e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240106, DOI: 10.37190/e-Inf240106

Activity-Based Detection of (Anti-)Patterns: An Embedded Case Study of the Fire Drill

Sebastian Hönel*, Petr Picha**, Morgan Ericsson***, Premek Brada****, Welf Löwe*****, Anna Wingkvist*****

*Faculty of Technology, Dept. of Comp. Sci. and Media, Linnaeus University, Sweden
**Faculty of Applied Sciences, Dept. of Comp. Sci. and Eng., University of Western Bohemia, Czechia
***Faculty of Technology, Dept. of Comp. Sci. and Media, Linnaeus University, Sweden
****Faculty of Applied Sciences, Dept. of Comp. Sci. and Eng., University of Western Bohemia, Czechia
*****Faculty of Technology, Dept. of Comp. Sci. and Media, Linnaeus University, Sweden

sebastian.honel@lnu.se, ppicha@ntis.zcu.cz, morgan.ericsson@lnu.se, brada@kiv.zcu.cz, welf.lowe@lnu.se, anna.wingkvist@lnu.se

"While it is certainly useful to study the successful ways people solve problems, the old adage that we learn from our mistakes suggests that studying failures might be even more fruitful.
– *Neill* et al.

Abstract

Background: Nowadays, expensive, error-prone, expert-based evaluations are needed to identify and assess software process anti-patterns. Process artifacts cannot be automatically used to quantitatively analyze and train prediction models without exact ground truth.
Aim: Develop a replicable methodology for organizational learning from process (anti-)patterns, demonstrating the mining of reliable ground truth and exploitation of process artifacts.
Method: We conduct an embedded case study to find manifestations of the Fire Drill anti-pattern in $n = 15$ projects. To ensure quality, three human experts agree. Their evaluation and the process' artifacts are utilized to establish a quantitative understanding and train a prediction model.
Results: Qualitative review shows many project issues. (i) Expert assessments consistently provide credible ground truth. (ii) Fire Drill phenomenological descriptions match project activity time (for example, development). (iii) Regression models trained on ≈ 12–25 examples are sufficiently stable.
Conclusion: The approach is data source-independent (source code or issue-tracking). It allows leveraging process artifacts for establishing additional phenomenon knowledge and training robust predictive models. The results indicate the aptness of the methodology for the identification of the Fire Drill and similar anti-pattern instances modeled using activities. Such identification could be used in post mortem process analysis supporting organizational learning for improving processes.

Keywords: anti-patterns Fire-Drill Case-study

PREVIEW

© 2024 The Authors. Published by Wrocław University of Science and Technology Publishing House.
This is an open access article under the CC BY license international.
Submitted: 03 Mar. 2023; Revised: O4 Jan. 2024; Accepted: 26 Feb. 2024; Available online: 01 Mar. 2024

1. Introduction

A pattern describes a reoccurring problem and its prototypical solution [2]. If the solution exacerbates – rather than ameliorates – a problem, a pattern becomes an *anti-pattern* [3, 4]. There exist anti-patterns related to, e.g., software architecture, that have a palpable effect on product quality. Other types of anti-patterns that have negative repercussions on the *process* and the resulting product could be attributed to management malpractices [5, 6]. Nelson summarizes a number of infamous project failure examples, most of which are the result of ineffective project management practices. The overall project failure categories are people, technology, product, and process [7], of which we focus on the latter. The software process is characterized by certain *activities*, such as development (e.g., implementation of features and fixing of bugs) or requirements engineering. The quality of this process depends, among other factors, on an adequate allocation of time for carrying out each activity. The quality of a software product was demonstrated to be tied closely to the quality of the software process [8]. Halvorsen and Conradi [9] even suggest the causal relation Quality(*Process*) $\Rightarrow$ Quality(*Product*). Therefore, it is worth studying phenomena related to process quality, in order to understand and improve product quality. Project failure is a frequently embraced opportunity for post mortem organizational learning [10]. Failure can often be attributed to process anti-patterns, of which the so-called "Fire Drill" is a prominent example, due to its clearly discernible symptoms [11, 12]. It is often characterized by "months of monotony [...]" (unsatisfactory early project progress) that is "[...] followed by a crisis" [4], due to forcing immediate delivery [13].

The Fire Drill is one of many existing management anti-patterns known to exacerbate software processes fraught with problems [14]. Usage of such anti-patterns for organizational learning is inhibited by a variety of factors today. The most pronounced problem is perhaps the lack of a quantitative description: anti-patterns are only described qualitatively using, e.g., structured templates with elements such as causes, symptoms, or consequences [5]. This effectively constrains the available methods of analysis to qualitative assessments. The manual evaluation depends on experts knowledgeable and available in the problem domain [15]. The problem is further aggravated by the fact that qualitative assessment is labor-intensive and error-prone because experts are likely to introduce their own subjective bias [16]. Although software development processes produce a multitude of diverse digital artifacts either as a by-product (e.g., source code) or based on the use of project or application lifecycle management (ALM) tools, most of these data cannot facilitate anti-pattern instance detection [17]. Even though there exist approaches that take advantage of such artifacts by using more formal and technical models, or thresholds and rules, the scope of their utility is severely limited to single process aspects, such as progress, variable dependencies, or estimating uncertainty, e.g., [18–22]. Lastly, scarcely available historical data, that is, data on past projects, impede or prohibit comprehensive and generalizable learning that results in organizational knowledge.

Many of the existing approaches have *software process improvement* as the ultimate goal. They first establish some *white-box* model and then analyze the effect of the measures implemented within the model boundaries [23, 24]. In contrast, we first conduct a longitudinal embedded case study [25, 26], which qualitatively evaluates the presence and severity (i.e., how strongly the phenomenon manifests) of a Fire Drill in $n = 15$ student projects collected over a period of three years, to find an accurate ground truth[1]. Through observer and data triangulation [29], as well as inter-rater reliability assessment [30], we ensure

[1]The feasibility of studies under similar constraints of scarce data have previously been successfully conducted in, e.g., biomedical engineering [27] and material sciences [28].

PREVIEW

a minimum quality of the observed evidence that makes the foundation for the ground truth as is required for further analyses [31]. The ground truth is then used to leverage previously unusable artifacts to establish quantitative measures and to derive knowledge. An adaptive training of a gray-/black-box model allows predicting the (severity of the) anti-pattern instances using the implemented measures [32]. Finally, the new insights are propagated back to the studied case. From the digital project artifacts, we can confidently derive the carried-out activities, such as adding features or engineering of requirements[2]. Then, the Fire Drill is modeled in terms of time spent on (temporal accumulations of) these activities. We learn that the phenomenon is sensitive to a certain balance of these activities, that is, how much time on each activity is spent in relation to any other activity, at any point in time. We argue that this is also the case for many related or similar phenomena. Therefore, next to the immediate contributions, the intended main contribution is the methodology that will allow full or partial replication of this study (using, e.g., different phenomena or modified contexts).

1.1. Data used in the study

Embedded case studies, such as the present one, are characterized by and rely on qualitative and quantitative data [35]. The study design is split to appropriately study either type of data. For the remainder of this work, we refer to these as *type-I data* and *type-II data*, respectively. Most data are produced in the development process and related to project planning, application lifecycle management, version control, and documentation. As either type contains both qualitative and quantitative artifacts, we define the types of data as follows.
Type-I Data. In addition to archival records, this case study uses direct observations, memory logs, meeting minutes (e.g., from stand-ups, retrospectives, customer meetings, or iteration planning), participant observation, team experience reports, customer comments, mentors' assessment notes, and wikis (or other types of note collections) maintained by each team as primary data sources for the qualitative portion. These data are largely unstructured and were recorded mostly subjectively. The type-I archival data is treated as a set of distinct data sources; the production and extraction of information related to these sources are outside the scope of the research method in this study [29].
Type-II Data. For the quantitative part, this study mainly uses out-of-sample testing for performing model validation and generalization error estimation. The data is structured and comes from the archives of the ALM tools. It includes version control systems that hold the source code (and its commits) and raw data from the underlying issue-tracking tools. The latter are mostly tickets that have been assigned a category and time estimates. Although there is some uncertainty in these tickets, all other data used in the quantitative analysis were objectively recorded.

1.2. Objective

Clearly defined objectives are a common element of the research design for case studies [29]. We pursue a single, principal objective refined into research questions to which we offer some answers. For this case study, the principal objective is defined as follows. *Automate the post mortem Fire Drill severity assessment, by utilizing the qualitatively won ground*

[2]There is a partial semantic overlap between our activities and the so-called disciplines as used in the (Rational) Unified Process [33, 34].

PREVIEW

truth and selected suitable quantitative, objective type-II artifacts. Pursuing this goal will necessarily result in the extraction and generation of knowledge from the type-II artifacts. The point of departure for the principal objective is one in which, before conducting the case study, we are relatively certain that the Fire Drill, given its existing phenomenological descriptions, is detectable and that some projects will exhibit one. Although this was the result of our pilot study (see Section 3.2), we are yet uncertain of the quality of the qualitative evaluation (whether severity can be determined *sufficiently* accurately). Although all data were available at the beginning of the study, that is, type-I and type-II data, we did not know if and how many of the type-II artifacts would be suitable for automated detection, implemented as a regression model. Also, manual facilitation of the type-II data is difficult, as the Fire Drill was not previously described from a quantitative perspective. Manual analysis of quantitative data is further inhibited by the data size (data points and dimensionality) and possibly non-linear correlations.

1.3. Propositions, Hypotheses, Research Questions

In this section, we will guide the reader through the main aspects of the case by presenting three sets of propositions, hypotheses, and research questions in a consecutive manner. For case study research, it is suggested to align the methodology close to these elements (however, not necessarily in this order; e.g., [29, 36]). The If-Then-styled propositions first give potential implications, while the generated hypotheses provide structure and detail to the formulated research questions.

1.3.1. Understand the Fire Drill Manifestation

The first set of propositions, hypotheses, and research questions concerns the manifestation of the Fire Drill in our context directly.

Pr1.1: If the Fire Drill phenomenon is described well enough and present in the projects, then manifestations of it can be found using type-I data only.

Pr1.2: If there is agreement between independent raters, then the existing phenomenological descriptions of the Fire Drill are sufficient for a qualitative post mortem evaluation.

Pr1.3: If there is agreement on the absence of the Fire Drill in a project, then the evidence that is counter-indicative of the phenomenon (true negatives) can be gathered.

Hyp1.1: The type and quality of the type-I data allow for accurate assessment of the severity of the Fire Drill using qualitative evaluation.

Hyp1.2: Projects not affected by the Fire Drill can be used to derive common symptoms whose presence can indicate its absence.

RQ1.1: Can the severity of the Fire Drill be accurately determined?

RQ1.2: What is the nature of the Fire Drill manifestation in each of the student projects?

RQ1.3: What evidence can be gathered indicating the absence of a Fire Drill?

1.3.2. Establish an Understanding Using Qualitative Data

The won ground truth now allows us to access, reason about, and leverage available quantitative data in an unprecedented way. The second set of propositions, hypotheses, and research questions was designed to establish a new quantitative understanding. Our approach here is characterized by instrumental motivation and nomothetic evaluation. Sets two and three facilitate quantitative project data of the same structure across all

PREVIEW

projects. While the focus is on the sub-unit of analysis, the projects are examined together (nomothetically), instead of individually.

In the following, *importance* refers to the concept (and technique) of (determining) variable importance for the prediction of a dependent variable (here: severity) [37]. When the timeline of a project is subdivided into phases, then each feature (variable) is exclusively assigned to a single phase. Therefore, variable importance allows us to determine and compare the relative importance of each project phase by aggregating the importance of each feature.

Pr2.1: If a ground truth can be determined accurately enough, then it can be exploited for the analysis, interpretation, understanding, and comprehension of the quantitative data.

Hyp2.1: Activities closely related to those described by the Fire Drill will display characteristic behavior that is in accordance with the Fire Drill's phenomenological descriptions.

Hyp2.2: All project phases will exhibit non-zero importance, with the later phases being of greater importance (the Fire Drill, supposedly, is more critical towards the project end).

RQ2.1: What are typical accumulations of (maintenance) activities characteristic of a Fire Drill?

RQ2.2: What phases and activities are most important for predicting the presence or severity?

1.3.3. Obtain a robust predictive model

The last set of propositions, hypotheses, and research questions was designed primarily to achieve the main objective (see Section 1.2), namely to automate the post mortem severity assessment. As a by-product, we will also improve our quantitative understanding, as continued from the previous set. In the following, *stability* refers to two criteria simultaneously, the first of which is the expected generalization error and the second of which is the confidence interval regarding the generalization error of a predictive model.

Pr3.1: If any type-II data (artifacts) are suitable, training should converge toward stability with increasing amounts of training data.

Pr3.2: If a stable model with acceptable generalization error can be obtained, then assessing the severity of the phenomenon with it could be instantaneous.

Hyp3.1: Type-I ground truth and type-II data from a few projects are apt for training a regression model with acceptable stability.

Hyp3.2: Using either source code or issue-tracking data should yield predictive models of similar stability, as the former is more objective, while the latter is more closely aligned with the Fire Drill's described activities.

RQ3.1 What source of data, issue-tracking or source code, yields better models?

RQ3.2: How many data points are required for obtaining a stable predictive model?

1.4. Notions and abbreviations

Activities use an all-caps sans serif font and are abbreviated by a few letters. For example, the activity *development* is denoted by DEV. It is also used as a subscript for functions over time, that is, f_{DEV}. Similarly, empirical observations, symptoms, and consequences

PREVIEW

use few letters and a number, in typewriter font. For example, `ESC01` is the first empirical observation of a symptom/consequence.

1.5. Structure of this article

The remainder of this article is structured as follows. The next Section, 2, introduces related and relevant work. Section 3 provides background information on phenomena described using a pattern language, with a dedicated focus on anti-patterns and the Fire Drill, as well as details about the preceding pilot study. The entire design of this study and the underlying methodology is presented in Section 4. It is followed by Section 5, which is dedicated to the presentation of the results of the analyses. The validity of our study, the limitations of its results, their replicability and generalizability, as well as a summary of the results obtained and how they relate to the established propositions and hypotheses, are discussed in Section 6. The conclusion and synthesis of all results related to the case studied, as well as prospects for future work, are given in Section 7.1. At the end of the article, the additional appendices A through E, providing supplementary details and insights, can be found.

2. Related work

Pattern-like phenomena related to management are not models because they only describe a well-known solution to some recurring problem [3], but effectively lack efficient operationalizability (in terms of, e.g., a predictive model for presence and/or severity). Simeckova *et al.* [5] have identified a wide semantic gap between the qualitative (textual, unstructured, and often ambiguous) and quantitative description of pattern-like phenomena, where the latter is practically impossible to specify. This is because patterns are deliberately left abstract (or even "vague", as Alexander *et al.* [2] put it). Therefore, a quantitative description would require, for example, context-specific thresholds or rules according to Simeckova *et al.*

Currently, no software process improvement model that could adequately represent or evaluate the presence of a Fire Drill based on quantitative data exists. Brown *et al.* [11] started to characterize the Fire Drill anecdotally, describing the problem it portrays, together with a refactored solution called *sheltering* (see Section 3.1). They did not attempt to abstract from this, i.e., there is no list of symptoms, for example. The only possible form of operationalization would be a manual evaluation of whether and to what degree the own project matches their description. Although they do not use the example of a Fire Drill, Laplante and Neill [4] were the first to collect various managerial and cultural anti-patterns and to represent each in a common, structured template. It included, for example, the anti-pattern's central concept, its dysfunction, a short vignette, a plain explanation of the anti-pattern, clues to alleviate the fallout, and – most importantly – a yes/no checklist with symptoms for identifying the anti-pattern's presence. More recently, Picha and Brada [18] started to collect and consolidate anti-patterns in a common template. They extract and gather various characteristics from each anti-pattern and translate some of their peculiarities into actionable symptoms, consequences, and solutions.

Attempts exist to operationalize various (anti-)patterns for software process quality. For example, next to the informal descriptions, there are more formal models, such as Bayesian (Belief) networks, ontologies, social networks, and Design structure matrices [6, 19–21]. Of these, only Bayesian networks are actionable to a limited extent, as they allow for modeling conditional probability distributions and visualization of dependencies between variables.

PREVIEW

The other methods are rather concerned with encoding project management knowledge into machine-readable form. The actionability in Bayesian networks is limited to assessing uncertainty; that is, it is not suitable for detecting the presence of anti-patterns but rather a tool for exploring relations [22]. However, they may be used to measure certain aspects of the process, such as its progress or quality.

There is plenty of other research on quantifying software quality using ALM artifacts. For example, Draheim and Pekacki [38] focuses on developers' activity throughout the project using collaboration, productivity, and evolution metrics. Ramsauer *et al.* [39] deal with estimating the maintenance costs in software development, and Tamburri *et al.* [40] study the organizational aspects of software projects and communities. Talpová and Čtvrtníková [41] use ALM data to investigate Scrum anti-patterns in a case study, though only as a secondary source of information combined with surveys and interviews. Hachemi [42] explores the reuse of patterns (in a more colloquial sense) in the modeling of software development processes. Although other researchers also focus on (anti-)patterns, they do not use ALM data. Frtala and Vranic [43] research organizational patterns and their adoption in individual organizations and projects using gamified learning techniques. Settas and Stamelos [20], as well as Stamelos [6] aim specifically at project management anti-patterns, their knowledge base, and effective communication using heterogeneity of personalities and character traits of developers. A major part of the research efforts also deals with the modeling of (anti-)patterns and detection through languages and ontologies [44], or models like the software process engineering meta-model [5] and the business process model and notation [45].

To the best of our knowledge, no one has previously attempted to operationalize an anti-pattern using the approach presented in this study. It was previously shown that performing post mortems is a viable path to organizational learning [10] and that learning from anti-patterns is deemed a way to eventually master management knowledge [6]. Also, it appears that examining and learning from eventuated anti-patterns is not limited to the context of software development. For example, Awad *et al.* [46] use them to detect compliance violations of business processes. Lastly, the Fire Drill is a phenomenon that can lead to anything between ever-so-slight and severe, negative repercussions, such as total project failure. Studying project failure typically results in the discovery of many interrelated symptoms and consequences [47]. Although compared to the application of common software process improvement models, we address only a single quality goal, we observe numerous different symptoms and consequences. It appears that Neill *et al.* [1]'s statement about the fruitfulness of studying project failure has become true.

3. Background

This section provides some background information about phenomena described using a pattern language, with a focus on the Fire Drill anti-pattern. It also gives an overview of previous work, that served as a pilot study.

