Success and Failure in Software Engineering: A Followup Systematic Literature Review

Damian A. Tamburri, *Member at Large, IEEE,* Fabio Palomba, *Member IEEE,* Rick Kazman, *Senior Member, IEEE,*

Abstract

Success and failure in software engineering are still among the least understood phenomena in the discipline. In a recent special journal issue on the topic, Mäntylä et al. started discussing these topics from different angles; the authors focused their contributions on offering a general overview of both topics without deeper detail. Recognising the importance and impact of the topic, we have executed a followup, more in-depth systematic literature review with additional analyses beyond what was previously provided. These new analyses offer: (a) a grounded-theory of success and failure factors, harvesting over 500+ factors from the literature; (b) 14 manually-validated clusters of factors that provide relevant areas for success- and failure-specific measurement and risk-analysis; (c) a quality model composed of previously unmeasured organizational structure quantities which are germane to software product, process, and community quality. We show that the topics of success and failure deserve further study as well as further automated tool support, e.g., monitoring tools and metrics able to track the factors and patterns emerging from our study. This paper provides managers with risks as well as a more fine-grained analysis of the parameters that can be appraised to anticipate the risks.

Index Terms

Success and failure; Software engineering; Systematic Literature Reviews.

1 Introduction

In the scope of software production and operation, the notions of success and failure are intriguing, having different forms and manifesting under varied conditions [1], [2]. In a recent special issue of *Empirical Software Engineering* on this topic [3], the editors remarked that, *"despite ongoing concerns over the failure rate of software projects, basic questions such as "How do we measure general software success?" and "How can software failure rates be measurably reduced?" remain still only partially explored"*. The editors concluded that addressing these questions is critical to further understand and steer software projects towards success. We pick up the challenge from where it was left off [3]. In this paper we refine and re-execute the research design set up by the editors in their special issue introduction, aimed at identifying and analyzing a set of papers focused on the topics of success and failure in software engineering research and practice.

The goal we address is to add further analyses on top of what Mäntylä et al. [3] offer as a preliminary analysis. Our objectives with these additional analyses are threefold. First, we aim to elicit a *grounded theory* of success and failure factors so that other researchers may identify such factors and how to measure them, ideally creating a general software success (or failure)

- *D. A. Tamburri is with the Eindhoven University of Technology and the Jheronimus academy of Data Science E-mail: d.a.tamburri@tue.nl*
- *F. Palomba is with University of Salerno E-mail: fpalomba@unisa.it*
- *R. Kazman is with University of Hawaii & SEI/CMU E-mail: kazman@hawaii.edu*

prediction model. Second, we aim to highlight the most relevant themes of factors thus identifying the areas of software engineering research and practice that are under-supported by measures. Third, we aim to elicit a rigorous quality model for these under-supported quantities.

Briefly, our results show that success and failure in software projects is mediated by over 500 factors (e.g., presence of users directly in the software process [4]) arranged in 40+ core-concepts (e.g., effort estimation). Furthermore, there exist 14 themes along which success and failure is determined (in practice) and studied (in research), such as *best practices evaluation and monitoring*, or *software measurement* or *organizational structure and motivation*. Finally, 5 out of the 14 themes reflect organizational structure quality which, to date, does not have a rigorous model (that is, a set of measurable quantities [5]). As the final contribution of this article, we offer a first attempt at such a quality model that captures the most recurrent measurable factors and quantities from the aforementioned 5 themes, such as *truck-factor* [6] or *socio-technical congruence* [7].

Replication Package. Finally, to encourage replication we make available a comprehensive package containing all papers, Grounded-Theory sources as well as analysis of data performed in this study [1].

Structure of the paper. Section 2 outlines the terminology used in the paper. In Section 3, we describe our research methods, while Section 4 overviews the results achieved. Section 5 provides discussions on the key findings of the paper. Finally, Section 6 concludes the paper and outlines our future research agenda on the topic.

