Coding and Root Cause Analysis," *Bell Labs Tech. J.*, no. June, pp. 3–21, 1998.

[37] N. Li, Z. Li, and X. Sun, "Classification of software defect detected by black-box testing: An empirical study," *Proc. - 2010 2nd WRI World Congr. Softw. Eng. WCSE 2010*, vol. 2, pp. 234–240, 2010.

[38] F. Pettersson, M. Ivarsson, T. Gorschek, and P. Ohman, "A practitioner's guide to light weight software process assessment and improvement planning," *J. Syst. Softw.*, vol. 81, pp. 972–995, 2008.

[39] F. Huang, B. Liu, S. Wang, and Q. Li, "The impact of software process consistency on residual defects," *J. Softw. Evol. Process*, vol. 27, no. 9, pp. 625–646, 2015.

[40] M. S. Krishnan and M. I. Kellner, "Measuring Process Consistency : Implications for Reducing Software Defects," *IEEE Trans. Softw. Eng.*, vol. 25, no. 6, pp. 800–815, 1999.

[41] D. E. Harter, C. F. Kemerer, and S. A. Slaughter, "Does software process improvement reduce the severity of defects? A longitudinal field study.," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 810–827, 2012.

[42] M. Diaz and J. Sligo, "How software process improvement helped Motorola," *IEEE Softw.*, vol. 14, no. 5, pp. 75–82, 1997.

[43] F. J. Pino, F. Garcia, and M. Piattini, "Software process improvement in small and medium software enterprises: a systematic review," *Softw. Qual. J.*, vol. 16, pp. 237–261, 2008.

[44] F. J. Pino, O. Pedreira, F. García, M. Rodríguez, and M. Piattini, "Using Scrum to guide the execution of software process improvement in small organizations," *J. Syst. Softw.*, vol. 83, no. 10, pp. 1662–1677, 2010.

[45] C. G. von Wangenheim, S. Weber, J. C. Rossa Hauck, and G. Trentin, "Experiences on establishing software processes in small companies," *Inf. Softw. Technol.*, vol. 48, pp. 890–900, 2006.

[46] J. Pernstål, R. Feldt, T. Gorschek, and D. Floré, "FLEX-RCA : a lean-based method for root cause analysis in software process improvement," *Softw. Qual. J.*, pp. 1–40, 2018.

[47] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007.

[48] H. F. Hsieh and S. E. Shannon, "Three approaches to qualitative content analysis," *Qual. Health Res.*, vol. 15, no. 9, pp. 1277–1288, 2005.

[49] B. W. Boehm, R. K. Mcclean, and D. E. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *IEEE Trans. Softw. Eng.*, vol. SE-1, no. 1, pp. 125–133, 1975.

[50] G. S. Walia and J. C. Carver, *Using error abstraction and classification to improve requirement quality: Conclusions from a family of four empirical studies*, vol. 18, no. 4. 2013.

[51] F. Huang, B. Liu, and B. Huang, "A Taxonomy System to Identify Human Error Causes for Software Defects," in *18th International Conference on Reliability and Quality in Design (ISSAT)*, 2012.

[52] T. J. Ostrand and E. J. Weyuker, "Collecting and categorizing software error data in an industrial environment," *J. Syst. Softw.*, vol. 4, no. 4, pp. 289–300, 1984.

[53] A. Endres, "An analysis of errors and their causes in systems programs," *IEEE Trans. Softw. Eng.*, vol. SE-1, pp. 140–149, 1975.

[54] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qual. Res. Psychol.*, vol. 3, no. 2, pp. 77–101, 2006.

[55] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 675–689, 1999.

[56] R. D. Austin and L. Devin, "Weighing the benefits and costs of flexibility in making software: Toward a contingency theory of the determinants of development process design," *Inf. Syst. Res.*, vol. 20, no. 3, pp. 462–477, 2009.

[57] B. C. Hardgrave, R. L. Wilson, and K. Eastman, "Toward a Contingency Model for Selecting an Information System Prototyping Strategy," *J. Manag. Inf. Syst.*, vol. 16, no. 2, pp. 113–136, 1999.

[58] J. Iivari, "A methodology for IS development as organizational change: A pragmatic contingency approach," *Inf. Syst. Dev. Hum. Prog. Organ.*, pp. 197–217, 1989.

[59] B. Fitzgerald, N. L. Russo, and T. O'Kane, "An Empirical Study of System Development Method Tailoring in Practice," *Eur. Conf. Inf. Syst.*, no. 2000, pp. 187–194, 2000.

[60] D. Truex, R. L. Baskerville, and J. Travis, "Amethodical systems development: The deferred meaning of systems development methods," *Accounting, Manag. Inf. Technol.*, vol. 10, no. 1, pp. 53–79, 2000.

[61] J. Iivari and N. Iivari, "The relationship between organizational culture and the deployment of agile methods," *Inf. Softw. Technol.*, vol. 53, no. 5, pp. 509–520, 2011.

[62] K. Conboy and B. Fitzgerald, "Method and developer characteristics for effective agile method tailoring," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 1, pp. 1–30, 2010.

[63] B. W. Boehm, "Get ready for agile methods, with care," *Computer (Long. Beach. Calif).*, vol. 35, no. 1, pp. 64–69, 2002.

[64] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay, "Building Theories in Software Engineering," *Guid. to Adv. Empir. Softw. Eng.*, pp. 312–336, 2008.

[65] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: Software inspections after 25 years," *Softw. Test. Verif. Reliab.*, vol. 12, no. 3, pp. 133–154, 2002.

[66] F. Lanubile, F. Shull, and V. R. Basili, "Experimenting with error abstraction in requirements documents," *Softw. Metrics Symp. 1998. Metrics 1998. Proceedings. Fifth Int.*, pp. 114–121, 1998.

[67] G. G. Roy, "A Risk Management Framework for Software Engineering Practice," *Proc. Aust. Softw. Eng. Conf.*, pp. 60–67, 2004.

[68] B. W. Boehm, "Software risk management: principles and practices," *IEEE Softw.*, vol. 8, no. 1, pp. 32–41, 1991.