3.1. Phenomena described using a pattern language

Generally, patterns are reoccurring and identifiable phenomena [2], that are especially prevalent in phases of design, project management, and in software development processes [48]. A pattern provides a general and proven solution to a common problem.

PREVIEW

> *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."* – Alexander *et al.*

However, as the definition by Alexander *et al.* [2] implies, the descriptions are deliberately left vague in order to leave room for case-specific applications of the solution the pattern represents. The definition also highlights the challenge of patterns to date, which lies in objectively definable and quantifiable problems, and the lack of technical and automated solutions to these. While efforts for a more technical way of describing patterns have been undertaken (e.g., [1, 11, 18]), they are predominantly described using a *pattern language*, structured text, or a template [6]. Therefore, patterns are most often not described quantitatively and lack clear connections to quantifiable properties, such as software metrics.

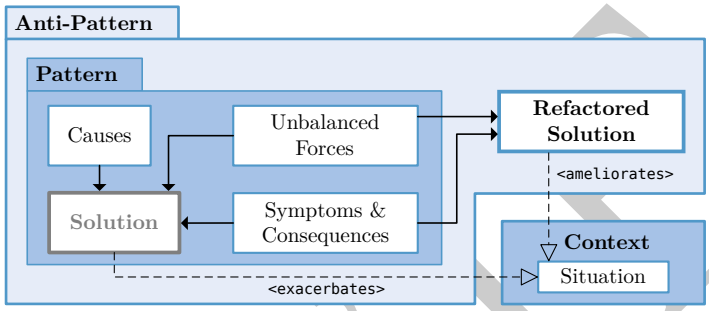

Figure 1. Some common elements of patterns and anti-patterns. The context element refers to the context (and its problematic situation) of the case study itself (see Section 4.1)

An **anti-pattern** can be conveyed as a larger concept than a pattern, in a way that it encompasses the elements of a pattern, but also adds an additional, so-called *refactored* solution [4]. The refactored solution is, for example, a restructured, improved, or otherwise optimized version of the original (regular) solution. The regular solution that comes from the enclosed pattern is what creates the anti-pattern in the first place because its application constitutes malpractice that **exacerbates** the problem, rather than ameliorating it (see Figure 1). Therefore, the actual mending solution is the refactored solution, rather than the regular solution. The refactored solution is an element that is known for some anti-patterns, but not for all. If present, it is commonly included in the pattern language or structured template (e.g., [14]). As a result, anti-pattern project management phenomena pose threats to project quality and delivery and are the outcome of human error. The ramifications include but are not limited to, developer churn, interpersonal and organizational tensions, a product of poor quality, delayed delivery, or even total project failure. The notion of an anti-pattern is, therefore, closely related to that of *project risk* [6].

3.1.1. The Fire Drill

The presence of a Fire Drill is a kind of problematic software project situation. Most recently, the Fire Drill was described following a common, structured template (see Appendix A). Rudimentarily speaking, it is construed as a short, desperate, and highly active phase towards the end of a software development project. It is preceded by a considerably longer, much less productive phase. Often, a disproportionately large amount of the project's resources is spent in the first phase, without advancing the project at the required pace.

PREVIEW

When management acknowledges the urgency of the imminent due date, the start of the typical Fire Drill is heralded [3, 11, 12].

While the Fire Drill is a phenomenon that unfolds as the result of poor management during the development phase of a project, some projects are predestined to exhibit one due to poor time allocation for the preceding approval and budgeting processes. This phenomenon is known as the "fuzzy front end" and often results in an aggressive development schedule from the beginning [49].

3.1.2. Patterns related and similar to the Fire Drill

The Fire Drill is related to other phenomena, some of which it is partially indistinguishable from. This happens because a certain set of indicators and symptoms & consequences are similarly indicative of other phenomena. Some related anti-patterns are, for example, "Analysis Paralysis"[3] (a potential cause) or "Collective Procrastination" [18] (a more generic case). Other anti-patterns, such as "Half Done is Enough"[4], or "Brook's Law" [50] may constitute typical symptoms associated with an early-/late-stage Fire Drill. The anti-pattern "Cart Before the Horse" emerged as part of a Fire Drill in some of the affected projects. While it is a pattern of its own, it is a typical, severe, and frequently occurring symptom of a Fire Drill (see `ESC2` and `E20` in Appendix C), that may have high project risk as one of its consequences. What is furthermore similar, is that these phenomena, in theory, can also be identified based on the ALM data or ongoing activities.

3.2. Previous work

In an earlier paper that served as a *pilot study* [51] for this work, we primarily investigated the type-II data. Next to exploring and visualizing the work carried out in the projects, the goal was to assess whether a model or a plain decision rule for presence detection can be derived from the data and applied to future projects. That study did not include a qualitative evaluation of the Fire Drill in each of the projects. Furthermore, the absence of more than two raters prohibited a proper assessment of the inter-rater agreement and the quality of their findings. From the pilot study, we conclude that the usage of naive models, whether expert-designed or purely data-driven, is not beneficial, as a model requires a more adequate representation of its features. We attempt to represent the time spent on activities in two different ways. After exploring the data, we explicitly define three activities for issue-tracking that are related to requirements engineering, development, and descoping (see Section 4.5.1). For source code data, we predict the so-called maintenance activity [52] that is associated with each commit [53] (see Section 4.5.2). These activities are related to adding features, correcting faults, and perfective changes (e.g., maintenance). For each instance of an activity, it is always recorded *when* it happened. This allows us to detect temporal accumulations of comparatively lower and higher density. In issue-tracking, we additionally have access to the duration (i.e., how *much* time was spent) of each recorded activity. The duration for commits, however, remains unknown. For issue-tracking data, we chose a cumulative and normalized representation, since this data tends to be more scarce (for example, bug tickets are only opened rarely). For source code data, we choose to represent the time spent as continuous-time random variables. We conclude that the

[3]Analysis Paralysis. 2017. http://wiki.c2.com/?AnalysisParalysis.
[4]Half Done Is Enough. 2023. http://wiki.c2.com/?HalfDoneIsEnough.

PREVIEW

latter representation is the most suitable for either type of data. Furthermore, we suggest using weighted density estimation for issue-tracking data, additionally considering the time spent as a weight for the temporal accumulations of these activities. We recommend using continuous probability densities also for another reason:It is straightforward to derive two kinds of features from them. First, a density can be integrated along an interval for estimating the relative amount that was carried out for an activity. Second, calculating a divergence between two densities is well understood and can be exploited for identifying disparities between activities. It is likely that common regression models (mind they require an accurate ground truth) will outperform the pilot study's approach using our findings. While a binary decision rule for presence detection can achieve a respectable accuracy, it is of obviously limited use as a severity assessment device and prone to producing false positives or negatives. Attempts to create a more fine-nuanced rule failed.

4. Case study design

We perform a *single-case embedded* case study based on *intrinsic* and *instrumental* motivation. We use qualitative and quantitative data and present the results in a mostly structured format [26]. From Sjøberg [54, 55] we may understand "a case [...] as a single, empirical configuration of actors, activities, technologies, and artifacts, all within a context." However, while all projects *do* share the same case, their empirical configuration varies. This circumstance necessitates the application of an embedded design. For example, in each project, a different product is developed, by a different group of students, applying individual practices (to some degree) to achieve their goals (see Appendix B for the full project setup). Therefore, the multiplicity rather lies in the analysis units (the projects themselves) and not in the cases, requiring an embedded design. A non-embedded design, if not studying multiple cases, would be concerned with a single unit of analysis and, therefore, not be a suitable choice for this work. Embedded case studies propagate the findings from the analyzed units back to the single case studied. Yin [25] notes that the project-level data may be highly quantitative, and the original evaluation would become a project study, i.e., a multiple case study of different projects if there is no investigation at the level of the original case. Scholz and Tietje [26] note that the multiplicity of evidence in an embedded case study is investigated partly in the projects, each focusing on different and salient aspects of the original case. In a multiple case study, each case should serve a specific purpose within the general scope of the investigation, which does not apply here. Furthermore, all propositions, hypotheses, and research questions are the same across all embedded units. Therefore, we only have a single case of study [25].

Multiple case studies follow a replication logic that starts with uncovering a significant finding that is subsequently replicated using additional case studies. However, our study is a single evaluation (concerning a single case) across multiple projects. The logical sub-units (or embedded units) are the selected $n = 15$ projects, which makes a holistic design inapplicable and warrants an embedded design instead. It is common for embedded case studies to facilitate (sampling of) quantitative data and the application of statistical analyses [35]. The **case** (subject, or main unit of analysis) is *the Fire Drill within a software engineering course* (see Figure 2). The course is the *Advanced Software Engineering course*

PREVIEW

at master level[5] conducted during the second out of a four-semester study at the University of West Bohemia in Pilsen. Given the setup of the projects (e.g., agile, iterative, milestones, etc.), they are naturally subject to the Fire Drill phenomenon. The full context of our study is given in the next Subsection, 4.1.

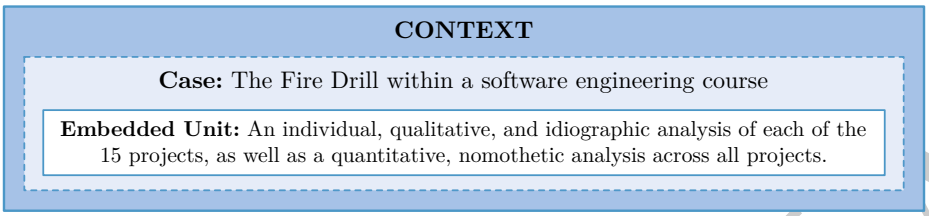

Figure 2. Case study design

Since we perform an embedded case study in the realm of software engineering, our method is closely aligned with guidelines from Runeson *et al.* [29] and Wohlin *et al.* [56]. The research strategy chosen here is exploratory, explanatory, and of an improving nature. It is not descriptive, as we are not trying to portray the current state of the Fire Drill phenomenon [29] (for that, the interested reader is referred to Section 3.1.1, Appendix A, and [3, 11]). It is, however, exploratory and explanatory since we seek new insights into the phenomenon's manifestation within the chosen context. The case is studied to interpret and explain symptoms, causes, and consequences and establish a connection to quantitative data that cannot be fully utilized. This is closely connected with the improving nature of this study as we attempt to add to the current qualitative understanding and establish a new quantitative awareness.

Wohlin and Rainer [55] provide a case study checklist. The first requirement is an identifiable case, which we have presented here. The second requirement is that of a real-life context. It is satisfied since we do not attempt to generalize beyond the chosen context (e.g., industrial). The third requirement is that of using multiple data collection methods. The variety of methods and sources used is described in Section 1.1. The fourth requirement is the study of a contemporary phenomenon. It is satisfied by us studying the projects as they occur. The last requirement is that the researchers do not act as change agents in the projects as they unfold. Although one of the raters was a mentor in most of the projects, that role was purely passive with respect to at least the Fire Drill phenomenon (see Section 4.1.3). This is evident because some projects showed strong manifestations of the phenomenon (or other anti-patterns) regardless. Furthermore, in software engineering, case studies are expected to establish long-term objective(s) (as opposed to action research that favors short-term change), which we did (see Section 1.3). We came to the characterization of our study as a single-case embedded case study, together with these properties, after discussing its nature and virtues in detail [57]. Although our analysis for the first part happened post mortem, we do not necessarily apply the full spectrum of software project post mortems. Therefore, we must separate ourselves from post mortems and processes as defined in, e.g., [58, 59]. Although our analysis partially resembles what Tiedeman describes as postmortem planning and design/verification, their conceptual framework is not applicable here for the above reasons.

[5]Course homepage. 2023. https://courseware.zcu.cz/portal/studium/courseware/kiv/aswi/?pc_lang=en.

PREVIEW

Case studies are *"... studies of something general, and of something particular ..."* [60]. These two different moments are characterized by an *idiographic* and a *nomothetic* part. Therefore, this study is divided into two consecutive parts (see Sections 4.4 and 4.5). The first qualitative part seeks to gain an understanding of the Fire Drill's manifestation in student projects. Here, we make an individual, idiographic effort to understand the phenomenon in its context. This is also where we assign a severity to each project (the ground truth). The first part is closely aligned with the first set of propositions, hypotheses, and research questions as presented in Section 1.3. The motivation here is mainly of what Scholz and Tietje [26] call *intrinsic* character, meaning that we have a somewhat personal incentive to understand this case, as one of the authors has previously been involved in the conduct of these projects and will continue to be in the future.

4.1. Context

We report the context of this study aligned with the applicable points of the framework, as suggested by Petersen and Wohlin [61] and Runeson *et al.* [29]. Specifically, we report details on the products (what was manufactured in each project), processes (activities, artifacts), practices (tools and techniques), people (students and researchers), organizations (the university), and the (hypothetical) market (the customer). Appendix B lists, for each project, what kind of application was produced, its size, the used programming language(s), the number of team members, project duration in days, number of iterations and issues, time logged on issue-tracking activities, and number of commits per maintenance activity.

4.1.1. Product

Each team developed a single product over the course period, commissioned by the customer. Typically, an individual customer was assigned to each project. It is up to the customer's discretion to select an appropriate balance between product maturity and quality and the number of features. Typically, two-week iterations were used to ensure a certain minimum level of product completeness, quality, and installability.

4.1.2. Processes and practices

The goal of the projects is to approximate a real-world setting as closely as possible. The projects typically run for approximately three months, with a typical workload of $\approx 65-80$ hours per student. Each project follows the Unified Software Development Process [33]. The methodology is influenced by agile practices, with two-week iterations ("Sprints") and a final fixed deadline for product delivery. The students manage the projects autonomously, with a staff member assigned to each project for oversight. For issue tracking and version control, teams must use common ALM tools, such as Redmine or GitHub. Most teams use additional tools, such as Wikis, automated testing and deployment (continuous integration/delivery), or shared documents for, e.g., the requirements specification, implemented architecture, meeting minutes, or customer communication. We classify the rich set of artifacts captured during the process into type-I and -II data (see Section 1.1).

PREVIEW

4.1.3. Students, researchers, customers, and the organization

All students involved studied at the second cycle (master's level). The number of students in each project is shown in Appendix B. The customer for each project came either from the university (the same or another faculty) or an external company. Since the projects were conducted in an academic context, the organization was the University of Western Bohemia in Pilsen. Instead of monetary interests, however, the university's interests are ensuring high educational standards (especially by studying the outcomes), as well as preparing the students for a potential future work environment.

Rater 1. The first rater is currently a doctoral student. Their main interest in the study is the detection of project management anti-patterns through the usage of ALM data. The first rater has about five years of industry experience working at DHL, predominantly with help and advice on using ALM tools (e.g., IBM Engineering Lifecycle Management) and consulting on software processes. They were previously involved in three process audits for industry, mining ground truth (quality of their processes or compliance to norms; using similar data plus interviews). These audits lasted for a few months, approximately one year, and one for approximately four years. The first rater worked as a mentor in the advanced software engineering course for nine consecutive years. On average, they were in charge of approximately five projects (with a range of three to eight). It is important to note that their mentor role is purely passive. They never actively attempted to avoid (or alleviate) the Fire Drill anti-pattern (or any other malpractice, for that matter). This becomes evident because some projects showed strong manifestations of the Fire Drill phenomenon (or other anti-patterns). The students could (and did) choose to ignore mentors' advice. Some students have previously failed the course.

Rater 2. The second rater holds a master's in software engineering and did not previously participate in the course as a mentor. The second rater is a former doctoral student with a main focus on project data analysis. They have over three years of industry experience working at Unicorn[6] as a project manager. Rater Two never had any active role in conducting the software engineering course.

Rater 3. The third rater also holds a master's degree in software engineering. They were previously enrolled in a doctoral program for about four years. The main focus of their third-cycle studies was the compatibility of architecture components. With more than four years of industry experience, rater three works at the Finnish software company Yoso[7], where the main customer is the Finnish state. Their role is that of an architect and a lead developer, as well as project analytics and -management (e.q., requirements analysis). They are also the head of the company's local branch in Pilsen. Rater Three never had any active role in conducting the software engineering course.

Assessor. The assessor's role was to analyze, interpret, and aggregate the raters' findings independently. They are currently enrolled in a doctoral program and hold a licentiate degree. Their main focus of study is software and information quality, with a strong focus on statistical learning and mathematical optimization. The assessor has approximately four years of prior, predominantly agile industry experience, working as a systems developer (at SAP Research[8]), lead architect, project manager, and product owner (at Softwerk[9]). As such, they have first-hand experience with agile product development. The assessor

[6]Unicorn: About Company. 2023. https://unicorn.com/en/company-profile.
[7]Yoso: Company Homepage. 2023. https://www.yoso.fi/.
[8]SAP: About Innovation and Research. 2023. https://www.sap.com/about/company/innovation.html.
[9]Softwerk: About Company. 2023. https://softwerk.se/en/about-us.

PREVIEW

also took similar roles as a teaching assistant in two unrelated first- and second-cycle agile software development courses over the course of three years (assuming a role in between two and eight projects simultaneously in each course). The assessor never had any active role in conducting the software engineering course.

4.2. Embedded unit selection

The natural choice for the (sub-)unit of study is student projects. We initially also considered studying the Fire Drill in open source or industrial projects, but this would probably have been a purely archival study due to the lack of a real-life context [54, 57]. Furthermore, the social context of open source projects is hardly observable, and the ALM data are notoriously incomplete or often absent entirely due to the absence of (proper) project management. The latter type of project is difficult to obtain, both in quantity and quality. Since the Fire Drill is a phenomenon that affects a project in its entirety, we regard quantity as slightly more important (for this reason, we conduct an embedded case study).

The benefit of using the student projects to study our case lies in the accessibility of the associated ALM data throughout the entire lifecycle of each project. As course runners, we directly observe and record the day-to-day realities of each project. This additional data and knowledge are crucial for an accurate assessment of the ground truth. Our data spans **15** projects, four of which were conducted in 2019, five in 2020, and six in 2021, between March and June each year. We have previously detailed the various data types available for each project (see Section 1.1). We only included projects that provided full access to the ALM data. Therefore, a few projects had to be excluded. The non-availability of data can be primarily ascribed to the usage of external, proprietary ALM tools, sometimes demanded by industrial customers' non-disclosure requirements.