2 Scope and Terminology

The scope intended for this work draws primarily from the single preliminary study reported in Mäntylä et al. [3] which encompasses a very large sample of research and discusses the concepts of success and failure or the context in which such phenomena manifest themselves from a very high-level. The scope we set out to investigate as a spin-off of the aforementioned previous work encompasses the high-relevance and high-impact research currently available in literature that elaborates either on (1) the primary studies emerged in Mäntylä et al. [3] or (2) on any of the concepts or conclusions emerged in the same paper. With respect to point 1 above, We are aware that these phenomena are complex and cannot be simplistically reduced to mere factors and dimensions. Our goal is to build upon the work by Mäntylä et al. and consequently collect a grounded-theory which acts as a foundation of what is known about these phenomena such that further work can be developed based on this foundation. With respect to point 2 above, our paper is a followup study to Mäntylä et al. and, for this reason, we inherit much of the terminology previously used in the target study [3]. In the following, we report those terms and their associated meaning:

Context. This reflects successful or failed software engineering projects and their study from any empirical, experimental, or theoretical perspective.

Success. Success represents the long-lasting conditions wherefore a software project is maintained in a state meeting its expectations.

Failure. A failure is the moment in time where a software project no longer meets its expectations.

In the next section, we describe the research methodology we employed to conduct our followup systematic literature review.

1. See here: https://figshare.com/s/e6f0968e55c2cd024389

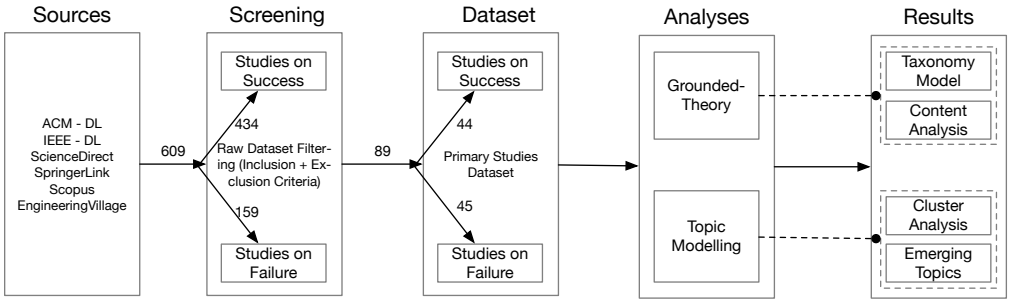

Fig. 1. An overview of our research design, from sources to results.

3 Research Approach

3.1 Research Design

The *goal* of this work is to obtain an in-depth overview on the phenomena of software success and failure. The *purpose* is to provide the research community with actionable insights on the factors impacting a software project to be successful, so that future studies could be devised to explicitly target novel methodologies and methods to take those factors under control. Our *perspective* is of both researchers and practitioners, who are interested in gathering deeper knowledge of the attributes to be monitored to mitigate the risk of software failure.

To this end we aimed at providing further analyses on top of the literature retrieved to provide a greater depth of understanding. The analyses we added aim at answering the following *research questions* (**RQ**s):

RQ$_1$. *What factors are reportedly connected to success or failure?*
RQ$_2$. *What themes emerge across such factors?*
RQ$_3$. *What themes are currently unobserved and what previously-existing metrics can support these unobserved themes?*

Note that, in the scope of **RQ**$_3$, by *unobserved* we mean the factors and themes emerging from **RQ**$_1$ and **RQ**$_2$ that currently have no accepted metrics to support their appraisal. The ultimate goal of this research question is to provide practitioners with a *quality model*, that is, an aggregate of measures which were previously defined, evaluated, and automated for these unobserved factors and themes. Fig. 1 recaps the main steps undertaken to attain results as well as the inputs and outputs of each phase using a simple box and line diagram. The main boxes in the figure represent steps that were undertaken while smaller boxes represent results of those steps linked by action arrows; for the early dataset and sample selection stages, arrows are augmented with quantities connected to the sampling process. Finally, dotted lines connect each analysis (quantitative or qualitative) with its analysis results.

3.2 Literature Retrieval Approach

To retrieve the target literature we executed an augmented retrieval strategy described in previous work [3]. The new strategy focuses on eliciting papers that focus on industrial applicability of the proposed claims, results, and contributions or which offer results stemming from industrial practice and experience. Specifically, to retrieve papers we execute the following search string:

TITLE-ABS ((“software engineering” OR “software development” OR “software project” OR “it project” OR “it development” OR “it engineering”) AND TITLE (“success” OR “failure”) AND BODY (“case-study” OR “industrial-*” OR “practiction*”))

where TITLE-ABS indicates that the subsequent search terms are considered only in the scope of title and abstract of the papers. TITLE indicates that the search is conducted only on papers titles whereas BODY indicates that the search is conducted only on the body of the articles. This is the exact search string defined by Mäntylä et al.