[69] C. Chapman, "Project risk analysis and management-- PRAM the generic process," *Int. J. Proj. Manag.*, vol. 15, no. 5, pp. 273–281, 1997.

[70] S. Alhawari, L. Karadsheh, A. Nehari, and E. Mansour, "Knowledge-Based Risk Management framework for Information Technology project," *Int. J. Inf. Manage.*, vol. 32, no. 1, pp. 50–65, 2012.

[71] R. Schmidt, K. Lyytinen, M. Keil, and P. Cule, "Identifying software project risks: An international Delphi study," *J. Manag. Inf. Syst.*, vol. 17, no. 4, pp. 5–36, 2001.

[72] J. Rasmussen, "Skills, Rules, and Knowledge; Signals, Signs, and Symbols, and Other Distinctions in Human Performance Models," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-13, no. 3, pp. 257–266, 1983.

[73] J. Reason, *Human error*. Cambridge University Press, 1990.

[74] J. Reason, "Human error: models and management," *BMJ Br. Med. J.*, vol. 320, no. March, pp. 4–6, 2000.

Appendix A: Taxonomy of Contextual Factors

	Environment factors	Human factors	Activity factors	Artifact factors
Region	• Time zone	• Culture (collective behavior and behavioral norms, differences in mental models)	NA	NA
Organization	• Business domain • Business strategy (e.g., improving quality, lowering cost) • Involvement of external parties • Organization size • Organizational structure (resulting in communication delays)	• Involvement of staff on several projects • Organizational culture (continual improvement, reactive/proactive thinking); • Trust in other project staff	• Availability of Reactive/Proactive processes • Maturity of processes (change in processes, data tracking and management practices) • Quality of analysis on inter-related projects (e.g., impact analysis, RCA, etc.) • Quality of fault reporting process • Quality of inter-project communication	NA
Project	• Budget and schedule • Degree of customer involvement • Education and training • Involvement of different stakeholders • Level of tolerable risk • Office ergonomics • Project size • Project structure (complexity of organization's projects: e.g., multi-project/single-project development) • Standards in place • Tool support	• Availability of dedicated testing staff • Commitment toward high-quality product development • Conflicting schedules of project staff • Defined responsibilities • Degree of attention to detail and priority of procedures • Degree of trust in other staff • Fear of data misuse • Frequency of change in staff members • Level of commitment to fault data collection • Staff knowledge, skill and experience • The number of project staff	• Alignment of testing and requirement analysis • Assignment and handling of priorities • Availability and usage of automated tests • Availability of definitions and guidelines (e.g. definition of testing stopping criteria; definition of milestones; terminologies commonly used, etc.) • Availability of feedback from other stakeholders • Concurrency of activities (e.g., coding and testing) • Coordination of testing activities • Fault fixing strategy (fixing faults in order of severity) • Degree of compliance with guidelines and standards • Development strategy (multiple release, open source, reuse, distributed and concurrent development) • Division of responsibilities between teams • Frequency of updates to documents • Independent fault detection • Interaction of developers with testing staff • Quality of fault and test case tracing • Quality of intra-project communication • Selection process of fault detection practices (test approach and strategy) • Iteration/sprint time span in incremental development • Synchronicity of communication • Traceability (Requirements to tests / faults to requirements);	• Application domain • Availability and Quality of documentation (e.g., requirements) • Backward compatibility • Criticality of subsystems • Fault classification scheme used for fault reporting • Fault profile • Design problem history (persistency and stability), • Expected lifetime of system • Fault proneness of modules • Modeling paradigm • Operational usage • Product complexity (clarity of interactions between subsystems) • Product size (e.g., LOC) • Programming language in-use • Quality of fault reports • Scope of system's possible behaviors • Source code evolvability • Testability • Volatility of requirements
Team	• Team size • Team's physical location (Distributed/co-located)	• Commitment to teamwork • Frequency of change in team members • Group synergy • Reviewers' collusion • Variation of team skills and experience • Tendency of teams toward production blocking and evaluation apprehension	NA	NA

Appendix B: contextual factors identified for two projects

Project 1 contextual factors		
Environment factors	Budget and schedule	Schedule is very tight
	Degree of customer involvement	High
	Education and training	None
	Involvement of different stakeholders	Sub-contractors and customers
	Level of tolerable risk	Low
	Office ergonomics	All members in one office
	Project size	Small
	Standards in place	Tailored ECSS E-40 standard
	Tool support	Redmine, Doors, Doxygen
Human factors	Availability of dedicated testing staff	No
	Degree of trust in other staff	High
	Frequency of change in staff members	No staff change
	The number of project staff	Three
Artifact factors	Application domain	Avionics - onboard flight system
	Availability and Quality of documentation	Everything in word docs
	Backward compatibility	Yes
	Fault classification scheme used for fault reporting	Provided by Redmine tool
	Expected lifetime of system	10 years
	Modeling paradigm	Not a model-driven development
	Operational usage	Known by operational scenarios
	Product complexity	A lot of interfaces and challenges of open source libraries
	Product size	Large
	Programming language used and its features	C, C++
	Scope of system's possible behaviors	Predictable by using a state machine
	Source code evolvability	Coding rules are defined, Doxygen documentation style
	Testability	No, no time for analysis
	Volatility of requirements	Volatile for new parts of the system
Activity factors	Availability of feedback	Non, reliance on customer feedback
	Concurrency of activities	Yes, concurrent with customer and sub-contractors, experiment container developers (customer) are working in parallel
	Development strategy	Heavy use of open source software and libraries
	Division of responsibilities between teams	No division. One person is responsible for all
	Independent fault detection	Yes, another team will test the system in the end but not at this stage
	Quality of intra-project communication	High, daily standup meeting, meetings within the project; with subcontractor and customer and phone Calls and emails
	Selection process of fault detection practices	Mainly oriented around customer requirements (reviews) but tests are designed in-house
	Synchronicity of communication	Synchronous with other members of the project, synchronous with subcontractors, bi-weekly meeting with customer,
	Traceability	Doors is used for manageability
Team contextual factors		
Environment factors	Team size	Small
	Team's physical location	Co-located