4.3. About the data

All data used in this study were previously made available as an anonymized open-access dataset [62]. It was used and made available only in digital form and was recorded exclusively digitally. All data are associated with application lifecycle management, that is, data related to application governance, development, and operations [63]. Therefore, most of the data are produced automatically by the required usage of ALM tools. The dataset includes all original artifacts for each project. To avoid introducing translation bias, textual artifacts recorded in Czech were kept pristine. The dataset also includes data collected by observation, such as mentor notes, meeting minutes, and retrospective records. The data was then digitized and added to the set.

The immediate purpose of the data collection was evaluation and grading, and the data are initially kept for no specific reason other than archival records (the university recommends keeping such course-related data for administrative purposes). The data are also kept and accessed while the projects are still running because it needs to be accessible during or after iteration reviews. Due to the absence of a specific reason, the recorded data are not specific to the Fire Drill phenomenon, as everything of potential relevance was kept or recorded. Therefore, the data can be used in a future study using a different case and/or research questions.

Since the data stem from software engineering projects conducted at a Czech university, the projects' and their data are subject to Czech legislation. As for ethical considerations, the legislation states that any student work done as a course assignment can be freely used

PREVIEW

by the educational institution for the purposes of its main functions, research obviously being one of those.

4.4. Qualitative design

Approaching (anti-)patterns like the Fire Drill is first done through the available phenomenological descriptions. The multiplicity of evidence is first investigated qualitatively on a per-project basis. We draw on methods for data source- and observer-triangulation to obtain robust ground truth estimates [29, 64].

The primary goal of the qualitative analysis was to understand if, how, and to what degree the Fire Drill manifests in each of the projects. To achieve this, the type-I data were subjected to manual and individual inspection by each of the three raters. Before individual analysis, a common understanding was established based on the available phenomenological descriptions (see Section 3.1) between the raters. The raters would then go ahead and extract evidence for the presence (and absence) of a Fire Drill (called the *raters' notes* in the dataset [62]). The notes reflect which symptoms and consequences are present, how severely they manifested, and how often they were observed. They also include findings that are counter-indicative of a Fire Drill. To date, the detection of (anti-)patterns has been subject to qualitative evaluation in practice.

After a complete evaluation of a single project, a rater would then indicate an overall severity using a linear numeric rating scale of 0–10, where 0 indicates the absence of the phenomenon and 10 the strongest possible manifestation. This assessment would then serve as **ground truth** in the subsequent use of type-II data (e.g., variable importance, regression model, etc.). We chose to assess the severity on a project level, since a Fire Drill, according to its existing descriptions, concerns a project over its full lifecycle. Furthermore, the Fire Drill's descriptions prohibited the use of a proper ordinal rating scale, such as a descriptive, verbal rating, or Likert scale. This is because the described symptoms and consequences (see Appendix C) do not come with a severity attached. For example, the second symptom/consequence, `SC2`, reads *"only analytical or documentational artifacts for a long time"*. Therefore, in the absence of a proper ordinal scale, the raters' subjective severity assessment formed only the basis for **three** subsequent analyzes.

The first analysis is to measure *inter-rater agreement*. As the ground truth assessment of the raters is subjective, the only way to objectively measure the proportion to which they agree is to use some agreed-upon scale or *benchmark*. To calculate the inter-rater agreement between more than two raters, Cohen's Kappa cannot be used. Instead, for example, one of Conger's, Fleiss', or Gwet's Kappa coefficients is required. We chose to report Gwet's "AC_1" Kappa coefficient, which outperforms other coefficients in terms of having reasonably small biases for estimating the true inter-rater reliability [66]. It is especially applicable in the presence of high agreement (which, as it turns out, is the case) because of its comparatively low bias. To benchmark the computed Kappa, we apply the widely used scale of Landis and Koch [67]. However, we should note that the proposed scale by Landis and Koch was arbitrarily chosen and that each Kappa is a point estimate associated with a probability distribution and a margin of error [68]. Therefore, it is recommended to properly benchmark the raw Kappa coefficients. Gwet suggests computing the probability that a Kappa falls into a certain range by integrating a standard normal distribution (where the Kappa coefficient is the mean and its associated error is the standard deviation) [30].

The second analysis was a common session between the raters conducted to reach a consensus on their rating, using the well-established Wideband Delphi method [69, 70]. In

PREVIEW

cases of diverging assessments, each rater presented arguments for their estimate, leading to a discussion, a repeated inspection, and a reconsideration of the information sources, until a final assessment was mutually agreed upon. Ties, rounding, and dissents were settled by the second rater since they never had any affiliation with the projects (e.g., as a mentor) and have the longest industry experience as a project manager to date. The raters were free to assign a final consensus value below or above their initial rating if there was sufficient reason to do so after the follow-up investigation of the data. The use of multiple experts is an effective measure to reduce subjective bias commonly introduced by expert-judgment software estimates [15]. The first two analyses were, in part, designed to guarantee a minimum quality of the obtained ground truth, as it is crucial for the analysis of the type-II data. Without the precautions implemented, robustness, reliability, and accuracy cannot be ensured otherwise [31].

The third analysis is a systematic approach that uses a well-defined ordinal scale to identify the prevalence of individual symptoms and consequences. Unlike the previously used numeric severity rating scale, the ordinal scale used here is of descriptive nature. Although each consecutive item represents a severity higher than that of the previous item, a linear increase is not implied. Severity is assigned or upgraded purely by description. The assessor (see Section 4.1.3) is to first classify each of the raters' notes and comments according to this scale. Furthermore, the assessor is to assign each observed empirical instance to one of the Fire Drill symptoms and consequences. This is done to allow us to answer the question of how the fire drill manifests itself in the projects empirically. Some of the observations, even though they are clearly related to our understanding of the phenomenon, may warrant a new superordinate symptom and consequence, especially if they cannot be assigned purely or only poorly to any of the existing symptoms and consequences.

After the first pass, a second pass is performed. In the second pass, observations (called empirical instances of a symptom or consequence and abbreviated as `ESCxx`) are conditionally aggregated and checked for data and/or observer triangulation, and the severity is adjusted accordingly. The following scale was used in both passes:

- `[0]` **None**: Not at all a problem: it applies mostly to false positives (e.g., a typical symptom that was caused out of the studied context and had no adverse effects).
- `[1]` **Miniscule**: Only slight indications of typical symptom(s) identified by at least one rater.
- `[2]` **Minor**: Multiple indicators and/or measurable/documented symptom(s); seldomly corroborated by another rater.
- `[3]` **Moderate**: Clearly identifiable and reoccurring symptoms or direct corroboration.
- `[4]` **Significant**: Like moderate, but the higher severity is evident through additional data- or observer-triangulation.
- `[5]` **Serious**: Agreement on the (recurrent) severe manifestation of a symptom by observer triangulation (often all raters) and/or data triangulation.

4.5. Quantitative design

The primary goal of the quantitative analysis was to facilitate the rich corpus of quantitative artifacts that are produced mostly automatically as a byproduct of conducting the projects and using the ALM tools. The goal is to enable quantitative data to contribute to the current phenomenological understanding and to automate expert-based post mortem assessment. The quantitative analyses facilitate the qualitatively won ground truth and type-II data exclusively. The available type-II data can be split into two main sources:

PREVIEW

source code and **issue-tracking**. This split is maintained throughout all analyses because, in reality, there might be access to only one of them. The won ground truth enables a wide range of statistical analyses and supervised learning. Quantitative data in the context of (anti)patterns are rarely useful. For example, consider the number of bugs over time. For this information to be useful, we would at least need some thresholds, and those would need to be universally valid. To address our objectives, hypotheses, and research questions, we also performed three analyses (Sections 4.6 through 4.8).

4.5.1. Activities in issue-tracking data

The Fire Drill and many related or similar phenomena are sensitive with regard to a certain balance of particular activities at any given point in time. For **issue-tracking**, we model three activities from the type-II data that, supposedly, are closely related to the activities described in the Fire Drill. These activities are as follows:

REQ: Activities related to requirements, analysis, and planning.
DEV: Time spent on development (implementation), testing, and bug-fixing activities.
DESC: Descoping; Effort that was planned for DEV, but never spent on it (i.e., the difference between the scope agreed on and delivered).

For each of the activities, the issue-tracking data provide timestamps (when an instance of the activity occurred) and duration. The title and description of the tickets were used to classify the issues into REQ and DEV, which were obvious choices to detect the Fire Drill. If there was only the slightest doubt, the issues were left uncategorized and not used. DEV reflects only adaptive engineering (i.e., adding features) because in these projects almost no maintenance activities are expected. Maintenance is only rarely done because there is no proper quality assurance and the delivery of agreed-upon requirements is of the greatest importance. Also, there are usually no or only a few ancillary functional requirements, such as response time, user-friendliness, or documentation artifacts. Therefore, activities other than forward engineering and bug-fixing were not considered. We also considered the frequency of the bugs, but it is unreliable because the reporting is not rigorous, often inconsistent, and sometimes completely absent. For example, many bugs were change requests in reality because the requirements were understood wrong. DESC, however, was constructed as it was deemed to be a valuable indicator for typical Fire Drill symptoms: Student projects have a hard deadline, so descoping happened frequently. The candidate solutions to a to-be-missed deadline are descoping, re-negotiation of the time frame, or overstraining people. Of these, descoping presents the *refactored* solution, while the others would exacerbate the situation. Therefore, descoping was the only allowed solution in this case.

4.5.2. Activities in source code data

From the source code, we model three types of activities from the type-II data as well. Source code is, compared to issue-tracking, a more objective source of information, because the data is not subject to human error and resulting inconsistencies (e.g., mislabeling of tickets). The source code repositories and commits thereof do not usually provide any form of annotations that would allow one to understand what kind of activity some committed work may relate to. Although it is possible to reference issues in commit messages, this feature was not used to label commits. Furthermore, we already derive three other activities from the tickets directly. By analyzing each commit's *source code density* [53], we can predict

PREVIEW

its associated maintenance activity [52] with great confidence. The three maintenance activities are as follows:

A: Activities related to adaptive/forward engineering (adding new features).

CP: Activities related to either corrective (fixing faults) or perfective (improve or change code to accommodate future features) work.

$\mathsf{FREQ_{nm}}$: Overall commit frequency, regardless of the associated maintenance activity.

While the used classifier can differentiate between corrective and perfective commits, the Fire Drill is not described using this distinction. Therefore, we combine these two activities into a single one. Although the overall commit frequency could be designed as a weighted mixture of A and CP, we chose not to do that. Instead, the frequency is designed by disregarding all labels, similar to CP, which does not make a distinction between corrective and perfective. Therefore, the variable is called $\mathsf{FREQ_{nm}}$, where the suffix "nm" indicates a non-mixture. The association between maintenance activities and the activities as described by the Fire Drill is likely to be worse than it is for issue-tracking. However, the increased objectivity of the source code data may give an edge to these data over issue-tracking data.

4.5.3. Modeling of activities as probability densities

We have previously identified and selected activities to be modeled, for which a proper representation has to be chosen. As a single consistent representation, we model each activity as a probability density function (PDF). This is similar to how the work distribution for certain workflows in certain phases is represented in the rational unified process [34]. The density reflects the timely accumulation of activities relative to each project's lifecycle. Since we do post mortem evaluation, the time between the first and last instance across all activities can be used to normalize all occurrences' timestamps into the range $[0, 1]$ after the project end. The temporal accumulation is estimated using kernel density estimation [71]. For both sources of data, source code and issue-tracking, we know the timestamp for which an instance of activity occurred. However, for the former, there is no indication as to the duration of each instance. For the latter, however, the duration of each instance is factored in as a relative weight when estimating a density, leading to a more accurate representation of the time spent.

4.5.4. Deriving features from activities

For each project, we have previously defined what activities we model and how. However, we have not yet derived any features. Generally, a feature is a measurable property with discriminatory power, which can be used in statistical analyses and machine learning. Since temporal accumulations of activities are modeled as probability densities (in case of issue-tracking also considering the duration using weighted estimation, see Sections 4.5.1 through 4.5.3), we choose **two major types** of features. The first type of feature is the amount (cumulative probability) of a certain activity that happens in a segment of a project. Its value can be determined by integrating the relevant interval of the activity's probability density. The second type of feature is the difference between any two activities on a segment that can be calculated using a (symmetric) divergence of their associated probability densities (denoted by the operator $|0$). In addition, we choose to subdivide each project into ten equally long segments. As the project time is normalized, we end up with a set of segments $\{[0.0, 0.1], ..., [0.9, 1.0]\}$. The original Fire Drill description only

PREVIEW

vaguely indicates (the length of) phases. Brown *et al.* [11] reports an anecdote of a typical project with improper burndown for about six months, followed by a Fire Drill of four weeks. However, this is in no way representative. Therefore, we choose said subdivision. We argue that ten segments will yield sufficient precision beyond Brown *et al.*'s two-phase model. Furthermore, the number of segments could be arbitrarily increased to further boost the precision, if desired.

The amount θ of a certain activity ACT in a segment $[a, b]$ is the integral of its associated probability density f_{ACT} over that interval. The probability densities of the activities are designed such that $\int_0^1 f_{\mathsf{ACT}}(x)\,dx = 1$, i.e., they integrate to one, the cut-off beyond the actual project time is sharp, and the PDFs of any two activities provide *absolute continuity*. Therefore, $[\sum \theta_i] = 1$ (where $i = 1 \ldots 10$ is the segment index). The amount of a feature in a segment can be directly interpreted as a percentage (the cumulative probability of observing the related activity in the segment).

The divergence between any two activities is commutative, that is, A diverges from B the same as B diverges from A. Therefore, a *symmetric* divergence is computed. The rationale behind this is rather practical. For the modeled activities, there is no distinction between the two mutual divergences. Furthermore, if $A \,|0\, B \neq B \,|0\, A$, the result would be twice the number of features. For three activities A, B, C, the resulting set would have a cardinality of six. However, for a symmetric divergence, only $A \,|0\, B$, $A \,|0\, C$, and $B \,|0\, C$ need to be computed. We choose the Jensen–Shannon divergence [72], which is a symmetric divergence (1). It is based on the Kullback–Leibler divergence $\mathrm{KL}\,(P \,|0\, Q)$. For two continuous random variables P, Q with probability densities p, q, the divergence is computed as follows.

$$\begin{aligned} \mathrm{JSD}(P \,|0\, Q) &= \frac{1}{2}\,\mathrm{KL}\left(P \,\middle|0\, \frac{P+Q}{2}\right) + \frac{1}{2}\,\mathrm{KL}\left(Q \,\middle|0\, \frac{P+Q}{2}\right) \\ &= \int_a^b \frac{p(x)}{2}\,\log\left(\frac{2\,p(x)}{p(x)+q(x)}\right) + \frac{q(x)}{2}\,\log\left(\frac{2\,q(x)}{p(x)+q(x)}\right) dx. \end{aligned} \tag{1}$$

The rationale behind computing segment-wise divergences is our assumption that the Fire Drill is sensitive with regard to a certain balance of particular activities at any given point in time. Therefore, we regard the divergences as an effective measurable property for observing such (im-)balances. Unlike the amount of activity, the divergence between two activities cannot be interpreted in a straightforward way and requires normalization.

4.6. First analysis: weighted mixtures

The first analysis examines the temporal accumulation of activities as is typical for when a Fire Drill is present in a project. For that, a *weighted mixture* for each activity across all projects is created. A weighted mixture is a convex combination of probability densities. In such a combination, each weight is greater than or equal to zero and the sum of all weights is equal to one (2). This is required for probability densities to ensure that no probability can become negative and the mixture integrates to 1. Recall that the raters' ground truth assessment was recorded on a numeric linear rating scale of $0 - 10$, with 0 indicating an absence of the phenomenon. The ground truth vector $\boldsymbol{\kappa}$ can, therefore, be scaled into a weight vector by normalizing it through the division of its sum (3). A mixture for some same activity ACT across all projects (4) is then created as a weighted sum (5).

$$\sum w_i = 1 \;\wedge\; \forall\, w_i \geq 0 \;\ldots\; \text{properties of a convex combination,} \tag{2}$$

PREVIEW

$$\boldsymbol{w} = \boldsymbol{\kappa} \left[\sum \kappa_i\right]^{-1}, \text{ the normalized ground truth,} \tag{3}$$

$$\boldsymbol{f}_{\mathsf{ACT}} \ldots \text{ vector of probability densities for activity } \mathsf{ACT}, \tag{4}$$

$$g_{\mathsf{ACT}}(x) = \sum w_i \, f_i(x), \text{ weighted mixture for activity } \mathsf{ACT}. \tag{5}$$

The result of this analysis will show us, for each activity, how it typically unfolds over the lifecycle of a project that is affected by a Fire Drill. The expectation is to observe a correlation between these activities and those as described, thereby establishing an additional, symptomatic, and quantitative understanding of the phenomenon. Although we currently have only $n = 15$ projects, each project added to the mixture will lead to a more accurate representation of the activities in the presence of a Fire Drill. With a sufficient amount of projects added in the future, the weighted mixtures may become their own, quantitative pattern description of the Fire Drill.

4.7. Second analysis: variable importance

The second analysis estimates the variable importance. The term *variable* can be interchangeably used with *feature*. It does not refer to a random variable, however. The variable importance is often used as the basis for selecting features that shall be part of the training [37]. Here, we compute it for a different reason, though, that is to enrich the existing phenomenological descriptions from a quantitative point of view. The results of this analysis are not used for adaptive training. We should note that the variable importance is specific and sensitive to the model with which it was computed. For example, a neural network will estimate it differently than partial least squares. Therefore, we average the variable importance as obtained from five different models, each repeated 100 times, to get a more solid understanding. The five models used are a boosted generalized additive model [73], a neural network [74], (generalized linear) partial least squares [75], and bagged CART [76].

In Section 4.5.4 we have described which and how we model activities, and what kind of features were engineered. To recall, two types of features, namely the amount of an activity and the (symmetric) divergence between two activities are used. Furthermore, the subdivision into ten equally long intervals was applied. In summary, for a single activity such as REQ or CP, the amount of activity in some segment is a single feature. Therefore, for either source of data (source code or issue-tracking), we modeled three activities each and segmented them into ten intervals, having two separate features (amount and divergence) per interval ($2 \times 3 \times 10 \times 2 = 120$). Therefore, the second analysis will answer questions the first could not answer. For example, which are the most (least) important segments (or phases) of the phenomenon, or whether the balance/divergence of activities is a more suitable predictor than the number of activities. The variable importance, therefore, complements the weighted mixtures of each activity. It cannot, however, answer as to which source of data, source code or issue-tracking, is more important (that is, more apt to predict an accurate severity). That is because variable importance is estimated on either data source exclusively, as we maintain the split.