The search process has been conducted on a number of different databases, namely:

- IEEE Xplore digital library (http://ieeexplore.ieee.org);
- ACM digital library (https://dl.acm.org);
- ScienceDirect (http://www.sciencedirect.com);
- SpringerLink (https://link.springer.com);
- Scopus (https://www.scopus.com);
- Engineering Village (https://www.engineeringvillage.com);

The selection of these databases was driven by our willingness to gather as many papers as possible to properly conduct our systematic literature review. In this respect, the selected sources are recognized as the most representative and complete for Software Engineering research and are used in many other SLRs [8] because they contain a massive amount of literature—journal articles, conference proceedings, books etc.—related to our research questions. As described by Mäntylä et al. [3], no paper on success and failure in software engineering was published before 1970: thus, our search targeted papers published between January 1970 and August 2019.

With the above procedure we elicited an initial set of 609 papers, of which 159 were on software project failures and 434 on software project successes with the others describing case-studies or direct practitioner experience without any specific success or failure discussion.

Then, we executed the same manual filtering process of Mäntylä et al. [3] to remove non-relevant sources. Specifically, we filtered out:

- papers that were not written in English;
- papers whose full text was not available;
- short papers (up to 4 pages) just reporting preliminary results;
- papers from workshops;
- papers that adopted the term "failure" to indicate software faults;
- papers that described a method or tool that theoretically would reduce the risk of project failure or increase the likelihood of success, but that did not actually assess it;
- papers that described the failure or success of introducing new tools or processes, but that did not relate this to project success or failure;
- papers referring to just one development phase rather than the entire software lifecycle;
- papers that were about project success and failure, but that did not provide research results;
- duplicate papers; specifically we excluded conference papers in case an extended journal article version was available.

The manual filtering was conducted by two of the authors of this paper, who jointly scanned each candidate paper and judged its suitability for the study. This initial process took two weeks and led to the final selection of 89 primary studies, almost evenly balanced between success and failure. These papers all come from well-established and high-ranking[2] conferences and journals sponsored by ACM, IEEE, Springer, Elsevier, and Wiley.

3.3 Analysis and Synthesis Methods

3.3.1 Qualitative Analysis

Analysis and synthesis of results were carried out through the well-known Grounded-Theory (GT) approach [9].

2. either class A* or A or B from the CORE Edu Rankings portal http://portal.core.edu.au/

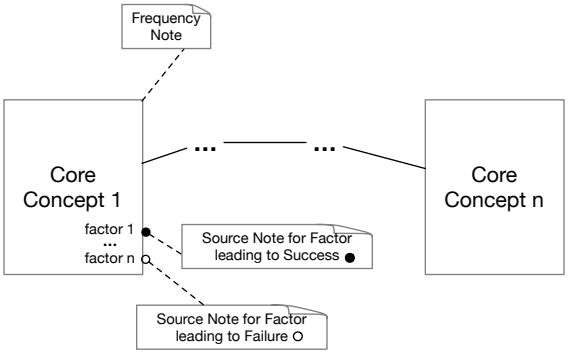

Fig. 2. Grounded Theory as Employed in This Study.

Fig. 2 outlines how we realized and represented the theory. Specifically, core concepts are represented as boxes with factors as attributes; relations reflect either memos or explicit relations found between factors. Every cluster is mapped to a note reporting its frequency and relative weight (measured in terms of reported code occurrences, and in how many papers those codes were reported) while every factor is mapped to a reference literature element with indication of whether the factor was leading to success (filled circle) or failure (unfilled circle). For example, the usage of the buddy-pairing best practice as part of Cisco systems' strategies to address global software engineering from one of our success-story reports is tagged with the "best practice" code, as well as the "global-software engineering process" code.

3.3.2 Quantitative Analysis

Following an approach similar to that proposed by Mäntylä et al. [3], we used the well-established topic modelling technique known as Latent Dirichlet Allocation (LDA). However, rather than applying the technique to our papers as done previously, we applied the technique to cluster the factors emerging from our grounded theory activities, along with their textual definitions. Clustering of such factors allowed us to elicit a detailed view of the factors themselves, thus enabling the extraction of valuable themes among them. Furthermore, to preserve the relations elicited through grounded theory, the cluster analysis was conducted using the native XMI formatted files extracted from the models defined previously in Sec. 3.3.1.