Project 2 contextual factors		
Environment factors	Degree of customer involvement	Regular meetings
	Education and training	None
	Involvement of different stakeholders	Minor support from other engineers
	Level of tolerable risk	High – because it is a prototype project
	Office ergonomics	Cubicles and ergonomics campaign
	Project size	Small
	Project structure	No other related projects
	Standards in place	Tailored ECSS E-40 standards
	Tool support	Redmine
Human factors	Availability of dedicated testing staff	No
	Degree of trust in other staff	High
	Frequency of change in staff members	Frequent
	Level of commitment to fault data collection	Time pressure can stop collection
	Staff knowledge, skill and experience	High
	The number of project staff	Two
Artifact factors	Application domain	Avionics
	Availability and Quality of documentation	High at the beginning but low at the end, might be problems because members have left
	Backward compatibility	NA
	Fault classification scheme used for fault reporting	Provided by Redmine tool
	Expected lifetime of system	NA
	Modeling paradigm	UML
	Operational usage	Prototype
	Product complexity	Low, subsystems and interactions are known
	Product size	Not large
	Source code evolvability	A matter of schedule, no reviews for this matter. But the customer has specific requirement for percentage of comments
	Testability	Not considered
	Volatility of requirements	Low
Activity factors	Assignment and handling of priorities	At the beginning of the project for bug fixes but a chance they would be ignored later
	Availability of definitions and guidelines	Available in certain cases like coding rules but no official procedure to review compliance
	Availability of feedback	No, customer reviews
	Coordination of testing activities	NA
	Division of responsibilities between teams	One person responsible for all tasks
	Frequency of updates to documents	Either immediately or next release
	Independent fault detection	No
	Interaction of developers with testing staff	NA
	Quality of intra-project communication	Good but still miscommunication is reported
	Selection process of fault detection practices	No defined process
	Synchronicity of communication	Synchronic with project leader, not synchronic with the customer
Team contextual factors		
Environment factors	Team size	One
	Team's physical location	Co-located

Appendix C: Mismatches and Erratic Behaviors of Project 2

Description of mismatches for project 2		
Mismatch #	**Between**	**Description**
Mismatch 1	Project schedule and developer tasks	If the schedule is tight and the developer has a lot of tasks to complete, the level of commitment to fault data collection falls
Mismatch 2	Level of commitment to fault data collection and the practice of prioritizing the fault fixes	It was stated by the interviewee that a mechanism for fault data collection existed but as the schedule becomes tighter, the level of commitment to fault data collection falls. Lack of commitment to fault data collection might primarily cause a problem when you consider that in the project fault fixes are prioritized. A fault that has not been reported might go unnoticed in planning and scheduling of fault fixes and subsequently slip through to the final product.
Mismatch 3	Level of commitment to fault data collection and staff changes	If faults data are not collected properly, considering that the project has seen several staff changes before, there is a chance for them going unnoticed.
Mismatch 4	Evolvability and frequency of changes in staff members	When members sacrifice commenting at the expense of catching deadlines there is a threat that if a staff change occurs, the next person will have difficulty understanding what was supposed to go on, what was supposed to be developed and etc. Misunderstanding of the works of previous developers can lead to introduction of faults in addition to a waste of valuable time.
Mismatch 5	Project schedule and documentation practices	Quality of documentation drops at the expense of catching deadlines.
Mismatch 6	Reliance on documentation, quality of documentation and frequency of staff changes	If quality of documentation drops at the expense of catching deadlines then reliance on documentation can introduce problems. The interviewee however claimed that he relies less on documentation in the latest phases. This does not solve the problem, however. If the documentation is not relied upon for development, then development becomes a matter of developer's experience and skills, considering the frequency of staff changes even if the current developer is highly skilled and experienced, the staff who are supposed to continue development in future or maintain and update the product in future might inadvertently introduce faults.
Mismatch 7	Degree of trust in other members and documentation quality	High trust in what previous members have done coupled with documentation quality that drops at the expense of deadlines, might inhibit critical analysis of documentation and result in faults slipping through to operation.
Mismatch 8	Availability of feedback with number of project members	Since there is only one person doing everything in this project, if that person does not receive constructive feedback, he is prone to not noticing his own mistakes. The interviewee admitted that this is not ideal. Even though the meetings with the customer can act as a feedback process, it might simply be too little too late.
Mismatch 9	Developer experience with DSDM and development method chosen	The interviewee stated that an agile development method called DSDM with a number of iterations were planned for the project in the beginning, he also admitted his lack of experience with this method. Had they actually stuck by their plans to develop using DSDM, such lack of experience with the chosen development method of the sole and main developer of the project could have led to ad-hoc development practices. However, the interviewee mentioned that they went through one V-cycle at the end.
Mismatch 10	Intention to reuse in future and availability of definitions and guidelines	Even though the interviewee expressed his lack of information whether this prototype project would continue, he did express that they intend to reuse several components in future. If this is the case, then lack of high quality documentation and non-evolvability of source code could lead to introduction of faults. Plus, definitions and guidelines would be necessary. As the interviewee mentioned they are not doing any extra effort.
Mismatch 11	Quality of documentation, reliance on documentation and timespan between updates	As the interviewee mentioned some inconsistencies in the documents goes unnoticed until they are reported by the customer, in such a case the inconsistencies are fixed in next stages, however, this lag coupled with non-reliance on documentation toward the final stages by the developer might come at a high price of developing using ad-hoc processes. Some issues might be forgotten or go unnoticed.
Mismatch 12	Commenting practices and project schedule	The interviewee mentioned that commenting might sometimes be sacrificed to catch deadlines. This could later lead to non-evolvability of the code. Which could in the long run lead to faults slipping through by misunderstanding in addition to waste of time.