PREVIEW

4.8. Adaptive training

The adaptive training, while also having analytical properties, is in direct correspondence with the main objective of this study: *To automate the post mortem severity assessment*. As a byproduct, we shall also learn, for example, whether and which source of data, source code or issue-tracking, is apt for use in a predictive model or what type of model works best. The design of the adaptive training is subject to the propositions that a ground truth with sufficient precision can be obtained (**Pr 2.1**), and that there exist some artifacts among the type-II data suitable for (adaptive) training (**Pr. 3.1**). We define the *adaptive training* to be a kind of **stability analysis**, a process to which training data is continuously added until the chosen *stability criterion* of sufficient confidence is satisfied [32]. The two principal quantities that we will consider are the empirical **generalization error** or ***risk***, R, and the confidence in obtaining predictions close to it.

4.8.1. Notations

We use notations very similar to those of Bousquet and Elisseeff [32]. Only symmetric (agnostic w.r.t. the order of the training samples) learning algorithms that produce a mapping from some input or feature space $\mathcal{X} \subset \mathbb{R}$ to some output space $\mathcal{Y} \subset \mathbb{R}$ are considered. All training is supervised. Hence, a training set S (6) consists of m tuples in $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, drawn i.i.d. from the unknown population D. An algorithm A, once fit, becomes the *hypothesis* $f : \mathcal{X} \to \mathcal{Y}$. Therefore, it is a function from $\mathcal{Z}^m$ into the hypothesis space $\mathcal{F} \subset \mathcal{Y}^{\mathcal{X}}$. We use the notation A_S to indicate that A was trained on S.

$$S = \{z_1 = (x_1, y_1), \ldots, z_m = (x_m, y_m)\}, \tag{6}$$

$$c : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+ \cup \{0\}, \tag{7}$$

$$\ell(f, z) = c(f(x), y). \tag{8}$$

The goodness of a fitted algorithm is evaluated using a cost function (7), which quantifies the difference between the true and predicted outcome. The loss of a fitted algorithm (a hypothesis f) with respect to an example $z = (x, y)$ is defined as in (8).

4.8.2. Stability analysis

The stability criterion chosen for a learning algorithm depends on the goal. In this study, the goal of the stability analysis is two-fold:

1. Select a champion model by the lowest empirical risk, among various configurations of learning algorithm, training data source, applied pre-processing, and used feature types.
2. Approximate empirical confidence intervals based on the probability that the champion model will predict an error that deviates from its expected error.

For (A), we require a model that satisfies the principal objective of predicting the severity of the Fire Drill on previously unseen projects with *sufficient confidence*. For that, we intend to repeatedly evaluate a large grid of various dimensions, in order to obtain robust estimates for the empirical generalization error for each setup (e.g., used model type, data source, etc.).

Instead of setting a threshold for what constitutes sufficient confidence, we select to tie the second goal, (B), to the amount of available ground truth once a champion was found since this is a source of actual scarcity. From pursuing the first goal, (A), the many

PREVIEW

repetitions for each constellation resulted in a set of validation errors. This set is regarded as a separate random variable, V. Using this notion, a confidence interval can take two possible forms. The first is to express it in terms of a probability that the model estimates an error that deviates less or more from the expectation, given some bounds for the lower (d_a) and upper (d_b) deviation (9). The second is to express it in terms of a deviation that corresponds to a given lower/upper probability (p_a, p_b), that is, given a probability, a function that indicates the maximally expected deviation from the expectation in either direction (10). We can conveniently express the two notions using the probability density function (PDF), cumulative distribution function (CDF), and percent-point function (PPF) of V.

f_V, F_V, f_V^{-1} ... PDF, CDF, and PPF of V,

$$\mu_V = \mathsf{E}[V] = \int_{-\infty}^{\infty} x\, f_V(x)\, dx, \text{ the expectation of } V,$$

$$g(d_a, d_b) = F_V(d_b + \mu_V) - F_V(\mu_V - d_a), \tag{9}$$

$$h(p_a, p_b) = \sup\left\{ \left| \mu_V - f_V^{-1}\left(F_V\left(\mu_V\right) - p_a\right) \right| , \left| F_V\left(\mu_V\right) - f_V^{-1}\left(\mu_V + p_b\right) \right| \right\}. \tag{10}$$

When V follows a unimodal distribution, common inequalities can be applied to estimate a confidence interval. For example, if $V \sim \mathcal{N}$, the Three-Sigma rule [77] can be applied. The following four common inequalities are ordered by their bounds, from tightest to loosest: Three-sigma rule < Vysochanskij–Petunin inequality < Gauss's inequality < Chebyshev's inequality [78–80]. If any of these should be applied, one shall first evaluate the tightest inequality for which the constraints are satisfied.

4.8.3. Training flow and model selection

The design of the training flow follows recommendations to obtain robust predictive models under the constraints of small sample sizes as given by, e.g., [81–84]. Varma and Simon and especially Vabalas *et al.* show that standard K-fold cross-validation (CV) produces strongly biased performance estimates, particularly with small sample sizes. This problem can be evaded by using some form of nested CV, train/test split approaches, and sufficiently many repeats. Using this approach, others have previously obtained high-performing models [85]. Due to the scarcity of our data, we oversample the dataset using the well-established synthetic minority oversampling technique (SMOTE) for regression, which has been proven to significantly increase model performance [86]. For the outer resampling of the nested CV, we define an extensive search grid. The dimensions are the following (the size of each dimension is in parentheses):

- (2) types of data source: either source code or issue-tracking.
- (6) types of models, one of bagged CART, generalized linear model [87], Gradient Boosting Machine [88], neural network, Random forest [89], and Support vector machine [90].
- (3) types of features used in training: Amount, divergence, or both (see Section 4.5.4).
- (2) conditionally apply pre-processing in the form of principal component analysis (PCA), in order to test a lower-dimensional input space $\mathcal{X}'$.
- (49) number of training samples (using between two and 50 samples for training).
- (50) repeats using a deterministically randomized dataset.

During each of the 50 repetitions, the entire data set is shuffled. Then, a number of training samples (m) are removed without replacement, which constitutes the training dataset $S = \{z_1, \ldots, z_m\}$. Then, a constant number of 50 validation samples is selected from the rest

PREVIEW

of the dataset. These samples are completely excluded from any training and only used to estimate the validation error using the root-mean-square error (RMSE) as a loss functional ℓ (8). The pre-processing pipeline consists of three steps: Removal of (near-)zero variance predictors, conditionally reducing dimensionality using PCA, and z-standardization of the data (center and scale). The pipeline is fitted to the training data and then applied to the validation data.

The above grid has 176,400 permutations. For each, a nested inner CV is performed. The empirical risk estimator used in the nested CV is the so-called *leave-one-out* cross-validation (LOOCV) [91]. It trains on all but the i-th instance of the training data. Therefore, it estimates the stability with respect to changes in the training set. This process shall be repeated for every $i \in \{1, \dots, m\}$ item in S, such that $S^{\setminus i}$ is the training set without that item (11). The associated R_{loo} estimator is the average over all m possible constellations of S (12). It is known as the estimator of *error stability*, which is used as a measure of the difference between true and empirical generalization error [92]. Since the training here uses all but one data point during LOOCV training, it should be noted that the empirical estimate of the generalization error has a slightly pessimistic bias [81].

$$S^{\setminus i} = \{z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_m\}, \tag{11}$$

$$R_{loo}(A, S) = \frac{1}{m} \sum_{i=1}^{m} \ell\left(A_{S^{\setminus i}}, z_i\right). \tag{12}$$

The inner CV is repeated for a fixed number of 25 folds during which a nested CV is performed for each. Each m-th estimate during LOOCV in addition is found by conducting a nested grid search for optimal model-specific hyperparameters, which typically span between $1e^1$ and $1e^3$ permutations. For the models *gbm* and *nnet*, fine-tuned grids and fewer inner repeats are used. The many-times repeated training allows us to approximate the probability density of V, its expectation $\mathsf{E}[V]$, and its associated confidence intervals.

5. Analysis and results

In this section, we report the results of the qualitative and quantitative analyses. Those results, their validity, and limitations are then further discussed in Section 6. The reported results represent our main findings and are those primarily relevant to the posed research questions. The results here are presented in the same order as the relevant methodology: First, we demonstrate results related to the qualitative evaluation. Then, results obtained quantitatively are shown from Section 5.4 and onwards. The interested reader is directed to an extensive technical report with numerous additional results that extend far beyond the scope of this study [78].

5.1. Inter-rater reliability and consensus

Prior to conducting a session for finding a consensus, we assessed the inter-rater agreement. This was done by first computing Gwet's *agreement coefficients* AC_1/AC_2 [66] and then benchmarking it against the scale as suggested by Landis and Koch [67]. Gwet's coefficient results in a percent agreement of $\approx 86.2\%$, a by-chance percent agreement of $\approx 56.8\%$

PREVIEW

(which it corrects for), an agreement coefficient (*Kappa*) of ≈ 0.681, a standard error (deviation) of ≈ 0.072, and a p-value of $\approx 2e^{-7}$. The p-value indicates, under any common significance level, that there is no practical corroboration for the null hypothesis of the inter-rater agreement test that the raters' agreement happened purely by chance. Instead, we accept its alternative hypothesis that chance did **not** cause the observed agreement [93]. According to the computed benchmark, the agreement is, for the largest portion of $\approx 82.18\%$, "substantial" [67]. Another $\approx 12.88\%$ of the agreement is "moderate", and some of it ($\approx 4.94\%$) is even regarded as "almost perfect." This is visualized in Figure 3.

Assessing inter-rater agreement addresses **RQ1.1** directly. The computed benchmark indicates that most of the agreement is substantial. However, in order to solidify our answer, we also compute the benchmarks of Cicchetti and Sparrow [94], Fleiss [95], and Regier *et al.* [96]. Note that the latter uses the same (positive) levels as the benchmark by Landis and Koch, but only changes the labels (from "Moderate" to "Fair", from "Substantial" to "Very Good", and from "Almost Perfect" to "Excellent"). For Cicchetti and Sparrow's benchmark, the agreement is "Fair" ($[0.4, 0.6] \approx 12.88\%$), "Good" ($[0.6, 0.75] \approx 70.14\%$), and "Excellent" ($[0.75, 1.0] \approx 16.98\%$). For Fleiss's benchmark, the agreement is "Intermediate to Good" ($[0.4, 0.75] \approx 83.02\%$), as well as "Excellent" ($[0.75, 1] \approx 16.98\%$).

Finding a consensus is different from only applying, say, a weighted average. While there was close or full agreement in many cases between the raters, some cases warranted for a common, retrospective inspection of the projects' artifacts, which led to ratings that were sometimes outside the range of the individual assessment. Most often, however, the raters were able to settle on one of the proposed ratings or a neighboring value.

5.2. Phenomenon prevalence and manifestation

Here, we report the summarized results of how the Fire Drill manifests across the projects, in terms of concretely observed empirical instances of certain symptoms and consequences. Those results address **RQ1.2** and were obtained by systematically analyzing the raters' notes. The most recent phenomenological descriptions of the Fire Drill define **seven**

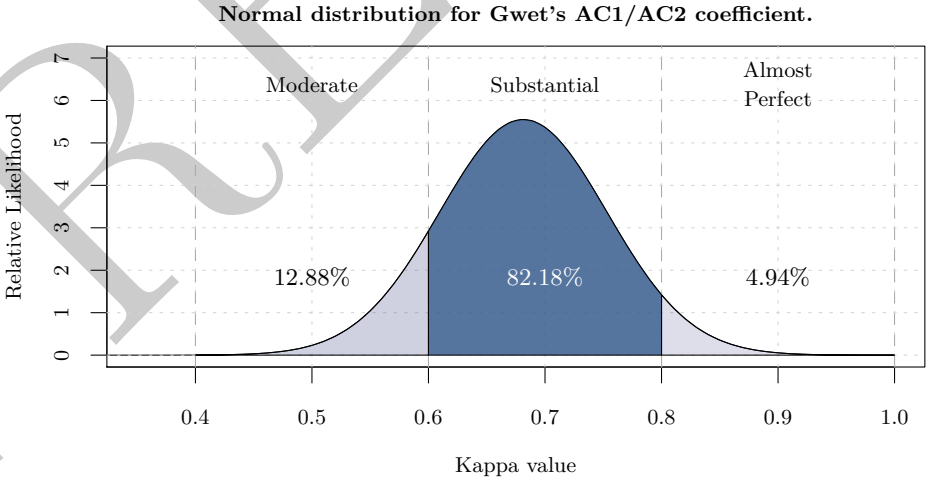

Figure 3. Gwet's AC_1/AC_2 coefficients [66] benchmarked against the Kappa scale of Landis and Koch [67]. The opacity indicates the probability for the agreement to be in the given range. The distribution's mean is at ≈ 0.681 and the standard deviation is ≈ 0.072

PREVIEW

supercategories for symptoms and consequences (see Appendix A). We refer to them using the notions SC1 through SC7. The qualitative evaluation led to us defining **three** additional supercategories. Since these were derived from empirical observations, we refer to them as ESC1, ESC2, and ESC3 (see Appendix C). An observation was only logged if the rater's notes allowed it. For example, some notes indicate a problem, without a cause: *"descope in later stages of the project,"* or *"poor testing"*. Here, we cannot assign an instance of a symptom/consequence. Also, if the severity cannot be decided between two levels, then the rater's Fire Drill severity is applied to reach a decision.

We should note that, due to the nature of the definition of SC7, no instances of it were found. Its description would affect a project rather globally. However, some of SC[1-6], as well as ESC[1-3] convey the portrayed problematic of SC7 in parts, so that findings were assigned to these instead. During the evaluation of the raters' notes, many instances emerged that could have been assigned to the original symptoms and consequences SC[1-7]. However, it became apparent that a new supercategory would perhaps be more suitable for many instances. Therefore, we introduced the notion of ESC[1-3]. Those reflect poor communication, high project risk, and poor usage of project management tools and methodologies, respectively. ESC[1-3] are not supposed to become part of the Fire Drill description. Rather, they were introduced for a more fine-grained assignment of observations to superordinated symptoms & consequences. Therefore, any observation assigned to ESC[1-3] could as well be assigned to the original symptoms & consequences.

Qualitative evaluation. Our findings indicate that the most reoccurring problem is high project risk (ESC2). It is closely followed by poor communication (ESC1) and a compromised project schedule or scope (SC6). The most infrequently observed problems (apart from SC7) are a spike of development efforts towards project end (SC3), the absence of sufficient quality assurance and project tracking during development (SC4), as well as the delivery of only analytical/documentational artifacts over a (too) long period (SC2).

Project risk (ESC2) is the most diverse supercategory, that is, it has the most different types of empirical observations by far (11). Risks are sometimes highly connected and emerge consecutively, such that the consequence of one problem is the symptom of the next problem. For example, an imbalance of activities (e.g., time spent on development when it had been required on requirements analysis instead) often leads to descoping, which itself caused frequent project schedule adaptations or quality regressions. Many problems can be attributed to the lack of (professional) experience in students, such as the misestimation of work items, technical difficulties (e.g., development environment, infrastructure, etc.), improper self-organization, or the (unwitting) misinterpretation of business requirements. However, not all problems were caused by students. For example, the customer representative was sometimes not available at the required capacity, thwarting the team. In other cases, the customer did not understand the technical challenges involved, which led to, e.g., unrealistic expectations or greatly diverging effort estimations.

While observations of the second-largest problem, poor communication (ESC1), could be attributed in many cases to project risk, many problems would go underappreciated if doing so. The most frequent observation was an unresponsive customer or unsatisfactory communication. It would often manifest through delayed, slow, or incomplete responses. Sometimes, critical material, decisions, or information were imparted too late, causing other problems, such as a stalled team or compromised schedule. This observation is perhaps the most prominent for one of the root causes of a Fire Drill: management stalls development. Poor communication can be mutual, that is, between parties, or be caused by just one party. Most of the problems can be attributed to the mutual category. However, there are

PREVIEW

also instances of students not renegotiating misunderstandings and of customers interfering with the students' project management without telling them.

The third-largest problem, a compromised project schedule or scope (SC6), had three different manifestations that were observed multiple times, both in the same project as well as across projects. Most often the schedule was compromised because the students accepted change requests, the re-prioritization of existing issues, or the addition of new issues in the middle of an iteration. This situation was often exacerbated by an improper change management process. The second-most frequent observation for this problem was that work was not completed as planned and the dragging of issues into the next iteration. The reasons for these observations were multifarious, such as an over-challenged team, misestimation, or unequal work distribution. Related to this observation, but still distinct from it, is that the team gets used to and regularly accepts doing overtime (and prolongs an iteration) or truncates scope (planned work).

As for problems of the remaining symptoms & consequences, the most frequent observations were related to a slow project startup (SC1), especially with regard to non-developmental activities. This was most often related to a familiarization process needed by the new team, such as familiarization with the other members, new or changed tools, ways of communication, or yet-to-be-improved early-stage procedures. Another common cause was an unclear scope, such that the team went into procrastination (underscoped) or did have a hard time finding their way into the project (overscoped). ESC3, the poor usage of project management tools and related methodologies, constituted the next bigger class of symptoms and consequences. Typical problems are attributed to the improper usage of ALM tools, such as using wrong item types (e.g., marking an Epic as a Task), not breaking down large deliverables and features, or careless and imprecise logging leading to a discrepancy between logged and done work. The teams were allowed to use additional tools for information and knowledge management, which led to confusion and unnecessary duplication. Lastly, SC5 concerns the final product, its quality, and delivery date. The quality was compromised in some cases by skipping over planned features or proper quality assurance. In some cases, the product was completed but delivered only after the final due date.

The refactored solution to a Fire Drill includes measures applicable for when there is time (iterations) and resources left. While some projects showed increasingly stronger signs of the phenomenon, no intermittent measures were implemented to alleviate the problems. It was only towards the very end that a solution had to be found. This is attributed to our context: While there are deliverables after each iteration, the customer is primarily interested in the final product, as feature-complete as possible. The students are primarily interested in passing the course, concentrating their focus on the final delivery, too. Refactored solutions to ameliorate an eventuated late-stage Fire Drill are to re-negotiate the delivery date, to triage the remaining budget into implementing missing features and quality assurance, or to descope. Due to the lack of monetary interests (and related pressure) on the customer's side and the impossibility of changing the delivery date (constraint of the context), the Fire Drill manifested predominantly through descoping, which proved to be a valuable predictor later.