For this topic modelling exercise, log-likelihood was used to assess clustering appropriateness. We began with the same number of clusters as the target study (k = 10 clusters) but that number was increased until at least one of the newly-emerging clusters contained less than half of the mean population of factors in the previous round. This approach was aimed at allowing the extraction of themes that were meaningful, i.e., they reflected semantic commonalities among factors. In addition, We used hyperparameter tuning over LDA hyperparameters alpha and beta [10]. To conduct all the above pre-processing and analyses we exploited the NetCulator bibliometric analytics tool[3] which supports LDA and a number of similar natural-language analyses and clustering techniques.

4 RESULTS

4.1 A Grounded Theory of Success and Failure: General Overview

The entire grounded theory we elicited cannot be trivially represented and reported here because of its size and extensive detail. The grounded theory counts 561 factors and 40 core concepts in

3. https://www.netculator.com/

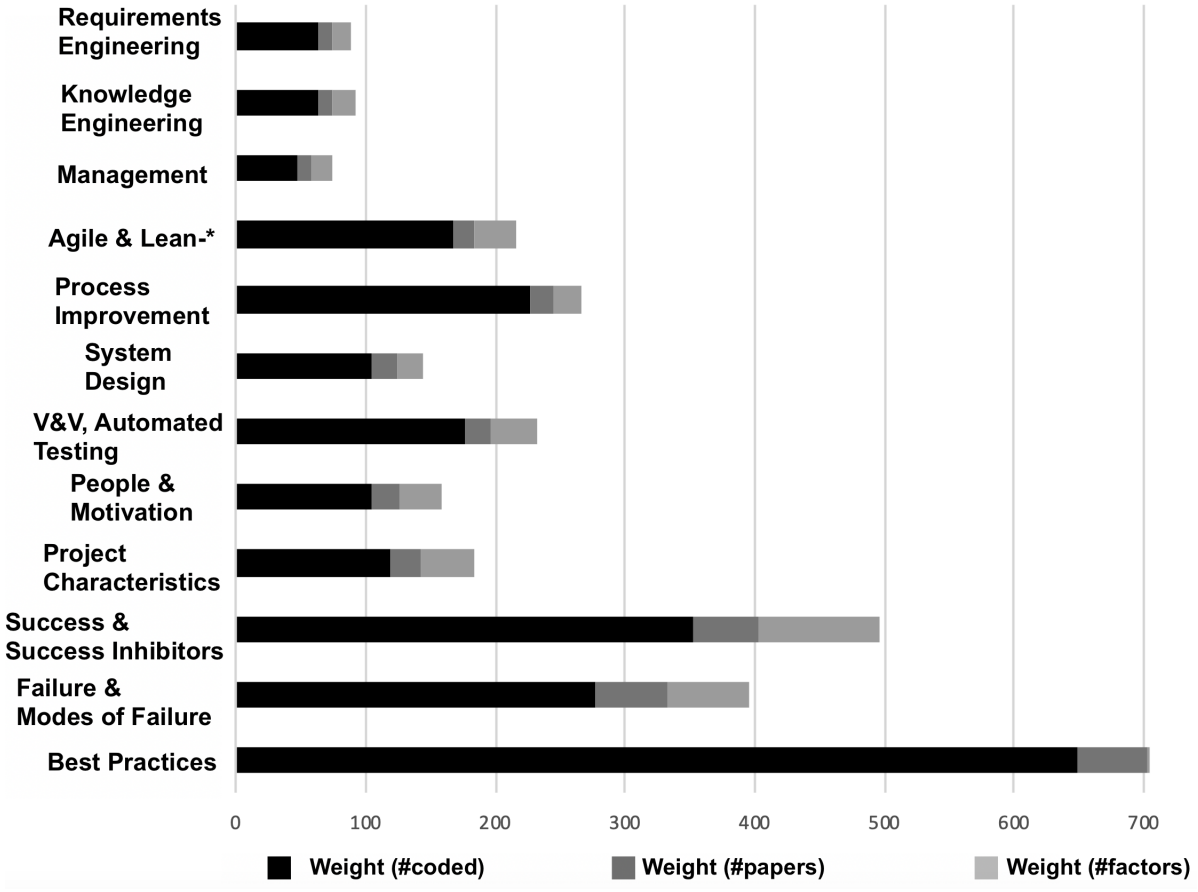

Fig. 3. Grounded Theory Content Analysis.

total, linked by 84 co-occurrence relations. However an overview is available to browse as an online image.[4] Furthermore, the grounded theory is available for further study as part of our replication package in three formats: MagicDraw resident UML format, XMI 2.11, and PDF.