Description of erratic behaviors for project 2		
Erratic behavior #	**Description**	**Cause**
Erratic behavior 1	Noncompliance with fault reporting procedures	Mismatch 1
Erratic behavior 2	Faults go unnoticed in planning and scheduling of fault fixes	Erratic behavior 1 and mismatch 2 and mismatch 3

Erratic behavior 3	Misunderstanding of the works of previous developers	(Mismatch12 and Mismatch 4) or Erratic behavior 5
Erratic behavior 4	Noncompliance with documentation procedures	Mismatch 5
Erratic behavior 5	Non-reliance on documentation for development	Erratic behavior 4 and mismatch 6
Erratic behavior 6	omission of critical analysis of documents	Mismatch 7 and Erratic behavior 4
Erratic behavior 7	Not noticing self-mistakes	Mismatch 8
Erratic behavior 8	Non-compliance with the development method and defined procedures	Mismatch 9 or (Erratic behavior 4 and mismatch 11)
Erratic behavior 9	Delayed delivery	Erratic behavior 8
Erratic behavior 10	Non-evolvability of the source code	Mismatch 12
Erratic behavior 11	Reuse of faulty components	Erratic behavior 4 and 10 and mismatch 10

APPENDIX D: Mismatches Implied in phase one interviews (Data for Table 4)

Implied possible mismatch between	**Interviewee response**
Mismatch between time and resources and the practice of code reviewing	We have envisaged to use code review in projects and we finally don't have time to do it; or resources. (Interview 1)
Mismatch between an organizational structure allowing a person to work in three projects could come at odds with a team communication channel of choice (Scrum's daily standup meetings in this case)	You cannot do daily [stand up] meetings […] when one person is working for three projects. (Interview 1)
Schedule, and the audit practices could lead to minimal unit testing. Lack of sufficient unit testing could lead to fault introduction and nondetection	I think it's more time pressure, let's say, for unit test which is probably the most useful, it requires a lot of maintenance and usually if you don't have someone behind that really sees that you invest some effort in doing unit testing, the result is that you end up testing for having the system working more or less, you don't care about finding all the defects, you say when problem appears in the future, I will solve this specific problem; I won't invest effort in developing a test suite that will double my maintenance work. (Interview 1)
Ideal testing practices and the project size	If you are developing a software, it is difficult to define tests that will discover problems because the problems you can think about, you have already put in there. But, also in that case, we didn't have the size scale to have two separate parts of organization [testers and developers]. (Interview 1)
Mismatch between the task and the background and ways of working of developers	It's a matter of also assigning, to each person working in the project, the activities that are more suited to the way of working, to the background. (Interview 1)
Mismatch between tool support and the existing planning practices	[…] those tools are awfully complicated. I mean, any planning tool is complicated, that's a problem of planning, it's very multiple dimensional […] In every planning, you have what we call a planning horizon, the duration, over which you can do [reliable] planning. The planning horizon here, as I understood it, is so short that it's best to naviguer à vue. So, personally I do not plan any more in the sense, I do some kind of planning but I'm not, using those tool[s] because I know I waste my time. (Interview 2)
Mismatch between organizational inertia and compliance with defined rules and practices. This mismatch could lead to non-compliance with the rules or application of an obsolete rule	And staff including myself who do not necessary know everything. Or, by the way, do not even concur, with the official rules. That's maybe another interesting thing. Because, the rules that are imposed to me. That was my best practice five years ago. In the meantime I'm, I mean, this is obsolete practice for me. But the inertia makes that I'm imposed now, sort of thing, that for me is, ridiculously… it [would] be a regression for me, to comply with the rules of the company. (Interview 2)
Defined practices and top management compliance	…there are people who are supposed to be the good example. Because they have invented, those things. Or they have been pushing for that. I mean, directors … That for example the role and responsibility of the project manager, includes this activity… But for a strange reason they don't care, they don't mind. And they find, some obscure good reason, (at least that) for not following the rules. Which by the way troubles everything afterwards […] they have pushed the employees to sign this document but they do not seem to, to follow themselves.(Interview 2)
Knowledge of standard and quality management practices	sometimes, people just by ignorance, do not follow, the good rules that are in the standards. Or they simply don't know enough the standard. Also the standard is not necessary, very educative. It's a statement, like a statement of law. The way they wrote that is really not educative at all. It's normal that people, to ignore good practice. It's even the same thing at the level of the QMS of the company. Everybody is supposed to have signed with his blood, a paper, yes I swear on the Holy Bible that I will follow the QMS of the company. But if I ask a few questions about, certain rules and procedure that are in there, they don't know. (Interview 2)
External stakeholders and requirement engineering practice	[external stakeholders] are not really capable to act as requirement engineer and, in the sense of producing the documents, at the end, everything is document. So producing user requirement document, or.. okay, software requirement is more the domain of the software solution but user requirement document, it is problematic. Even if I give hints, and concrete help for producing the document, telling you should follow this template, you should do this you should do that [breathes in deeply] it's getting messy. (Interview 2)
Mismatch between time and the practice of code reviewing	we would like to have a full review of our documentation and the code before we release. Currently this is not always the case, mainly due to time constraints (Interview 3)
Mismatch between having defined processes and consistent application of the process	On average project I would say, in most projects they really do their best to try and follow the process. But there is still no predictability, if you look to the different projects I would say. So, depending on the designer, he puts more focus on one part of the process and the other designer puts more focus on the other parts of the process. (Interview 3)
Mismatch between communication mechanisms, culture and the international nature of the development team	[…], due to the fact that [we are an] international company, [and the] development team is also international, it sometimes makes it more difficult to communicate between the system architect and the different subgroups. A lot of designers are like, ah, the system architect says I want to do it in this way and they just