Quantitative evaluation. We report on the average severity for each symptom & consequence, as well as on the total severity. The utility of the average severity is a ranking of the supercategories, that is, to determine what the common severity of the manifestation for each superordinate category is. The total severity is the sum of the observations' severity. Its utility is more specific for learning for our concrete case, as it should give an indication as to which symptom and consequence is the most pronounced in our projects.

PREVIEW

Figure 4 shows the average severity per symptom & consequence. `SC[2-5]` have only a single or few observations, which should be considered when ordering supercategories. `SC1` and `SC6` make for a good comparison, for example. While both have similarly many observations, the latter has a significantly stronger average severity. Encountering instances of either symptom & consequence might be similarly likely, but an observation of the latter indicates a worse case of a Fire Drill. The strongest average severity is exhibited by `ESC1` (disregarding `SC3` and `SC5` which have only a few observations) with a value of ≈ 2.78. Recall that a value of 3 corresponds to the ordinal level "Moderate" (see Section 4.4), which is characterized as clearly identifiable and reoccurring symptoms or direct corroboration. Since its average severity exceeds that of `SC6`, the poor communication between stakeholders comparatively appears to be the worst of all the problems in our case.

This impression slightly shifts towards `ESC2` when looking at the total observed severity (see Figure 5). Ten more instances (28 total) were observed of `ESC2`, which explains its high total severity. In other words, while poor communication is comparatively worse when it happens, the projects were substantially more often subject to some form of risk. A third, quite evident problem is a compromised schedule or scope (`SC6`). By a larger margin, this is followed by problems related to a longer stall during the project beginning (`SC1`), a final product with low quality (`SC5`), and poor usage of ALM tools (`ESC3`). The low count of observations for `SC[2-5]` also results in a comparatively low total severity.

5.3. Phenomenon absence

The evaluation of the raters' notes also exhibited evidence for the absence of a Fire Drill for a number of projects. In Appendix D, we have gathered a list of observed symptoms and consequences in order to answer **RQ1.3**. This list is comprised of symptoms and consequences that are *counter-indicative* to what constitutes a Fire Drill. While the absence of evidence is not evidence of absence, the gathered items should be more regarded as

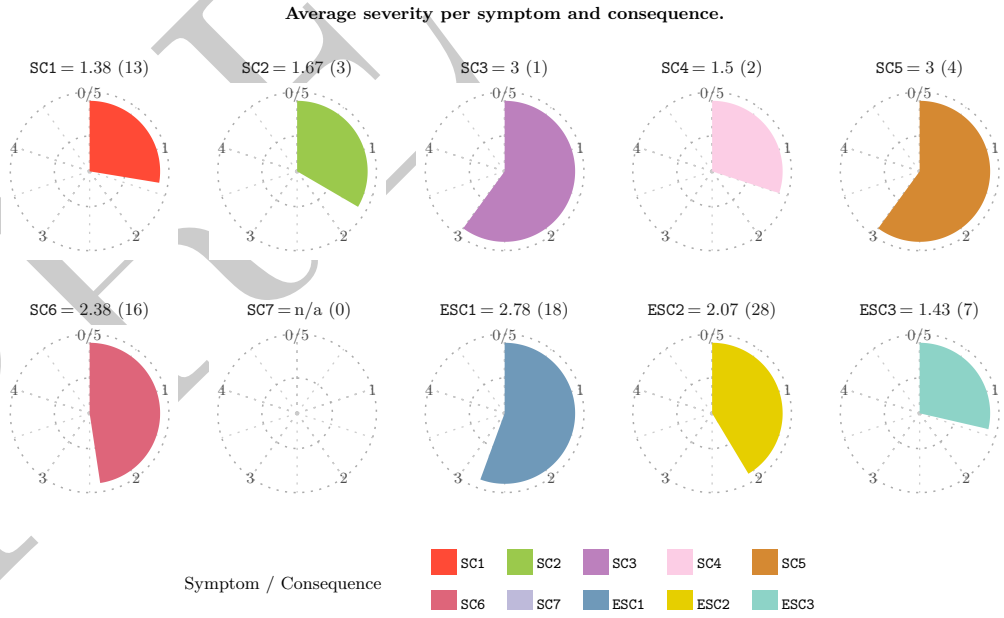

Figure 4. Average severity for each (empirical) Symptom/Consequence. Also shown is the number of observed instances in parentheses

PREVIEW

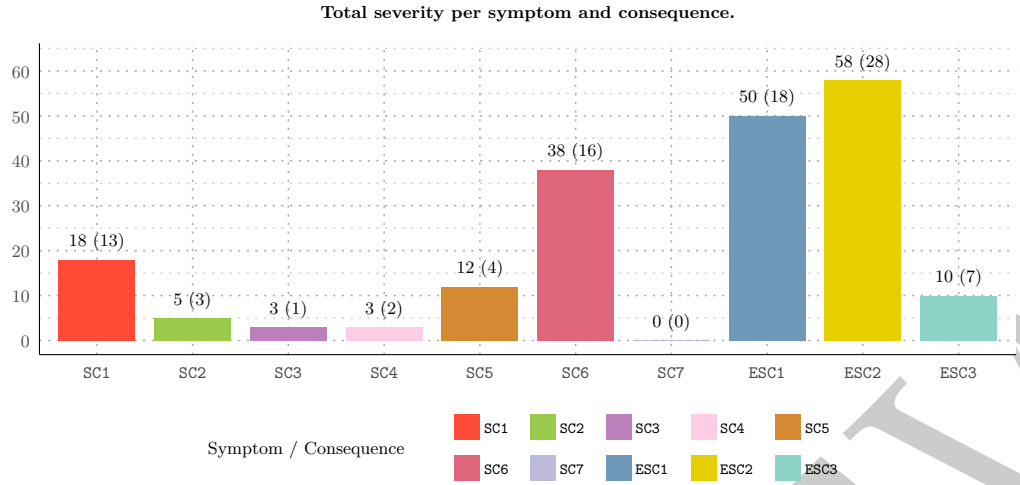

Figure 5. Total severity for each (empirical) symptom/consequence. Also shown is the number of observed instances in parentheses

true negatives. This list does not claim to be complete, nor is the presence of a single item (or few) sufficient proof for the phenomenon's absence. On the contrary, evaluation of the raters' notes indicates that a project may exhibit symptoms & consequences for and against a Fire Drill, even simultaneously. For example, communication and collaboration with the customer might be seamless (as indicated by the counter-indicative symptom & consequence `CISC01`), but the project's schedule might still get compromised due to misestimation caused by inexperience. Sometimes, there are also signs for the opposite of a Fire Drill (or other patterns). Our main observation here is essentially related to an underchallenged or underutilized team, and often related to the quick completion of work items, delivery before the deadline, a too-simple product, or skipping over quality assurance or planning and/or analysis with the direct start of implementation.

5.4. Quantitative phenomenon manifestation

The previously won ground truth allows leveraging the quantitative type-II data. With each project having a severity attached, we can find typical accumulations of (maintenance) activities that are characteristic of a Fire Drill (**RQ2.1**). Figure 6 shows, for each activity as it is found in source code and issue-tracking data, a weighted mixture (convex combination (2) using the ground truth as weight). Three out of 15 projects had a ground truth consensus of 0 and, therefore, do not contribute to any of the weighted mixtures. Another effect of this circumstance is that the weighted mixtures quite obviously mirror the activities from a few, severely affected projects. Since each activity's mixture is an ordinary probability distribution (with integral $= 1$), we can compare them in a straightforward manner.

Source code. The activities in the source code are derived from maintenance activities (adaptive $\sim$ A, corrective+perfective $\sim$ CP) [52]. The overall commit frequency, regardless of the associated activity, is depicted as FREQ. Adaptive activities show a somewhat slow start, followed by a peak in the third quarter of the project, and a sharp decline afterward. To some degree, A follows the expectations according to the Fire Drill's phenomenological descriptions. CP and FREQ on the other hand, steadily increase for the first $\approx 80\%$ of the project, only to peak shortly after at about $\approx 90\%$ of the project time. The combined

PREVIEW

behavior of A and CP is counter-intuitive to the expected behavior of these activities, as per the Fire Drill's phenomenological descriptions (i.e., a significant increase of adaptive activities with a simultaneous decrease in corrective+perfective activities). FREQ, on the other hand, distinctively shows the expected peak of a Fire Drill towards the project end.

Issue-tracking. The issue-tracking activities provide us with an additional quantitative perspective. Due to the nature of the projects, DEV is expected to only reflect adaptive engineering, similar to source code's A. The qualitative evaluation confirms this expectation, because corrective and perfective activities were rarely, if at all, logged in the project management tools. Indeed, we observe a quite similar temporal accumulation between A and DEV, with a peak in the third quarter as well. More or less inversely proportional to DEV is the accumulation of activities as captured by REQ. Both of these activities are in accordance with the existing phenomenological descriptions, such as an imbalance where in the beginning of a project activities connected to requirements, analysis, and planning prevail, while development is thwarted for one or the other reason. Approximately uniform distributions for these two activities would be asymptomatic for a Fire Drill in the ideal case. The descoping activity DESC was designed by us based on the assumptions that it would make for a good predictor of the Fire Drill. We observe that this activity steadily – in fact almost linearly – accumulates from project start to end. This indicates that affected projects are subject to descoping from the beginning and that these projects do not manage to remedy this situation.

5.5. Variable importance

Another question we sought to answer quantitatively using type-II data was about the importance of activities and project phases for accurately predicting the phenomenon severity (**RQ2.2**). For that, we subdivided the normalized relative project time into ten equally long, consecutive segments (see Section 4.5.3). Figure 7 shows a graphical result of this. The exact numeric results as shown in the figure are included in Table E1 and Table E2, which are to be found in Appendix E. Looking at the final numbers, we can say that the divergence features have greater variable importance than the amount features,

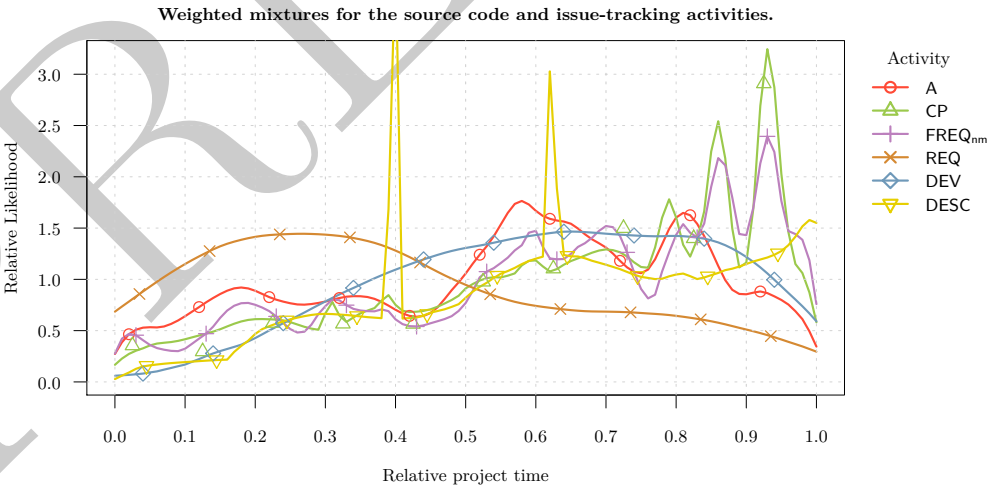

Figure 6. Weighted mixtures for each of the source code and issue-tracking activities (using the ground truth to create convex combinations)

PREVIEW

both for source code (≈52.5%) and even more so for issue-tracking (≈56.2%). The two most important features for either data source are A|0 CP (19%) and REQ|0 DESC (≈20.7%). The two least important ones are A (≈14.2%) and REQ (≈13%).

As for the per-segment importance, the results diverge between data sources, with a larger variance for source code. There, it ranges from ≈6.2% to ≈15.3%, with a standard deviation of ≈3.3%. The first five segments, that is, the first half of the project, account for ≈55.9% of all importance, where segments three and four seem to be particularly important. Interestingly, segments seven, nine, and ten exhibit a comparatively low variable importance for predicting phenomenon severity. The range for issue-tracking data is from ≈5.2% to ≈13.5%, with a lower standard deviation of ≈2.3%. Both project halves are equally important when using issue-tracking data. Except for segments seven and nine, each segment is almost equally important.

5.6. Adaptive training

The third set of results is related to the third set of propositions and hypotheses and addresses the research questions therein. According to the previously described methodology (see Section 4.8), we conduct the adaptive training, using a large number of constellations and repeats in order to obtain robust estimates. We find that models trained using source code data converge faster and result in a lower final training error (answer to **RQ3.1**), that not reducing dimensionality using PCA works better for a slight majority of cases, and that using both kinds of features simultaneously leads to robust convergence. Figure 8 shows the distribution of validation errors for all six models when trained on 20 instances. When we compare the data sources, models trained on source code achieve a lower mean (0.75/0.92), median (0.52/0.71), minimum (0.152/0.158), and standard deviation (0.58/0.65) for the validation RMSE error, summarized across all the different constellations.

The champion model is a standard feed-forward neural network that uses divergence features from source code only. This is also reflected in Figure 8, however, here it is only

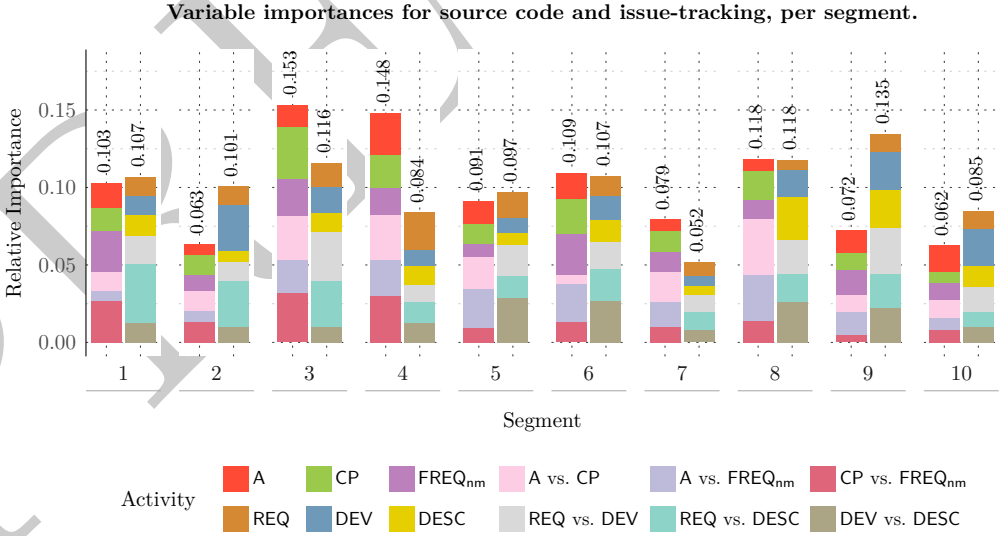

Figure 7. Per-segment and per-feature variable importances as averaged across five models, each repeated 100 times: Boosted Generalized Additive Model, Neural network, (Generalized Linear) Partial Least Squares, and Bagged CART

PREVIEW

shown for 20 instances. The full adaptive training of the champion model is shown in Figure 9. The many observed validation errors obtained through the numerous repeats are almost always normally distributed, indicating the robustness of the final model. That also allowed us to apply the Three Sigma rule (which has the tightest bounds) to determine a confidence interval per number of training instances and to find some answers to **RQ3.2**. Most notably, training the champion model on 25 instances produces an expected validation error of ≈ 0.46, and this error will not be larger than ≈ 0.96 by a probability of 99.7%[10]. In other words, such a model will predict the Fire Drill severity on a scale of $0-10$, and by doing so it will almost surely not deviate by more than one point on this scale (the expected deviation is even less than 0.5). The expected RMSE falls below 1.0 for the first time when trained on twelve instances. For 15 instances, the probability that the prediction will be off by maximally 1 is already at $\geq 76\%$, as can be seen from the figure. For 20 instances, this probability increases towards $\geq 92\%$. For 50 instances, the expected error plus deviation is 0.52 or less, by a probability of 99.7%.

6. Discussion

In this section, we first summarize all results and relate them to the previously established propositions and hypotheses. We then discuss the validity of our study and its results, its limitations, and address replicability and generalizability.

6.1. Summary of the results

We have shown that experts can independently agree on a subjectively chosen rating and that their agreement was substantial and did not occur by chance. This was the basis for the propositions **Pr1.1** and **Pr1.2**. The existing phenomenological descriptions

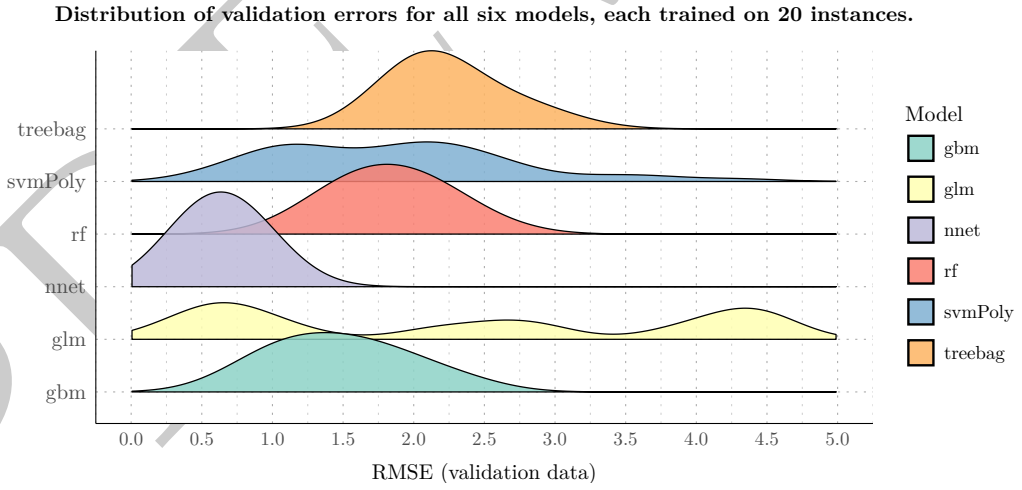

Figure 8. Distribution of validation errors (50 instances) computed using the root-mean-square error for all six models, where each was repeatedly trained on 20 instances of the source code dataset, without applying PCA

[10]While these and the following results can be seen in Figure 9, they are calculated exactly using the previously introduced notions for probabilities (9) and deviations (10) of confidence intervals, respectively.