	implement it, without thinking or without challenging him [...]. System architect has, this, like, standing, and the designers are below the system architect; and in certain cultures it's really hierarchical. I give this to you and no questions asked how we do it. So they don't challenge. (Interview 3)
Mismatch between external stakeholders and iterative software development practice	In general people, outside of the software center, they don't get or don't see the benefits of the iterative approach, or the early delivery. They don't understand. So most, like digital and, test, and, yeah, other groups, they work like this, big bang approach. So, yeah, I get my requirements, I do my work, here is my deliverable, (funnel). So they don't almost have any iterations whatsoever, they just work in one sequence […] you deliver(ed) a very complex system and then bugs start popping up from everywhere, then you have to dig in this complex system and to pinpoint where the bugs are coming from… (Interview 3)
Mismatch between tool support and information management practices which leads to information unavailability	[…] we have a revisioning system where normally all information should be at a certain location, but most of the time this information is at another location. So we have to try and search for this information and it's really, almost, unacceptable […] information is sometimes hard to find, and, you have to go and get it basically, and then it depends on the designer, whether he's introvert or extrovert […] (Interview 3) For the Office-based, Word-based, binary files, I would say, it's, hell. Because you always have to open it, not everybody has the same language settings, not everybody has the same Word document['s, revision], I don't know what, so, it fails always […] in our repository we cannot see the diff[erence]s between what is changed and what is not […] (Interview 3)
Mismatch between tool support and traceability practices which leads to reliance on individuals' discipline to keep track of changes.	[…] not everything is in [issue tracking software name]. And, most of it is basically in the documentation, and there, we don't have then traceability anymore and it really depends on the people itself, the discipline of the people that they track their references and that they keep track of all the status in the documents and it's really a hassle. (Interview 3)
Mismatch between number of code reviews per reviewer and manual testing	we review the code manually and in any case human makes mistake. Also the number of code reviews is high. For example at the moment I have three developers that I review their tests and someone else review my tests. But yeah I have to review these three developers' test and it is a lot of code and you might miss something when doing it manually. (Interview 4)
Mismatch between tools used and different coding styles for each developer	our problem in testing is that we use two languages both Python and Robot Framework. They match with each other but they have different coding styles especially Robot Framework doesn't have an extensive documentation. And maybe because of this, developers have difficulties (Interview 4)
Mismatch between quick development and compliance coding rules	[developers] are under lots of pressure to deliver quickly… they constantly deliver code it goes under review and like this… this might also affect that they say we don't spend time on following [coding] rules (Interview 4)
Mismatch between available time and testing practices	time is limiting factor so you start ignoring proper unit tests or you start doing things which directly save you the time (Interview 5)
Mismatch between code age and following guidelines	old things tend to not really change however if you do a related thing, if you change something on existing things, and there are no tests, for example there are no unit-tests, you should be writing them from scratch and all of them (even if you don't touch the class and plus there are some specific bugs and issues dedicated) […] but that is because we don't follow because in the past either us or that company had it before us did not really pay attention to that and violated the guideline (Interview 5)
Mismatch between evolving code and refactoring practices	I would say like, that, if there is a part which is evolving stably or there is a prediction be evolved in the future, you definitely should be refactoring it constantly because if you do not you will end up with very big sh*t (Interview 5)
Mismatch between transparent communication between team members (i.e during standup meetings) and pressure from work	When you are stuck in a project, when you are stuck in something and you have this guilt or this fear that you might not be able to do your project and you go under certain pressure that I have to finish it or, I don't know, I might get stuck in very big problems, transparency would help removing some part of this process by like talking about them everyday and by knowing that this is not just for you, but this is something that belongs to the team and will be solved in the team. So, for example, these daily standups are very important as soon as someone says that ok what did we do yesterday, what are you doing today and what is the problem? We will find someone. (Interview 6)
Mismatch between tool support and fault reporting practices	None of these existing tools can be altered enough that can be modified based on what you do in the company or what you do in your product. so you end up writing the description and it will be really dependent on the language in which people are expressing themselves and these can be sometimes very complicated (Interview 6)
Mismatch between market share of a product, it's complexity and commenting practices	When you have product that its market share is improving, it necessarily would increase in complexity as well. When you have more customers, more requests will come, it gets more complicated and definitely the number of people that know its whole point of view will decrease and that's why commenting is important because there will be people that have experience and understand, saw something in detail that the experts cannot see, experts that at different various points should give their comments, and it will grow from a single picture to a puzzle that is made out of many different people that their comments are a lot more valuable (Interview 6)
Mismatch between testing practices and time	sometimes small projects are more simple to develop and place them up and

constraints	running if we just skip some parts of that.. but my experience tells me that later that will be thrown back to us because I've.. I maybe, I may have taken some options to ignore Unit-Tests in development stage and I think well the code seems to be good, this will work, let's put it in production but when I get back to that code, to that application, I will see that I've missed some (Interview 7)
Mismatch between testing practices and budget	I believe budget is one these issues and sometimes bad project management… but budget essentially. When we sell a project, we sometimes forget to include the test part and testing is very important and it's hard to sell time to do tests, it's very hard to sell. Our customer will not understand why we are taking so long doing tests even [if] we tried to explain [to] them those test parts are important and will ensure software quality in the future but it's hard to sell those time and sometimes we just place that time on top [of] any other tasks on the project or project management process. We place that time under that task. (Interview 7)
Mismatch between quick delivery and testing and documentation practices	I think for instance I think testing and documentation they suffer the most They're the first things getting out of the way [to deliver quickly]. so testing and documentation always come at the end.. you have your part done you think.. Now you should test but at the end, you start cutting at the end! (Interview 8)
Mismatch between project members' coding styles and recommended coding standard practiced in the project	I've been in projects where this was a problem: you have four persons you have four styles of coding… Yeah so one of the reasons is also to keep it consistent. (Interview 8)
Mismatch between projected software lifecycle and use of technologies	if it's a software project that's going to live for a long time for ten to twenty years we have projects like that for twenty years or almost then and it's important to do the things in a way we know it's going to be sustainable for two next decades or something like that. it usually also means of not try new weird technologies that no one tested because we have to be sure that we're going to.. we don't want to maintenance of the project would cost as much as developing a new. (Interview 9)
Mismatch between number of tools (technologies) in each project and coding standards practiced in the project	The kind of coding standards that depends a bit on the technology so far we had much more technology than projects. So it's a little bit hard to say we are following rules because those rules, coding styles are just for one kind of technology and for another kind of technology don't apply anymore. (Interview 9)
Mismatch between project timeframe and code review practice	[Conducting code reviews] depends on.. well the time frame that we have available. So if we have a short timeframe, sometimes make a decision … and let these things go like this so we can keep the deadline or not. (Interview 9)
Mismatch between rapidly changing technology and integration testing	every six months, you have a major framework coming of age and then everything switches to that and we have like four applications and each one uses a different framework and different methods. Every time we need to make a new feature we need to update everything behind because the framework has changed and so it's not easy to have a clean roadmap about integration test and everything because it simply doesn't work. (Interview 10)
Mismatch between staff changes and documentation practices	right now the most senior guy which was from the [***] is leaving the company so we are going to be three new guys and the senior guy is going away and the last month is going to be to document everything that he knows but is not documented and there are lots and lots of processing and things to be documented. (Interview 10)
Mismatch between multiple contractors and compliance with development guidelines	the same issue we have now which is the lack of a technical person from the client to take the same decisions in every team like ok this team has these guidelines, you should have these guidelines, and if you don't want them we'll talk about it and find out why. But, right now you get a new company there or some person from other companies and they say no no in our company, we do it like this. And, since we are two different companies, we have been talking about it but there's no central decision. (Interview 10)
Mismatch between rapidly changing technology and the practice of slow documentation	[the documentation team], they are working on this for a year and they talked to us like one week ago for the first time. So it's a very slow movement because no one wants to document. we find the issues and then oops we should have documentation.. and there was another issue like server changes every API and the main application stop working and no one told the mobile application guys, which were us, that the API had changed. (Interview 10)

APPENDIX E: Data for Evaluation

Theme	Evaluator 1 (Interview 13)	Evaluator 2 (Interview 14)
complementary nature	I think the method works as a complementary to the, sort of, rigorous mathematical models and I do think that [a conventional RCA model] also needs complementary methods, in the sense that, if you just focus on, sort of, these are the faults that we have identified and these are the weak areas because they have most faults; it's analysis that is not easy to do either (Interview 6)	NA
resources needed	I do not know how I would adapt this to Scrum, I don't know how this would scale. Do you need to interview everybody, so on and so on? For example team retrospectives, I think, there, the nice thing about it is you get to the 'what do we do to improve', much quicker, because, here, time is spent on the analysis or what causes what, which is good, I think, in terms of completeness, but, even, how [our] software projects are, you cannot, usually, strive for completeness, you can strive for, let's do, at least, something. So, I think I would go for it in larger projects with big risks. (Interview 6) […]because the thing that looks forward is, in Scrum, is sprint planning but this is not a sprint planning method because it's all about the contents of the sprints. So this would, basically be, sort of, a tool for planning the sprints, in the sense that, these are the things we should take into account as we go forward. (Interview 6)	It requires that participants or the [facilitator] should have a really good knowledge of domain of the context. I think the person should know the context pretty well. It can be a bit.. if a person who doesn't have enough knowledge of the domain, if he's the one participant or the facilitator it might also lead to some not completely let's say fruitful results.
Effectiveness	I think it would at least force the project and the company into honesty […] the, sort of, value mismatches, […] so this would actually bring it out that disregarding something like rules or ideals or even strategy, initiative or whatever; that it would be a conscious decision rather than something that, would be, implicitly forced by something else. (Interview 6)	They [developers] would be open to the idea but they need to see the results at some point. If they, let's say they do this, they follow this method and after a while they still don't see a tangible result the might get a bit demotivated … sometimes it's very hard to see the results. If I see that the results have changed dramatically, it's obvious, but if I see the results are slightly better, it's hard to say that it's because of this method or because the team included someone more skillful or we are in a part of the project in which the tasks are easier. So, in a way, I would say it is difficult to measure it but it's also not impossible, if you follow this method, find mismatches etc. and then again do this after one month, two months and see if those still exist and how it affected the product
suggestions	[…]what I found that was missing was that, you get a nice graph, but how do you prioritize things (Interview 6)	So I think this method can have some sort of a space or place for some unseen problems that would come and you also need to perhaps allow some time and resource when you plan your development. In our team we always have scrum and we always put one man per day just for nothing. And it's always used.
Taxonomy	I looked at it and unfortunately, I think it is one of those things that I need to apply it myself in the method in order to give any feedback. I think it looked very comprehensive but is this sort of classification correct, I actually don't know, because it's one of those things that I can look at it and say 'ok, this makes sense' but to give any sort of 'oh, I think I would move this here and this here and I think this is a whole different category', I can't tell without actually running through the process myself. I think there, going forward to sort of lower the barrier for, you know, adoption for the method, I think then providing some sort of examples that perhaps would not be so large, broad, because it's like a lot of things, then, I think that would help in , you know, if you have this sort of project, you know, try this, with maybe these modifications or if you have this sort of a project and this sort of company try these and so on.	There's also always, there are many projects that, in our company also that the customer is very powerful. The customer is saying, ok, I want this and I want it by this time. They are the ones who are paying so maybe one factor here in that taxonomy table could be influence of customer. It affects the deadline it affects the features of the program that you are giving them. in our field for example many times the customer we just did something, we provided something, we have a tool, and our tool is customized based on the…. But some of our customers, who are big and who have been our customer for ten years or something, they sometimes want some features and those features from a development point of view, it's a bit of a struggle to make those features and to tell them but they are the one that [pay], our company exists because of them, they are the ones that pay, that's a factor that also affects the deadline and resources and everything. I see you have this 'involvement of different stakeholders' and that can be related to the customer.

APPENDIX F: Literature search

Development of the taxonomy was done using directed qualitative content analysis [1] on 142 studies. To this end, an extensive literature search was conducted on topics associated with software reliability, fault prevention and fault removal. The literature search was conducted using snowballing [2]–[4]. Figure 1 shows the process using which the literature search was conducted. The initial search was conducted using the keywords 'software reliability, 'fault prevention', and 'software reliability engineering'. As Figure 1 indicates, further keywords were included later on as the literature search progressed. These keywords were searched on the fly and they were not recorded.