PREVIEW

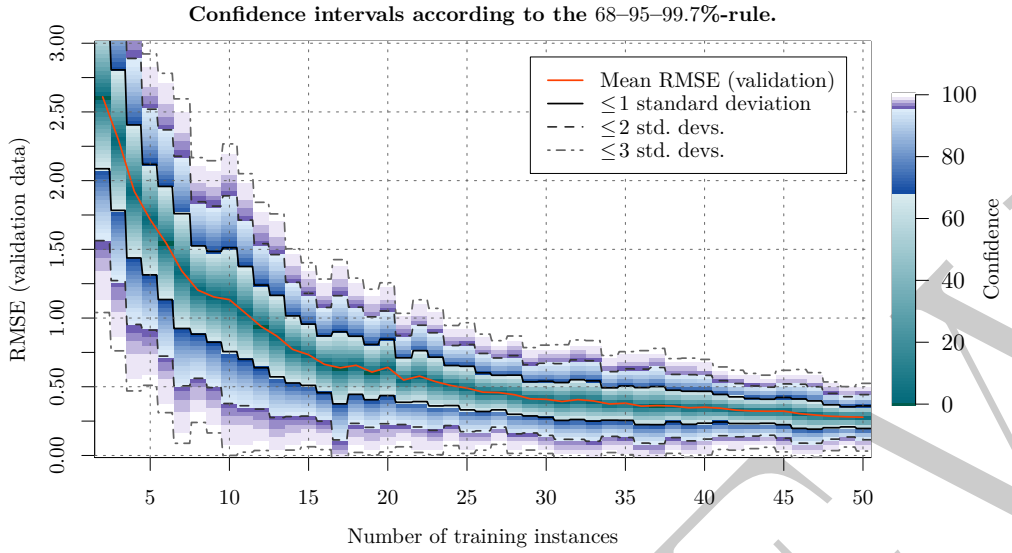

Figure 9. Continuous confidence of the neural network predictor, with regard to the number of training instances. Shown are the values according to the 68–95–99.7%-rule (assuming a normal distribution for every generalization error). The mean RMSE was determined using 50 models' predictions on validation data. The three color gradients correspond to the three sigmas

of Fire Drill are sufficiently precise, and its severity can be accurately determined. It also confirmed the hypothesis **Hyp1.1** that the mostly unstructured type-I data provide sufficient quality for the assessment task. The agreement on the absence was also reliable (**Pr1.3** and **Hyp1.2**) so we were able to gather a list of counter-indicative symptoms and consequences. Most notably, however, the qualitative assessment and the following quantitative evaluation of the Fire Drill's manifestation allowed us to identify the most pronounced and prevalent symptoms and consequences in the considered projects. We find prime examples of reoccurring Fire Drill elements across our projects, such as instances of (chains of) project risk, poor stakeholder communication, and regressions in the form of descoping and compromised schedules. The circumstances and constraints of the studied context allow us to deduce explanations for the concrete manifestation.

For the first quantitative analysis, we compose a weighted average of how activities typical for a Fire Drill-affected project accumulate. It is deemed an appropriate representation since we previously confirmed hypothesis **Hyp2.1**, that is, activities will display a behavior characteristic of or conforming to the Fire Drill's phenomenological descriptions. These allow us to establish an additional *quantitative* understanding of the phenomenon. We take advantage of the wide range of digital artifacts and reliable ground truth to do so (**Pr2.1**). The weighted mixtures unveiled some unexpected results, which are, however, explained by the confines of the studied context and case and corroborated by findings from our qualitative analysis. Late-stage Fire Drills were most often addressed by descoping, which is a valid – here the only valid – *refactored* solution. This is reflected quite well in the activities found in the source code. Instead of rushing to add the last missing features, the focus is predominantly on corrective activities and then on perfective activities. We know this from decomposing CP into its separate activities. While the Fire Drill description to date does not differentiate between corrective (C) and perfective (P), using source code allowed us to gain this insight. This is surprising since the delivery of agreed-upon requirements is

PREVIEW

deliberately prioritized over maintenance in these projects. An explanation for this might be that it became apparent in some of these projects that some form of compromise of scope is inevitable, so the efforts were likely re-focused on ensuring that a minimum quality corresponding to the requirements, for the to-date existing features, was met, or that the product works at all.

The second analysis of variable importance unveiled two principal findings. First, knowing about the (im-)balance of two activities is a more important predictor for Fire Drill severity than knowing how much time was spent on the activities. This is an important finding, as the balance between two activities is a relative metric. The phenomenological descriptions, as well as the results of our qualitative analysis, both support this. This result means that, at least on average, it is of greater relevance to know how diverging any two activities are, rather than their concrete amount (cumulative probability). Rather than waiting until the project end to calculate the amount of each activity in each segment, we can observe how activities diverge in a single segment intermittently and use this as an indicator in the future. The second important finding is that segments (project phases) are, except for a few cases, almost equally important (also, no phase has (near-)zero importance, **Hyp2.2**).

The third analysis was designed to determine whether a robust, reliable, and *low-risk* predictive model can be obtained. The designed adaptive training workflow showed that we can obtain a suitable model from those data (**Pr3.1**, **Hyp3.1**). With sufficient confidence as our stability criterion, we demonstrate that models trained on as few as 15 instances will completely automate severity assessment in the future, within acceptable confidence intervals (**Pr3.2**). Both sources of data individually, source code or issue-tracking, are suitable for this task (**Hyp3.2**). Generally, we observe that the adaptive training flow converges nicely with increasing amounts of training data. Clearly, the stability of the model is closely related to the amount of available training data. As for **RQ3.2**, the precise threshold for *stable* depends on the application, but we figure that to predict the severity of the Fire Drill in our case and context, a model trained on 15 to 25 instances would deliver sufficient stability, as the results would not be misleading, even when off slightly.

6.2. Validity, limitations, replicability, and generalizability

The choice to conduct an embedded case study was a natural one, given the principal objective and the previous pilot study [51]. Our study is of longitudinal character, as the same course was studied repeatedly and projects were selected over a duration of three years. Each project is unique with regard to individuals, groups, social structure, software, etc., and it is unlikely that the same set of events unfolds again in the same way [97]. Therefore, to maximize reliability and minimize bias, projects were selected from each year. Instead of conducting a case study for each project, we chose a common case and context that allows us to collect and analyze quantitative data as well, especially since we intended to backpropagate the gained knowledge to the phenomenological descriptions and, therefore, the studied case.

A case study should be chosen when conducting an empirical investigation of a contemporary phenomenon in its real-life context, especially when the phenomenon cannot be clearly separated from its context [55]. Furthermore, a variety of data sources is required (see Section 1.1). The Fire Drill is a phenomenon that is *always* embedded in some (social) context it cannot be separated from, and it is not the goal of this study to propose or attempt a separation. Instead, we have made extensive efforts to minimize the impact our study

PREVIEW

may have. In order to maximize the degree to which this study can be (partially) replicated, we outline an extensive protocol (see Section 4) and publish all original data [62] and experimental designs [78]. Construct validity is achieved by a variety of measures, such as the usage of multiple data sources and observers (triangulation), ascertaining of inter-rater reliability, and controlled experiments (e.g., many repeats, model averaging, training with stability criterion, etc.). Independent raters and an assessor use systematic protocols, which allowed us to suggest and confirm chains of causality and ensure repeatability and replicability. The usage of two completely independent data sources (source code and issue-tracking) provided a second perspective that we exploited to corroborate our findings.

The limitations of the obtained results lie in the external validity and generalizability. Results such as the predictive model, weighted mixtures, or variable importance do not have validity outside the studied context, as it introduced constraints that force the phenomenon to take certain turns. We have observed a great number of concrete instances of symptoms and consequences. Although extensive, these observations cannot be exhaustive, nor can their distribution be representative outside our context. This is similarly true for the asymptomatic, counter-indicative observations. The Fire Drill shares similarities and indicators with other kindred phenomena, which are also based on temporal accumulations of activities (see Section 3.1.2). Therefore, we expect the external validity of the protocol suggested for studies of these phenomena. The generalizability of this study comes from subsequent and replicating case studies. For example, one study may examine the pattern "Half Done is Enough" in the same context and another one the Fire Drill in an industrial context. Only with a sufficient amount of case studies will we be able to reach a definitive understanding of the Fire Drill and how it does (not) manifest in certain contexts.

7. Conclusions and future work

We have shown that activity-based detection of complex phenomena is viable and can be used to reduce the otherwise required amount of expert-based, qualitative, and extensive analyses. Our work has several practical implications. First, instances of maintenance activities are plentifully found in typical software projects and make for highly cost-effective and robust features. Second, an existing ground truth can be leveraged to make sense out of these quantitative features. Third, predictive models using these features require only a few observations (projects) to produce low-risk predictions. Last, A well-trained model can produce predictions that are accurate enough such that they can support or (partially) replace the expensive, error-prone, and expert-based evaluations. The practical implication is that such a model can be reused on only the quantitative data (e.g., we have shown that the commits of the repository are sufficient) of future projects for predicting the severity of complex phenomena. This may be useful in circumstances where a fast and computationally cheap analysis is required to, e.g., quickly filter and flag (for a full follow-up in-depth qualitative analysis) projects affected by a problem.

7.1. Synthesis

For an embedded case study, it is important to synthesize all results and to backpropagate them to the studied case [25]. We have presented results from each project's individual, qualitative, and idiographic study, as well as results from the quantitative, nomothetic analysis across all projects (see Section 5).

PREVIEW

We studied the Fire Drill as is embedded within a software engineering course. For this specific case and its surrounding context, we predominantly find results that are in agreement with the phenomenological descriptions. The in-depth individual study of each project, as well as a common evaluation across projects, allowed us to find explanations for all the results diverging from it. In affected projects, we observe typical peculiarities of a Fire Drill, such as management that stalls development, or late-stage rushes. While the anti-pattern suggests renegotiating the final deadline as one solution, the students truncated the scope of their applications to mitigate the fallout, thereby implementing the only valid refactored solution within the course. We gather the most significant symptoms and consequences of a Fire Drill within the course. Evaluation of total and average severity showed that project risk, poor communication, and a compromised project schedule or -scope are the biggest problems that students encounter. We learn that a Fire Drill may affect a project as a whole, but we also observe micro instances affecting single iterations. It was previously suggested and now confirmed by us that the Fire Drill is an anti-pattern that can be the result of, encompass, or cause other, often conceptually smaller anti-patterns such as "Analysis Paralysis" or "Cart Before The Horse". We find evidence that is counter-indicative of a Fire Drill and observe projects that exhibit evidence both for and against its presence, simultaneously. Raters are able to identify this and other circumstances by being provably able to confidently agree on a severity. Our conjecture that descoping makes for a strong predictor is confirmed; projects are affected by it over the course of their entire lifecycle and the amount of time wasted on it consistently increases. We conclude that the Fire Drill is an anti-pattern that was deliberately described vaguely, but that it is still possible to derive concrete and specific problem instances from it. Observing only a few instances proved sufficient for building low-risk predictive models that can exploit activities that are modeled after either source code or issue-tracking data, as both kinds of data sources are eligible for the task.

We suggest that other phenomena that are characterized by activities that can be captured and analyzed similarly can be subjected to the methodology presented in this work. Therefore, we intend for the methodology to be the main contribution. Our empirical observations are likely valid in other similar cases and contexts, too, but subsequent (partially) replicating case studies will have to show this. Most patterns today are only described from anecdotes or other literature (phenomenological descriptions), but rarely come with a set of empirical observations attached. We contribute directly to the existing understanding of the Fire Drill by unearthing concrete empirical instances associated with its ascribed symptoms and consequences (the supercategories). More significantly, though, we have shown how to use a (scarce) ground truth to establish an additional, quantitative understanding by leveraging the available data (which was not possible previously), making the Fire Drill perhaps the first pattern-like phenomenon that is described from both perspectives. Lastly, having uncovered the most frequent and prevalent issues that participants of the course encounter, we will attempt to incorporate the won insights into future editions of the course to improve all participants' experiences.

7.2. Future work

This study is the first to properly replicate the significant findings from the previous pilot study [51]. We intend to replicate this study with other related and similar phenomena. Candidates are, for example, Cart Before the Horse, Half Done is Enough, the Net-negative Producing Programmer, the Lone Wolf, Nine Pregnant Women (a variant of Brook's law),

PREVIEW

or Analysis Paralysis. We also encourage conducting (partial) replication studies with the same or a different phenomenon and alterations to the context, especially in industrial settings. It might also be worthwhile to consider additional artifacts found in the application lifecycle management data for data triangulation. Additional analyses, such as an earned value analysis, might provide additional corroboration, especially when its result can be correlated with the maintenance activities. Subsequent studies that analyze the Fire Drill will contribute towards a more complete picture of the phenomenon. Studies using other phenomena will also help to increase the margin between phenomena, making them more distinguishable from each other. Once a sufficient number of case studies were conducted, a multiple case study should be performed that summarizes all findings.

7.3. Acknowledgments

We would like to express our gratitude towards raters two and three, who helped to find a ground truth. We acknowledge the support of Linnaeus University's Centre for Data Intensive Sciences and Applications (DISA) and the Swedish Research School of Management and IT (MIT). This work was supported by the European structural and investment funds (ESIF) project CZ.02.1.01/0.0/0.0/17_048/0007267 (InteCom). We express our gratitude towards the anonymous reviewers who provided thorough and constructive feedback that allowed us to considerably improve our work.

References

[1] C. J. Neill, P. A. Laplante, and J. F. DeFranco, *Antipatterns: Managing Software Organizations and People*, 2nd ed. Auerbach Publications, 2011.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language – Towns, Buildings, Construction*. Oxford University Press, 1977.

[3] W. H. Brown, R. C. Malveau, H. W. McCormick III, and T. J. Mowbray, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc., 1998.

[4] P. A. Laplante and C. J. Neill, *Antipatterns: Identification, Refactoring, and Management* (Auerbach Series on Applied Software Engineering), 1st ed. CRC Press, Auerbach Publications, 2005, 336 pp.

[5] L. Simeckova, P. Brada, and P. Picha, "SPEM-based process anti-pattern models for detection in project data," in *46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, Portoroz, Slovenia, August 26–28, 2020*, IEEE, 2020, pp. 89–92.

[6] I. Stamelos, "Software project management anti-patterns," *Journal of Systems and Software*, vol. 83, no. 1, pp. 52–59, 2010.

[7] R. R. Nelson, "It project management: Infamous failures, classic mistakes, and best practices," *MIS Quarterly Executive*, vol. 6, no. 2, 2008. [Online]. Available: https://aisel.aisnet.org/misqe/vol6/iss2/4.

[8] R. S. Kenett and E. R. Baker, *Software Process Quality: Management and Control* (Computer Aided Engineering New York, N.Y., 6.), 1st ed. Marcel Dekker Inc., 1999.

PREVIEW

[9] C. P. Halvorsen and R. Conradi, "A taxonomy to compare SPI frameworks," in *Software Process Technology, 8th European Workshop, EWSPT 2001, Witten, Germany, June 19–21, 2001, Proceedings*, V. Ambriola, Ed., ser. Lecture Notes in Computer Science, vol. 2077, Springer, 2001, pp. 217–235.

[10] A. Birk, T. Dingsoyr, and T. Stalhane, "Postmortem: Never leave a project without it," *IEEE Software*, vol. 19, no. 3, pp. 43–45, 2002.

[11] W. J. Brown, H. W. McCormick III, and S. W. Thomas, *AntiPatterns in Project Management*. John Wiley & Sons, Inc., 2000.

[12] P. Silva, A. M. Moreno, and L. Peters, "Software project management: Learning from our mistakes," *IEEE Software*, vol. 32, no. 3, pp. 40–43, 2015.

[13] A. Nizam, "Software project failure process definition," *IEEE Access*, vol. 10, pp. 34 428–34 441, 2022.

[14] P. Brada and P. Picha, "Software process anti-patterns catalogue," in *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019, Irsee, Germany, July 3–7, 2019*, T. B. Sousa, Ed., ser. EuroPLoP'19, ACM, 2019, 28:1–28:10.

[15] P. G. F. Matsubara, B. F. Gadelha, I. Steinmacher, and T. U. Conte, "SEXTAMT: A systematic map to navigate the wide seas of factors affecting expert judgment software estimates," *Journal of Systems and Software*, p. 111 148, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221002429.

[16] F. U. Muram, B. Gallina, and L. G. Rodriguez, "Preventing omission of key evidence fallacy in process-based argumentations," in *11th International Conference on the Quality of Information and Communications Technology, QUATIC 2018, Coimbra, Portugal, September 4–7, 2018*, A. Bertolino, V. Amaral, P. Rupino, and M. Vieira, Eds., IEEE Computer Society, 2018, pp. 65–73.

[17] P. Picha, P. Brada, R. Ramsauer, and W. Mauerer, "Towards architect's activity detection through a common model for project pattern analysis," in *2017 IEEE International Conference on Software Architecture Workshops, ICSA Workshops 2017, Gothenburg, Sweden, April 5–7, 2017*, IEEE Computer Society, 2017, pp. 175–178.

[18] P. Picha and P. Brada, "Software process anti-pattern detection in project data," in *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019, Irsee, Germany, July 3–7, 2019*, T. B. Sousa, Ed., ser. EuroPLoP'19, ACM, 2019, 20:1–20:12.

[19] D. Settas, S. Bibi, P. Sfetsos, I. Stamelos, and V. C. Gerogiannis, "Using bayesian belief networks to model software project management antipatterns," in *Fourth International Conference on Software Engineering, Research, Management and Applications (SERA 2006), 9–11 August 2006, Seattle, Washington, USA*, IEEE Computer Society, 2006, pp. 117–124.

[20] D. Settas and I. Stamelos, "Using ontologies to represent software project management antipatterns," in *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2007), Boston, Massachusetts, USA, July 9–11, 2007*, Knowledge Systems Institute Graduate School, 2007, pp. 604–609.

[21] M. B. Perkusich, G. Soares, H. O. Almeida, and A. Perkusich, "A procedure to detect problems of processes in software development projects using bayesian networks," *Expert Systems with Applications*, vol. 42, no. 1, pp. 437–450, 2015.

[22] N. E. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and M. Tailor, "Making resource decisions for software projects," in *26th International Conference on Software Engi-*

PREVIEW

neering (ICSE 2004), 23–28 May 2004, Edinburgh, United Kingdom, A. Finkelstein, J. Estublier, and D. S. Rosenblum, Eds., IEEE Computer Society, 2004, pp. 397–406.

[23] M. Unterkalmsteiner, T. Gorschek, A. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement—a systematic literature review," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 398–424, 2012.