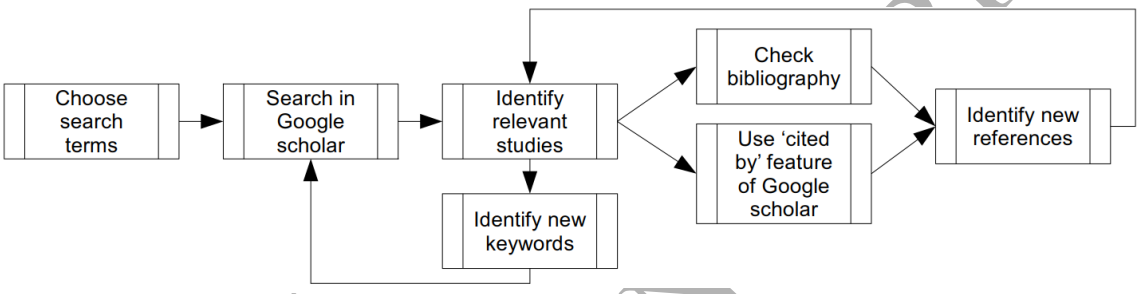

Figure 1 Literature search process

For inclusion of a study, first the title was investigated, if the title revealed new or relevant information regarding software reliability, fault prevention, fault detection, and RCA, that study was selected for abstract review. If the same conditions proved right for the abstract then the study was selected for full review. The stopping rule for material extraction was increasing frequency of repeating and irrelevant entries. However, later on a calendar date constraint was also set to stop the search. The literature search was concluded in 2015. Please note that RCA is also used in project management and for examining project failures. We have not included studies that use RCA for such purposes in this literature search. Overall, the literature search revealed 179 studies. However, before the review process for developing taxonomy of contextual factors started, these 179 studies were categorized based on their topic area. This step was similar to conducting a systematic mapping study [5]. After this categorization, studies in a number of categories were removed, as they were deemed irrelevant. The excluded groups consisted of studies on safety, maintenance and agile methods. At the end, 142 studies were reviewed for taxonomy development. It is important to note that results reported in Table 1 and Table 3 were also based on the studies revealed in the literature search. However, the literature search for these tables was not limited to a calendar deadline and therefore these tables include a few studies that were analyzed for taxonomy development.

[1] H. F. Hsieh and S. E. Shannon, "Three approaches to qualitative content analysis," *Qual. Health Res.*, vol. 15, no. 9, pp. 1277–1288, 2005.

[2] S. Jalali and C. Wohlin, "Systematic Literature Studies : Database Searches vs . Backward Snowballing," *Proc. ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas.*, pp. 29–38, 2012.

[3] C. Wohlin, "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering," *Proc. 18th Int. Conf. Eval. Assess. Softw. Eng.*, 2014.

[4] D. Badampudi, C. Wohlin, and K. Petersen, "Experiences from using snowballing and database searches in systematic literature studies," *Int. Conf. Eval. Assess. Softw. Eng.*, 2015.

[5] B. A. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," 2007.

APPENDIX G: Interview guides

Overall, we conducted 14 semi-structured interviews. Interview guides are provided below. Interviews were conducted by two of the researchers independent of each other before the analysis started. Please note that the interviews did not follow the linear progression that is presented in the below guides. The researchers allowed discussions during the interviews to resolve based on their own momentum. Therefore, some interview questions shown below may not have been asked at all. However, due to the importance of Interview 11 and 12 for demonstration purposes, interview guide for these were followed almost to the letter.

Interviews 1-10	**I. Background** 1. Could you please introduce yourself and let us know about your background and role at [company name]? 2. Could you please briefly introduce [company name]? **II. General information** 1. Could you please explain the development method currently being practiced at [company name]? 2. Are there any contractors involved in the development? For example in coding, testing, etc. 3. How are sub-projects and development of sub-components dealt with (Contractors, separate teams in a serial manner, teams working in parallel, distributed development)? How does this affect the development method? 4. How frequently are components reused at [company name] or are they at all? Do reused components go through a defect detection process too? 5. What standards are complied with? 6. Could you please explain the verification and validation practices at [company name]? 7. What mechanisms are in place at [company name] to help developer teams prevent, detect and remove faults? 8. What are the general practices at [company name] to make sure developers comply with practices and policies? 9. Are there practices in [company name] that promote and encourage developers to enhance their personal disciplines? (education, training, feedback on frequent mistakes) **III. Detailed questions** **Agile methods** 1. Does [company name] have any experience with or considered using agile methods and/or practices for development? For example, pair programming, Test-Driven Development, scrum sprints, daily stand-up meetings, etc. 2. If yes, how are such practices chosen and adopted? **Customer reliability requirements** 3. How does [COMPANY NAME] determine customer reliability requirements? For example, the customer asks for a certain reliability level, certification standards determine the necessary reliability, or by contacting the customer and extracting the requirements from discussions. 4. How is reliability measured at [COMPANY NAME]? 5. Is criticality analysis of functions and components performed at [COMPANY NAME]? Is there a difference between critical components and non-critical ones in terms of development and reliability practices? 6. Are the most frequently used functions of a system under development identified? **Fault data and changes** 7. How do you deal with changes during development at [COMPANY NAME]? Do you have mechanisms like a Change Control Board (CCB), use agile processes, or there is a customer proxy involved in the project? 8. Are defects, failures and changes traceable? What are the mechanisms used? What tools are used? How do you ensure that the traces are kept up to date (Is there a certain role that is responsible for keeping them up to date or each developer must make sure he/she submits the changes to a repository)? 9. How fault data is collected at [COMPANY NAME] or is it at all? What tools are used? 10. Could you please explain briefly what sort of information is collected for defects? Do developers fill in different forms at different stages of development? 11. How often does the structure of fault reports change or does it at all? 12. Who is responsible for defect detection (testers, inspectors, all project stakeholders)? 13. Who can report defects or failures (coders, designers, testers, sales persons, or anyone in the company)? Who has access to tools for fault reporting? 14. How is fault reporting enforced? What happens if a developer does not fill in fault reports or does not provide the necessary information? 15. In general how do you see developers' perception of fault reporting (as an overhead or important part of work)? 16. Does the quality of defect data filled in by developers allow analysis of data or is it ambiguous, too

	coarse-grained, etc.? 17. What kind of analysis is performed on fault/change data at [COMPANY NAME]? 18. Do you look for root causes of problems (frequent, severe, etc.) at [COMPANY NAME]? How? Do you perform Root Cause Analysis? 19. Do you deliver process improvements to prevent faults? How? **Defect detection** 20. How is testing performed (in-house testing department or testing team, independent contractors)? 21. Do you use theorem proving and/or model checking techniques for verification? Do you use any tools helping with that? 22. Is inspection performed at [COMPANY NAME] in order to detect defects? 23. Could you please explain a typical inspection meeting? 24. Who do the inspection teams consist of? 25. At what points during development an inspection meeting is held (after each milestone, each sprint, iteration, etc.)? 26. Is analysis of defect data used to guide defect detection? 27. Are test cases traceable? What tools are used? Who is responsible? 28. How is the testing strategy determined? 29. Are static code analysis tools used? **Developer communication** 30. What are the communication mechanisms between developers, used at [COMPANY NAME] (Official meeting, unofficial meetings, Scrum standup meetings)? What are the tools that enable such communication? 31. In particular what mechanisms exist in [COMPANY NAME] to allow testers and other developers (coders, designer, analysts, etc.) communicate? For example, how do the developers let testers know of changes? Are testers involved in early planning stages? 32. Is testability considered during requirements specification, design, and coding? **Other practices** 33. What are the commenting practices at [COMPANY NAME]? What if a developer does not comply with commenting policies or best practices? How do make sure comments are kept up to date? 34. Are there any coding standards defined for coders? How are they enforced? What if someone does not comply?
Interview 11-12	**I. Background** 1. Could you please introduce yourself and let us know about your background and role at [company name] and the project you are involved with? 2. Could you please briefly introduce [company name] and the project? 3. What is the application domain of the system under development? 3.1 How many people (approx.) are working on the project? 3.2 How complex is the system under development? (scope of system's possible behaviors large or small, interactions between system's sub-systems, etc.) 3.3 How large is the system under development? 3.4 What is the expected lifetime of the system? 3.5 Are there any external parties involved in the development? For example in coding, testing, etc. 3.6 Is independent defect detection performed in the project? 3.7 Contractors? 3.8 Is it a multiple release project or just one release at the end of the project? 3.9 Who/what is the user of the system being developed? 3.10 Is the operational usage known to developers including the frequency of usage? 3.11 Do you need to take backward compatibility in mind? 3.12 How does the customer get involved in the project? During, before and after. **II. General information** 4. Could you please explain the development method currently being practiced at the project? 4.1 Do you use agile methods and/or practices for development? For example, pair programming, Test-Driven Development, scrum sprints, daily stand-up meetings, etc. If yes, how are such practices chosen and adopted? 4.2 How frequently are components reused at [company name] or are they at all? 4.3 Do reused components go through a defect detection process too? 5. How are the teams managed (assigned responsibilities) in the project in which you are involved (division of responsibilities between teams, etc.)? 5.1 Is there a virtual development environment? Do you have virtual teams? 6. What precautions are taken to reduce the number of faults introduced? 6.1 Do you care for testability during development (all stages)? 6.2 Do you look for root causes of failures and faults? Do you perform Root Cause Analysis? Is there a defined feedback process (for example to let developers know what type of mistakes they have made and etc.)? 7. What are the defect detection practices used in the project? (testing strategies, testing techniques, type of reviews, people involved, , automatic scripts, etc.) How are they chosen? 7.1 How are test activities coordinated? 7.2 Are lower and upper bounds for defects detected during reviews? 8. What are the mechanisms to ensure high quality of documentation? 8.1 How much do you rely on documentation in the project? 8.2 How long does it take for a document (requirement, design, etc.) to be updated if there is any change? 8.3 How committed are project members to document? 8.4 What are the defect reporting mechanisms?

	8.5 How good are the defect reports in terms of quality? 8.6 How committed are project members to defect data collection? 9. What are the defect fixing mechanisms? 9.1 What information is relied on for fixing? 9.2 What is the defect fixing strategy (for example fixing low severity defects later and attend to high severity defects now)? 10. What are the general practices at [company name] to make sure developers comply with practices and policies? 10.1 Are there defined guidelines and procedures available to members of the project? 10.2 Are there project specific standards that you have to comply with? 11. Is this a critical project in terms of reliability? 11.1 What percentage of the components of the system is critical? 11.2 Is there a difference between the way you handle critical components and non-critical ones in terms of development practices including requirements analysis, design, defect detection, fault reporting, etc.? 12. What are the mechanisms that ensure information flow between requirements analysis and testing? 12.1 Are the defects detected traced back to test cases that detected them? **III. Detailed questions** 13. How volatile are the requirements? How do you deal with changes during development in this project? 13.1 Do you have mechanisms like a Change Control Board (CCB), league of experts or you use agile processes for this purpose? 13.2 Are the defects traced back to requirements? 14. What are the communication mechanisms (Face-2-face, email, a proprietary system, Official meeting, unofficial meetings, Scrum standup meetings)? 14.1 Is communication synchronic or is it deferred? 14.2 How friendly is the interaction between project members; specifically testing staff and developers? 14.3 How hard is it to organize a meeting in the project considering the busy schedules of parties involved? 15. What are the evolvability practices (commenting for code, coding standards, coding styles, design paradigm, etc.)? 15.1 How much do you rely on them, for example on code comments in the project? 15.2 Are there any coding standards defined for coders? 16. Is there a priority list or a similar mechanism to handle high priority tasks? 17. How often do the project members change? What about other staff members who have an influence on the project? 18. How much do you rely on and trust other project members? Is there a fear of data misuse by other members among project staff? 19. Please describe the office ergonomics.
Interview 13-14	1. Would you consider adopting this method for software development projects? Why? 2. How do you think developers would react to this method if you adopt it? What about managers or other quality assurance experts? 3. What do you think are the weaknesses and strengths of PRORCA. 4. What is your opinion about the idea that mismatches between development context and practices lead to erratic behaviors? Do you find proactive identification of erratic behaviors valuable (the way PRORCA recommends)? 5. What do you think about the fact that fault data in not used for analysis in PRORCA? 6. What do you think about resource-intensity of PRORCA? 7. What is your opinion about PRORCA's effectiveness in improving software quality and its efficiency? 8. How do you find the taxonomy? Is it informative? Would you consider using it to find mismatches between development practices and context?