[24] J. J. P. Schalken, S. Brinkkemper, and H. van Vliet, "Using linear regression models to analyse the effect of software process improvement," in *Product-Focused Software Process Improvement, 7th International Conference, PROFES 2006, Amsterdam, The Netherlands, June 12–14, 2006, Proceedings*, J. Münch and M. Vierimaa, Eds., ser. Lecture Notes in Computer Science, vol. 4034, Springer, 2006, pp. 234–248.

[25] R. K. Yin, *Case Study Research: Design and Methods* (Applied Social Research Methods), 5th ed. SAGE Publications, 2013.

[26] R. W. Scholz and O. Tietje, *Embedded Case Study Methods: Integrating Quantitative and Qualitative Knowledge*, 1st. SAGE Publications, Inc, 2001.

[27] T. Shaikhina, D. Lowe, S. Daga, D. Briggs, R. Higgins, and N. Khovanova, "Machine learning for predictive modelling based on small data in biomedical engineering," *IFAC-PapersOnLine*, vol. 48, no. 20, pp. 469–474, 2015, 9th IFAC Symposium on Biological and Medical Systems BMS 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896315020765.

[28] Y. Zhang and C. Ling, "A strategy to apply machine learning to small datasets in materials science," *npj Computational Materials*, vol. 4, no. 1, p. 25, May 2018.

[29] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering – Guidelines and Examples*. Wiley, 2012. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118104358.html.

[30] K. L. Gwet, *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters*, 4th ed. Advanced Analytics, Sep. 2014.

[31] E. Tüzün, H. Erdogmus, M. T. Baldassarre, M. Felderer, R. Feldt, and B. Turhan, "Ground-truth deficiencies in software engineering: When codifying the past can be counterproductive," *IEEE Software*, vol. 39, no. 3, pp. 85–95, 2022.

[32] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of Machine Learning Research*, vol. 2, pp. 499–526, 2002. [Online]. Available: http://jmlr.org/papers/v2/bousquet02a.html.

[33] K. Scott, *The unified process explained*, en, 1st ed. Boston, MA: Addison Wesley Professional, Nov. 2001.

[34] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP* (Addison-Wesley object technology series), en. Boston, MA: Addison-Wesley Educational, Apr. 2003, 464 pp.

[35] W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*, 3rd ed. Houghton Mifflin Company, 2002.

[36] J. M. Verner, J. Sampson, V. Tosic, N. A. A. Bakar, and B. A. Kitchenham, "Guidelines for industrially-based multiple case studies in software engineering," in *Proceedings of the Third IEEE International Conference on Research Challenges in Information Science, RCIS 2009, Fès, Morocco, 22–24 April 2009*, A. Flory and M. Collard, Eds., IEEE, 2009, pp. 313–324.

[37] R. Zhu, D. Zeng, and M. R. Kosorok, "Reinforcement learning trees," *Journal of the American Statistical Association*, vol. 110, no. 512, pp. 1770–1784, 2015, PMID:26903687. eprint: https://doi.org/10.1080/01621459.2015.1036994.

PREVIEW

[38] D. Draheim and L. Pekacki, "Process-centric analytical processing of version control data," in *6th International Workshop on Principles of Software Evolution (IWPSE 2003), 1–2 September 2003, Helsinki, Finland*, IEEE Computer Society, 2003, p. 131.

[39] R. Ramsauer, D. Lohmann, and W. Mauerer, "Observing custom software modifications: A quantitative approach of tracking the evolution of patch stacks," in *Proceedings of the 12th International Symposium on Open Collaboration, OpenSym 2016, Berlin, Germany, August 17–19, 2016*, A. I. Wasserman, Ed., ACM, 2016, 4:1–4:4.

[40] D. A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman, "Discovering community patterns in open-source: A systematic approach and its evaluation," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1369–1417, 2019.

[41] S. Z. Talpová and T. Čtvrtníková, "Scrum anti-patterns, team performance and responsibility," *International Journal of Agile Systems and Management*, vol. 14, no. 1, p. 170, 2021.

[42] A. Hachemi, "Software development process modeling with patterns," in *WSSE 2020: The 2nd World Symposium on Software Engineering, Chengdu, China, September 25–27, 2020*, ACM, 2020, pp. 37–41.

[43] T. Frtala and V. Vranic, "Animating organizational patterns," in *8th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015, Florence, Italy, May 18, 2015*, A. Begel, R. Prikladnicki, Y. Dittrich, C. R. B. de Souza, A. Sarma, and S. Athavale, Eds., IEEE Computer Society, 2015, pp. 8–14.

[44] A. H. M. ter Hofstede, C. Ouyang, M. L. Rosa, L. Song, J. Wang, and A. Polyvyanyy, "APQL: A process-model query language," in *Asia Pacific Business Process Management – First Asia Pacific Conference, AP-BPM 2013, Beijing, China, August 29–30, 2013. Selected Papers*, M. Song, M. T. Wynn, and J. Liu, Eds., ser. Lecture Notes in Business Information Processing, vol. 159, Springer, 2013, pp. 23–38.

[45] J. Roa, E. Reynares, M. L. Caliusco, and P. D. Villarreal, "Towards ontology-based anti-patterns for the verification of business process behavior," in *New Advances in Information Systems and Technologies – Volume 2 [WorldCIST'16, Recife, Pernambuco, Brazil, March 22–24, 2016]*, ser. Advances in Intelligent Systems and Computing, Á. Rocha, A. M. R. Correia, H. Adeli, L. P. Reis, and M. M. Teixeira, Eds., vol. 445, Springer, 2016, pp. 665–673.

[46] A. Awad, A. Barnawi, A. Elgammal, R. E. Shawi, A. Almalaise, and S. Sakr, "Runtime detection of business process compliance violations: An approach based on anti patterns," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13–17, 2015*, R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong, Eds., ACM, 2015, pp. 1203–1210.

[47] T. O. A. Lehtinen, M. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius, "Perceived causes of software project failures – an analysis of their relationships," *Information and Software Technology*, vol. 56, no. 6, pp. 623–643, 2014.

[48] L. Rising and N. S. Janoff, "The scrum software development process for small teams," *IEEE Software*, vol. 17, no. 4, pp. 26–32, 2000.

[49] P. G. Smith and D. G. Reinertsen, *Developing Products in Half the Time: New Rules, New Tools*, en, 2nd ed. Nashville, TN: John Wiley & Sons, Oct. 1997.

[50] F. P. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*, 2nd ed. Boston, MA: Addison-Wesley Longman, Aug. 1995.

PREVIEW

[51] P. Picha *et al.*, "Process anti-pattern detection in student projects – a case study," in *Proceedings of the 27th European Conference on Pattern Languages of Programs, EuroPLoP 2022, Irsee, Germany, July 6–10, 2022*, T. B. Sousa, Ed., ser. EuroPLoP'22, ACM, 2022.

[52] E. B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2nd International Conference on Software Engineering, San Francisco, California, USA, October 13–15, 1976*, R. T. Yeh and C. V. Ramamoorthy, Eds., IEEE Computer Society, 1976, pp. 492–497. [Online]. Available: http://dl.acm.org/citation.cfm?id=807723.

[53] S. Hönel, M. Ericsson, W. Löwe, and A. Wingkvist, "Using source code density to improve the accuracy of automatic commit classification into maintenance activities," *Journal of Systems and Software*, vol. 168, p. 110 673, 2020.

[54] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay, "Building theories in software engineering," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds., Springer, 2008, pp. 312–336.

[55] C. Wohlin and A. Rainer, "Is it a case study? – A critical analysis and guidance," *Journal of Systems and Software*, vol. 192, p. 111 395, 2022.

[56] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, 1st ed. Springer, 2012.

[57] S. Hönel and C. Wohlin, Personal communication, Prof. Wohlin recently authored guidelines for correctly classifying studies [55]., Dec. 2022.

[58] M. J. Tiedeman, "Post-mortems – Methodology and experiences," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 2, pp. 176–180, 1990.

[59] B. Collier, T. DeMarco, and P. Fearey, "A defined process for project post mortem review," *IEEE Software*, vol. 13, no. 4, pp. 65–72, 1996.

[60] J. Gerring, *Case Study Research: Principles and Practices* (Strategies for Social Inquiry), 2nd ed. Cambridge University Press, 2017.

[61] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, October 15–16, 2009, Lake Buena Vista, Florida, USA*, IEEE Computer Society, 2009, pp. 401–404.

[62] S. Hönel, P. Pícha, P. Brada, L. Rychtarova, and J. Danek, *Detection of the Fire Drill anti-pattern: 15 real-world projects with ground truth, issue-tracking data, source code density, models and code*, The repository for the source code based method is at: https://github.com/MrShoenel/anti-pattern-models, Jan. 2023.

[63] D. Chappell, *What is application lifecycle management?* Dec. 2008. [Online]. Available: https://web.archive.org/web/20141207012857/http://www.microsoft.com/global/applicationplatform/en/us/RenderingAssets/Whitepapers/What%20is%20Application%20Lifecycle%20Management.pdf (visited on 12/07/2014).

[64] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[65] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit.," *Psychological bulletin*, vol. 70, no. 4, p. 213, 1968.

[66] K. L. Gwet, "Computing inter-rater reliability and its variance in the presence of high agreement," *British Journal of Mathematical and Statistical Psychology*, vol. 61, no. 1, pp. 29–48, 2008.

PREVIEW

[67] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, vol. 33, no. 2, pp. 363–374, Jun. 1977, PMID:884196.

[68] D. Klein, "Implementing a general framework for assessing interrater agreement in stata," *The Stata Journal*, vol. 18, no. 4, pp. 871–901, 2018.

[69] B. W. Boehm, *Software Engineering Economics*, 1st ed. Philadelphia, PA: Prentice Hall, Oct. 1981.

[70] N. C. Dalkey, "The Delphi Method: An experimental study of group opinion," The RAND Corporation, Santa Monica, CA, Tech. Rep., 1969, Document Number: RM-5888-PR. [Online]. Available: https://www.rand.org/pubs/research_memoranda/RM5888.html.

[71] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956, zbMATH:0073.14602, MathSciNet:MR79873.

[72] D. M. Endres and J. E. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1858–1860, 2003. [Online]. Available: https://doi.org/10.1109/TIT.2003.813506.

[73] B. Hofner, L. Boccuto, and M. Göker, "Controlling false discoveries in high-dimensional situations: Boosting with stability selection," *BMC Bioinformatics*, vol. 16, no. 1, p. 144, May 2015.

[74] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, Fourth. New York: Springer, 2002. [Online]. Available: http://www.stats.ox.ac.uk/pub/MASS4.

[75] F. Bertrand and M. Maumy-Bertrand, *plsRglm: Partial least squares linear and generalized linear regression for processing incomplete datasets by cross-validation and bootstrap techniques with R*, arXiv, 2018.

[76] A. Peters and T. Hothorn, *Ipred: Improved predictors*, R package version 0.9-9, 2019. [Online]. Available: https://CRAN.R-project.org/package=ipred.

[77] F. Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994. [Online]. Available: http://www.jstor.org/stable/2684253.

[78] S. Hönel, "Technical reports compilation: Detecting the Fire Drill anti-pattern using source code and issue-tracking data," *CoRR*, vol. abs/2104.15090, Jan. 2023. arXiv: 2104.15090 [cs.SE].

[79] D. Vysochanskij and Y. I. Petunin, "Justification of the 3σ rule for unimodal distributions," *Theory of Probability and Mathematical Statistics*, vol. 21, no. 25-36, 1980.

[80] P. Tchébychef, "Des Valeurs Moyennes," *Journal de Mathématiques Pures et Appliquées*, 2nd ser., vol. 12, pp. 177–184, 1867, Traduction du Russe par M. N. de Khanikof. [Online]. Available: http://eudml.org/doc/234989.

[81] G. C. Cawley and N. L. C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010.

[82] S. Raudys and A. K. Jain, "Small sample size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252–264, 1991.

[83] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinformatics*, vol. 7, no. 1, Feb. 2006.

PREVIEW

[84] A. Vabalas, E. Gowen, E. Poliakoff, and A. J. Casson, "Machine learning algorithm validation with a limited sample size," *PLOS ONE*, vol. 14, no. 11, E. Hernandez-Lemus, Ed., pp. 1–20, Nov. 2019.

[85] T. Shaikhina, D. Lowe, S. Daga, D. Briggs, R. Higgins, and N. Khovanova, "Machine learning for predictive modelling based on small data in biomedical engineering," *IFAC-PapersOnLine*, vol. 48, no. 20, pp. 469–474, 2015.

[86] L. Torgo, R. P. Ribeiro, B. Pfahringer, and P. Branco, "SMOTE for regression," in *Progress in Artificial Intelligence – 16th Portuguese Conference on Artificial Intelligence, EPIA 2013, Angra do Heroísmo, Azores, Portugal, September 9–12, 2013. Proceedings*, L. Correia, L. P. Reis, and J. Cascalho, Eds., ser. Lecture Notes in Computer Science, vol. 8154, Springer, 2013, pp. 378–389.

[87] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2020. [Online]. Available: https://www.R-project.org/.

[88] B. Greenwell, B. Boehmke, J. Cunningham, and G. Developers, *Gbm: Generalized boosted regression models*, R package version 2.1.8, 2020. [Online]. Available: https://CRAN.R-project.org/package=gbm.

[89] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: https://CRAN.R-project.org/doc/Rnews/.

[90] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, "Kernlab – an S4 package for kernel methods in R," *Journal of Statistical Software*, vol. 11, no. 9, pp. 1–20, 2004. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v011i09.

[91] P. A. Lachenbruch and M. R. Mickey, "Estimation of error rates in discriminant analysis," *Technometrics*, vol. 10, no. 1, pp. 1–11, 1968. [Online]. Available: http://www.jstor.org/stable/1266219.

[92] M. J. Kearns and D. Ron, "Algorithmic stability and sanity-check bounds for leave-one-out cross-validation," *Neural Computation*, vol. 11, no. 6, pp. 1427–1453, 1999.

[93] J. L. Fleiss, J. Cohen, and B. S. Everitt, "Large sample standard errors of kappa and weighted kappa.," *Psychological Bulletin*, vol. 72, no. 5, pp. 323–327, Nov. 1969.

[94] D. V. Cicchetti and S. A. Sparrow, "Developing criteria for establishing interrater reliability of specific items: applications to assessment of adaptive behavior," en, *American Journal of Mental Deficiency*, vol. 86, no. 2, pp. 127–137, Sep. 1981, PMID:7315877.

[95] J. L. Fleiss, *Statistical Methods for Rates and Proportions* (Probability & Mathematical Statistics S.), en, 2nd ed. Nashville, TN: John Wiley & Sons, May 1981.

[96] D. A. Regier *et al.*, "DSM-5 field trials in the United States and Canada, part II: Test-retest reliability of selected categorical diagnoses," en, *American Journal of Psychiatry*, vol. 170, no. 1, pp. 59–70, Jan. 2013.

[97] A. S. Lee, "A scientific methodology for MIS case studies," *MIS Quarterly*, vol. 13, no. 1, pp. 33–50, 1989.

PREVIEW

Appendices

The following appendices provide additional information about the Fire Drill phenomenon in full (A), the projects' setup (B), the Fire Drill's observed symptoms & consequences, as well as supercategories (C), observations counter-indicative of a Fire Drill (D), and, lastly, a more detailed and numeric view of the quantitative analysis' variable importance (E).

Appendix A. Full Fire Drill description

Here, we include the most recent and complete description of the Fire Drill anti-pattern using a pattern language and a typically structured template [3, 12]. This study operates on this description and any won insights, new results, forces, symptoms & consequences, etc., were *not* incorporated into this original description. The following is an exact copy of the original resource by Picha *et al.* [51][11]. The elements *Also Known As*, *Variations (optional)*, *Example(s) (optional)*, and *Notes (optional)* were left out as they are currently empty and reserved for future use.

Fire Drill

Summary

Requirements and Analysis phases prolonged and consuming disproportionate amount of resources (because management want to do them "right"), then frantic "everything needs to be done yesterday" period to finish on time (when management finds out they wasted most of project's schedule and resources on analysis).

Context

Waterfall(ish) projects, especially when project oversight is loose and/or management is not driven by outcome.

Unbalanced forces

- need (desire) to have specifications perfect
- management consumed by internal (political) issues
- actual development of a high-quality product takes time
- quality objectives formally stated and high
- strict deadlines for delivery

Symptoms and consequences

- long period at project start where activities connected to requirements, analysis and planning prevail, and design and implementation activities are rare
- only analytical or documentational artefacts for a long time
- relatively short period towards project end with sudden increase in development efforts (i.e. rock-edge burndown, especially when viewing implementation tasks only)
- little testing/QA and project progress tracking activities during development period

[11] Full Fire Drill Description. 2022. https://github.com/ReliSA/Software-process-antipatterns-catalogue/blob/master/catalogue/Fire_Drill.md.

PREVIEW

– final product with poor code quality, many open bug reports, poor or patchy documentation
– if the previous trhee points do not apply, project schedule or scope is compromised (i.e., either delayed delivery or descoping occurs)
– stark contrast between interlevel communication in project hierarchy (management - developers) during the first period (close to silence) and after realizating the problem (panic and loud noise)

Causes

– management does not take seriously development effort (time) estimates
– management absorbed in "various technopolitical issues (…) prevent[ing] the development staff from making progress"
– team is happy to produce artifacts early in the project
– requirements are complex and their prioritization is not forced early on
– team overseeing the need to prioritize "working code over comprehensive documentation"
– management wants to appear the project to be on track
– management believes it is more important to deliver complete functionality than good quality
– project tracking and oversight is loose, easily lulled inco complacency by easy-to-reach outcomes

(Refactored) solution

– force the team to start delivering (parts of) the "consumable solution" early, possibly alongside the analysis and planning artefacts, by instituting strong project tracking and oversight related to actual outcomes
– it helps to follow an iterative process, architecture-driven development, and have a well-performing product owner

Anti-pattern	Relation
Analysis Paralysis	potential cause
Collective Procrastination [18]	more generic case

Sources

[12], [SOU'18] Fire Drill, [3]

Appendix B. Project setup

Table B1 gives an overview about each project's setup, including number of team members, project duration, number of iterations, man-hours logged on issues, and the number of issues or commits per activity (see Section 4.5). Each project developed a different kind of application. The following list gives a brief description for the developed application in each.

Pr. 1: Desktop and web application for full-text searches in scanned documents
Pr. 2: Heatmap web application over open data

PREVIEW

Pr. 3: Enhancement of web application for complex graph visualization
Pr. 4: Web application for HR management
Pr. 5: A Universal deserializer in and for Java
Pr. 6: Upgrade of a web application for linguistic research
Pr. 7: Mobile application for museum visitors
Pr. 8: Bitmap generator for public transportation
Pr. 9: Web application simulating pivot tables
Pr. 10: Client web front-end for sensor data
Pr. 11: Application for certificate management with web and desktop interfaces
Pr. 12: Web application over two databases with linguistic research data
Pr. 13: Web and mobile application for open weather data visualization and prediction
Pr. 14: Virtual Reality application arm rehabilitation machine control
Pr. 15: Sensor dashboard for the Raspberry Pi platform

Lastly, the projects were free to choose a programming language that best suited their needs and conformed with the customer's requirements. The following list shows the share each language had, obtained from the lines of code[12].

Pr. 1: 43.64 % Java, 37.90 % TypeScript, 9.70 % HTML, and 8.76 % Others
Pr. 2: 30.90 % Python, 18.28 % CSS, 18.23 % PHP, 11.41 % SCSS, 8.50 % JavaScript, 8.48 % Twig, and 4.19 % Others
Pr. 3: 57.89 % JavaScript, 28.16 % Java, 8.47 % CSS, and 5.48 % Others
Pr. 4: 77.41 % Java, 17.29 % JavaScript, and 5.30 % Others
Pr. 5: 98.95 % JavaScript, and 1.05 % Java and Others
Pr. 6: 88.34 % PHP, 3.29 % PLpgSQL, and 8.37 % HTML, JavaScript and Others
Pr. 7: 63.71 % PHP, 35.48 % Blade, and 0.81 % Others
Pr. 8: 100.00 % Java
Pr. 9: 78.84 % Java, 10.92 % HTML, 7.84 % HTML, and 2.39 % CSS
Pr. 10: 62.87 % TypeScript, 24.07 % SCSS, 12.03 % HTML, and 1.03 % Others
Pr. 11: 80.14 % Python, 10.57 % HTML, 7.32 % JavaScript, and 1.97 % Others
Pr. 12: 42.52 % JavaScript, 27.30 % HTML, 16.58 % PHP, 11.17 % Java, and 2.43 % CSS and Others

Table B1. General characteristics of the student projects

Project	Team members	Project duration (d)	Iterations	Time spent (h)	Issues	Time logged (h) REQ/DEV/DESC	Commits A/C/P
1	4	78	6	277.95	104	62/158/0	36/32/48
2	4	93	6	399.35	98	115/177.9/4	42/108/76
3	4	97	7	303.80	113	59.3/136.7/24.5	26/35/50
4	4	100	6	346.75	118	67.5/221.8/66	29/59/38
5	2	61	4	238.25	44	60/131.3/14	33/26/51
6	4	98	5	234.62	101	39.4/157.9/2	79/63/75
7	5	100	6	396.35	114	80.8/183.1/26.5	83/64/36
8	4	94	7	206.00	85	46.5/89.5/0	10/6/14
9	4	84	6	285.35	112	37/176.3/51.5	54/23/23
10	4	91	6	348.40	128	59.7/152.8/9.5	74/38/46
11	4	83	6	300.00	114	45.8/147.6/15	151/168/223
12	4	94	7	292.25	81	35/172.6/0	35/61/40
13	4	105	6	409.25	181	49.3/210.6/122.4	67/48/77
14	3	95	6	204.75	72	14.3/104.8/7.4	34/8/20
15	4	98	6	246.30	62	18.8/86.5/0.8	24/27/15

[12]GitHub Linguist was used to compute the shares, see https://github.com/github/linguist (2023).

PREVIEW

Pr. 13: 72.51 % C#, 22.92 % ShaderLab, 3.76 % HLSL, and 0.82 % HTML
Pr. 14: 76.31 % C++, 14.44 % C#, 6.37 % Python, 1.79 % QMake, and 1.09 % Others
Pr. 15: 100.00 % Python

Appendix C. Fire Drill symptoms and consequences

This appendix shows the results of the systematic analysis of the raters' notes for each project, according to the methodology described in Section 4.4. The raters' notes, as well as all other data is to be found in [62]. This list is an excerpt from the most recent Fire Drill description[13]. It is organized into a top-level list of symptoms and consequences (`SC1`–`SC7` and `ESC1`–`ESC3`) and a nested list for each with concrete observed, empirical instances (denoted as `Exx`). Each empirical instance has a severity attached. It follows the format `[project, rater(s), severity]`, e.g., `[1,AB,3]` indicates that raters A and B commonly identify a problem instance in project one, and the severity is three out of five. As of the second pass, some observations and their severity were aggregated. For example, (`[13,A,3]`, `[13,B,2]`, `[13,C,4]`) ⇒`[13,ABC,5]` means that all three raters observed a symptom/consequence in project 13, with varying severity each. The aggregation reduces this to a single observation with higher severity, according to the ordinal scale's description.

- `SC1`. long period at project start where activities connected to requirements, analysis and planning prevail, and design and implementation activities are rare
  - `E01`. can be caused by, e.g., the team needing some time for familiarization with the (new, changed) tools, way of communication, or process in the beginning `[1,A,1]`, `[3,B,2]`, `[5,A,1]`, `[7,B,1]`, `[9,B,2]`, `[13,A,1]`, `[14,A,1]`, `[15,A,2]`
  - `E02`. work delayed due to external factors, such as ramping-down of previous project or other, non-related work `[1,A,0]`, `[15,A,0]`
  - `E03`. project is under- or over-scoped from the beginning, so the team spends time idling or does not know where to start `[3,C,1]`, (`[13,A,3]`, `[13,B,2]`, `[13,C,4]`) ⇒`[13,ABC,5]`, `[15,C,1]`
- `SC2`. only analytical or documentational artifacts for a long time
  - `E04`. Development takes place only at the end of an iteration, sometimes rushed (opposite of `CISC08`) `[3,C,1]`, `[6,A,1]`, `[14,A,3]`
- `SC3`. relatively short period towards project end with a sudden increase in development efforts (i.e. rock-edge burndown, especially when viewing implementation tasks only)
  - `E05`. team rushes to deliver at least an MVP to meet the final deadline `[13,A,3]`
- `SC4`. little testing/QA and project progress tracking activities during the development period
  - `E06`. sometimes caused by improper usage of project management tools, for example logging time only at the end of a phase `[3,C,2]`
  - `E07`. too much focus on "visible" progress by managerial decision while testing/QA is neglected, which leads to an accumulation of technical debt in the long-term (Half Done Is Enough) `[12,A,1]`
- `SC5`. final product with poor code quality, many open bug reports, poor or patchy documentation

[13]The Fire Drill description in the process anti-pattern catalog. 2023. https://github.com/Mr-Shoenel/Software-process-antipatterns-catalogue/blob/7a4d8/catalogue/Fire_Drill.md.

PREVIEW

  - E08. too meet the final delivery date, the product quality is decreased by skipping, e.g., features or proper Q/A (alternatively, the may be delivered late, but as agreed) ([4,B,1], [4,C,2]) ⇒[4,BC,3], [6,C,2], [7,B,2], ([13,A,4], [13,C,2]) ⇒[13,AC,5]
- SC6. if points SC3 through SC5 do not apply, (likely) the project schedule or scope is compromised (i.e., either delayed delivery or descoping occurs)
  - E09. team accepts change requests, re-prioritization of existing or new issues within a phase (e.g., after the start of a sprint); improper change management process [1,A,1], [3,C,4], ([5,A,1], [5,B,1]) ⇒[5,AB,2], [9,A,1], [10,B,2], [12,A,3], ([13,A,4], [13,B,4]) ⇒[13,AB,5]
  - E10. planned work is not completed and overflows into the next phase (e.g., sprint), due to, e.g., an over-challenged team (opposite of CISC28), misestimation, or unequal work distribution [3,C,4], [7,C,3], [9,C,3], [10,C,1], [14,A,1], [15,B,1]
  - E11. iterations are too short and are artificially prolonged, forcing the team to do overtime or to truncate the workload [3,C,4], [4,C,2], [7,C,1]
- SC7. stark contrast between interlevel communication in project hierarchy (management - developers) during the first period (close to silence) and after realizing the problem (panic and loud noise)

Here is a list of new, empirical symptoms and causes:

- ESC1. poor communication (e.g., unresponsive, relayed, large overhead, or underqualified decision-maker) between stakeholders (e.g., customer) and the development team
  - E12. unresponsive customer or unsatisfactory (e.g., late, incomplete, or slow) communication (critical infos or materials not provided timely) ([3,A,4], [3,B,4]) ⇒[3,AB,5], ([4,A,3], [4,B,4], [4,C,3]) ⇒[4,ABC,5], [5,C,1], ([6,A,3], [6,B,3]) ⇒[6,AB,4], ([7,A,3], [7,C,3]) ⇒[7,AC,4], ([10,A,3], [10,B,3], [10,C,2]) ⇒[10,ABC,5], [11,A,1], [12,A,1], [13,C,1], [14,A,1], ([15,A,1], [15,C,1]) ⇒[15,AC,2]
  - E13. requirements cannot be clearly negotiated or are ambiguous [3,C,3], ([10,A,3], [10,B,3]) ⇒[10,AB,4], ([13,A,3], [13,C,4]) ⇒[13,AC,5]
  - E14. post-negotiation misunderstandings (without proper re-negotiation) [3,A,2]
  - E15. tacit misunderstanding (stakeholder and team believe they are on the same page, but they are not in actuality) [3,A,3], [4,C,1]
  - E16. customer interferes with project management without properly communicating the made changes, which directly translates into a project risk [9,B,2]
- ESC2. high project risk (opposite of CISC26), often manifests itself through, e.g., unrealistic work item estimates, the absence of proper testing (opposite of CISC09), or improper documentation
  - E17. business requirements (tacitly/unknowingly) misinterpreted ([3,C,3], [3,A,3]) ⇒[3,AC,4], [4,C,3], [6,A,2], [12,A,1]
  - E18. finalized work does not conform to the defined specification/expectation [3,C,2]
  - E19. new functionality introduces bugs, and not enough slack was allocated during planning for the fixing (or preventing by proper testing) of these [3,C,1], [4,A,2]
  - E20. tasks are done in the wrong order (Cart Before Horse), esp. development before properly analyzing and planning ([6,A,2], [6,B,2]) ⇒[6,AB,3], [12,A,1], [15,A,1]
  - E21. imbalanced activities at the beginning, end, or during the project, such as too much focus on development early and requirements analysis later that leads

PREVIEW

to descoping, for example ([6,B,2], [6,C,2]) ⇒[6,BC,3], ([7,A,1], [7,B,1]) ⇒[7,AB,2], [9,A,1], [12,C,1], ([13,A,3], [13,B,2]) ⇒[13,AB,4]
  - E22. lack of experience that leads to misestimation of work items [6,C,1], [9,A,1], [13,C,2]
  - E23. work items or goals not properly defined, absent, or defined too late [7,C,4], [13,A,4]
  - E24. management fails to ascertain that the development team is available to its planned capacity (e.g., it allows the team to be affected by external factors), which has a negative impact on the progression of the project, such as descoping, quality regression, or delayed delivery [9,C,1], [11,A,1], [13,A,1]
  - E25. Strong dependency between stakeholders and the development team, such that the team cannot proceed very long or at all by themselves (opposite of CISC15) [10,C,1]
  - E26. technical difficulties in the environment, such as the infrastructure, which cause unexpected delays to, e.g., the development or deployment ([12,A,3], [12,B,1]) ⇒[12,AB,3], ([15,A,1], [15,C,1]) ⇒[15,AC,2]
  - E27. frequent project schedule adaptions, manifested by excessive use of administrative tools, leading to a low-quality product [13,B,4], [15,A,2]
- ESC3. poor usage of project management tools and methodologies which gives rise to management misinterpreting the progress and state of the project
  - E28. too-large goals that were not properly broken down into smaller issues [3,C,1]
  - E29. mislabeling of items; for example, marking an Epic as a Task [4,C,1]
  - E30. a discrepancy between the defined work and the actual work exists, due to, e.g., time not logged properly, issues not defined, or work completed during undocumented overtime [3,C,1], [7,C,4], [10,A,1], [15,A,1]
  - E31. information mismanagement, for example, by duplication or using too many different systems for storing and disseminating information [4,B,1]

Appendix D. Symptoms and consequences indicating the absence

During the analysis of the raters' notes, recurring elements of healthy projects, showing no or miniscule signs of a Fire Drill, emerged. While the absence of evidence does not mean that a Fire Drill is not present, we gathered some empirical evidence for symptoms and consequences that would be *counter-indicative* of the phenomenon. The following unordered list is another excerpt from the most recent description. It is numbered similarly to the list of symptoms and consequences (CISC01–CISC29), but does not include the number of observations or how strong an observation manifested. We use the abbreviation CISC to mean counter-indicative symptom & consequence.

- CISC01. communication and collaboration with the customer is seamless
- CISC02. no descoping which typically happens towards the end of a phase (sprint, milestone, etc.)
- CISC03. timely product delivery according to agreed-upon quality
- CISC04. satisfaction among all stakeholders (e.g., team, customer, etc.)
- CISC05. regular and successful iteration evaluations that do not result in the unveiling of (large/additional) problems
- CISC06. clear understanding of the requirements and resulting unproblematic execution

PREVIEW

- CISC07. equal (or almost equal) work distribution among team members (also: fair work distribution among differently-skilled/-tasked team members)
- CISC08. linear burn-down (i.e., done work is distributed uniformly, instead of, e.g., at the end of a phase)
- CISC09. product tested properly (e.g., appropriate tests and/or good coverage), as well regularly/continuously
- CISC10. starting to implement features right from the project inception (clear requirements)
- CISC11. proper allocation of project resources (esp. time)
- CISC12. proper (planning of) distribution of time (spent) across the required activities (e.g., enough time spent on defining requirements properly)
- CISC13. appropriate prioritization of activities when resources (often time) become (temporarily) scarce
- CISC14. successful intermediate and final product deliveries (according to customer's acceptance criteria)
- CISC15. team can proceed at least short-term even if the customer is unavailable (good internal crisis management)
- CISC16. accurate work-item estimates (time, points, etc.), esp. no over-estimation (which indicates high level of uncertainty and, therefore, risk)
- CISC17. project management tool(s) used accordingly; e.g., proper usage of primitives (item types), the Scrum/Kanban board (or swimlanes), regular updates (all these indicate proper management)
- CISC18. regular activities according to used methodology (e.g., Scrum), such as daily meetings, retrospectives, and milestones
- CISC19. change requests, re-prioritization of existing or new issues are rejected by the team once the phase (e.g., sprint) started in which they were planned for (as should be)
- CISC20. proper communication among team members; direct messaging, as well as dedicated channels and often the usage of bots (from, e.g., a CI pipeline)
- CISC21. efficient communication with customer, that is, direct (no relays), quick, unproblematic, of high quality, tending to the necessary aspects of the product (low overhead)
- CISC22. stable team (no developer churn) and harmony among members
- CISC23. mutual understanding: effort estimations between customer and team are similar (customer understands technical challenges and team understands business requirements)
- CISC24. activities in right order (e.g., analysis before design before implementation etc.)
- CISC25. progress is reflected empirically (objectively), i.e., provably no discrepancy between reported and actual progress exists
- CISC26. proper risk management through, e.g., the development of prototypes
- CISC27. the scope may change and adapt over the course of the project (due to the agile nature), but it does not increase/widen without additional resources
- CISC28. team is not undersized for the project: no (steady) accumulation of non-finished work items into the next phase
- CISC29. when external forces and (un)forseeable events happen, development is suspended and/or the product is delayed accordingly, allowing the team to catch up (rather than forcing them to do overtime)

PREVIEW

Appendix E. Detailed variable importance

Table E1 and Table E2 show the detailed variable importance for source code and issue-tracking, respectively. Both tables reflect the same result as shown in Figure 7.

Table E1. Detailed variable importance (scaled to percent) for the no-pattern source code dataset, including means and sums across segments and features

Variable (feature)	Seg1	Seg2	Seg3	Seg4	Seg5	Seg6	Seg7	Seg8	Seg9	Seg10	sum
A	1.58	0.69	1.44	2.70	1.47	1.67	0.73	0.81	1.44	1.68	14.20
CP	1.53	1.32	3.39	2.19	1.33	2.31	1.38	1.89	1.10	0.72	17.16
$\mathsf{FREQ}_{\mathsf{nm}}$	2.60	1.00	2.35	1.69	0.79	2.62	1.23	1.17	1.59	1.08	16.12
A |0 CP	1.27	1.28	2.87	2.95	2.08	0.59	2.01	3.65	1.13	1.18	19.00
A |0 $\mathsf{FREQ}_{\mathsf{nm}}$	0.64	0.74	2.12	2.27	2.53	2.47	1.61	2.91	1.49	0.77	17.55
CP |0 $\mathsf{FREQ}_{\mathsf{nm}}$	2.65	1.31	3.16	3.02	0.91	1.28	0.98	1.41	0.45	0.80	15.97
mean	1.71	1.06	2.55	2.47	1.52	1.82	1.32	1.97	1.20	1.04	n/a
sum	10.28	6.35	15.33	14.81	9.10	10.94	7.93	11.84	7.21	6.23	200.00

Table E2. Detailed variable importance (scaled to percent) for the no-pattern issue-tracking dataset, including means and sums across segments and features.

Variable (Feature)	Seg1	Seg2	Seg3	Seg4	Seg5	Seg6	Seg7	Seg8	Seg9	Seg10	sum
REQ	1.19	1.20	1.50	2.42	1.63	1.25	0.89	0.64	1.15	1.16	13.02
DEV	1.20	2.93	1.70	1.02	1.00	1.55	0.62	1.74	2.48	2.39	16.63
DESC	1.39	0.72	1.24	1.26	0.78	1.50	0.60	2.84	2.47	1.39	14.19
REQ |0 DEV	1.76	1.21	3.14	1.07	1.99	1.68	1.06	2.15	2.91	1.58	18.55
REQ |0 DESC	3.86	2.99	2.94	1.31	1.42	2.07	1.17	1.75	2.23	0.99	20.72
DEV |0 DESC	1.25	1.00	1.04	1.30	2.88	2.68	0.83	2.67	2.23	1.00	16.89
mean	1.78	1.68	1.93	1.40	1.62	1.79	0.86	1.97	2.24	1.42	n/a
sum	10.65	10.05	11.56	8.38	9.69	10.74	5.17	11.79	13.47	8.50	200.00

PREVIEW