In the scope of this study and to address $\mathbf{RQ}_1$ we offer an outline of the core-concepts and their content analysis [11]. Figure 3 plots an overview of the most frequently occurring core-concepts captured with the method as described in Sec. 3.3.1. This plot shows the clusters ordered from top to bottom by increasing number of coded papers per cluster; every bar reports the stacked numbers of (1) coded occurrences, (2) papers in which the codes were applied for the core-concept, and (3) number of factors reported for the core concept. Occurrences reported on 9 papers or fewer were omitted for the sake of readability.

The clusters we report reflect an equal mix of typical software lifecycle phases (e.g., *requirements engineering*, at the top of Fig. 3) as well as practices used in those phases (e.g., *V&V and Automated Testing*). Moreover, the clusters reflect varied levels of abstraction among core concepts. Remarkably, the most frequent code is in the low-abstraction spectrum of the aforementioned level; according to our analysis the application of best practices as well as their success appraisal (bottom of the plot) is the most frequent code cluster. This evidence reflects that best practices as a construct of software engineering shows a presence which is comparable to the most frequent core-concepts. This indicates, (1) a gap in the levels of abstraction concerning both success and failure as evident from the state of the art and (2) a relative distance in the depth of knowledge in the respective clusters found. Specifically, the clusters reflect the definitions outlined below:

4. https://tinyurl.com/y79hfvby

1) *Requirements Engineering.* Factors in this cluster address the creation, processing, resolution, traceability or quality of requirements as well as any factors influencing any phases of their lifecycle. Example factors include *requirements validation by end users* as well as *use of adequate language with the stakeholders*.
2) *[Software] Knowledge Engineering.* Factors in this cluster address the creation and retrieval of knowledge and artifacts of a software design, its implementation, and its operations. Sample factors include *tacit knowledge* as well as *knowledge brokers*.
3) *Project Management.* Factors in this cluster address the fallacies and pitfalls manifesting during, or related to the role of project management. Factors include the *choice of software development model* as well as *post-mortem analysis*.
4) *Agile and Lean-*.* Factors in this cluster address any positive or negative characteristics agile tenets, according to the definitions in Schwaber [12] and Kumar [13]. Sample factors include developer software production *worflow awareness* as well as *human agile metrics*.
5) Process Improvement. Factors in this cluster refers to the quantities and qualities of software processes that can be tracked and measurably improved. For example, the *adoption of a common vocabulary* or the development of a shared *vision*.
6) *System Design.* Factors in this cluster refer to the pros, fallacies, and pitfalls surrounding or in connection to a system's design and designers. For example, factors include *software design reviews* as well as *detailed design verification*.
7) *Verification & Validation, Automated Testing.* Factors in this cluster refers to connections between software success/failure and its V&V processes and tools. Sample factors include *design-for-testability* and *ensuring test coverage*.
8) *People & Motivation.* Factors in this cluster address the human and organizational issues of people and their motivations. Sample factors include *low staff turnover* and *supportive relationships*.
9) *Project Characteristics.* Factors in this cluster cover characteristics of projects and their role as mediators for success and failure. Sample factors include *project lifespan* and *object-level concurrency control*.
10) *Success and Success inhibitors.* This cluster contains success drivers and factors contributing to its definition and its inhibition. Such factors include *errors in tracking the actual costs and debts* as well as *dependency on other projects*.
11) *Failure and modes of failure.* This cluster contains failure drivers and factors contributing to software failure as well as modes of failure reported in literature. Factors include *failure reticence* and *wrong automation of manual processes*.
12) *Best-Practices.* This cluster contains factors relating to the definition, application, and successful appraisal of *best practices*. Out of all the clusters, this one the least abstract, containing just three sub-factors: (1) *the degree of dissemination and use of best practices*; (2) *the appraisal of successful use of best practices*; (3) *preconditions for best practice use*.

4.2 Success and Failure Distilled: Topic Modelling Results

The results of the topic modelling exercise are recapped in Fig. 4 and Fig. 5.

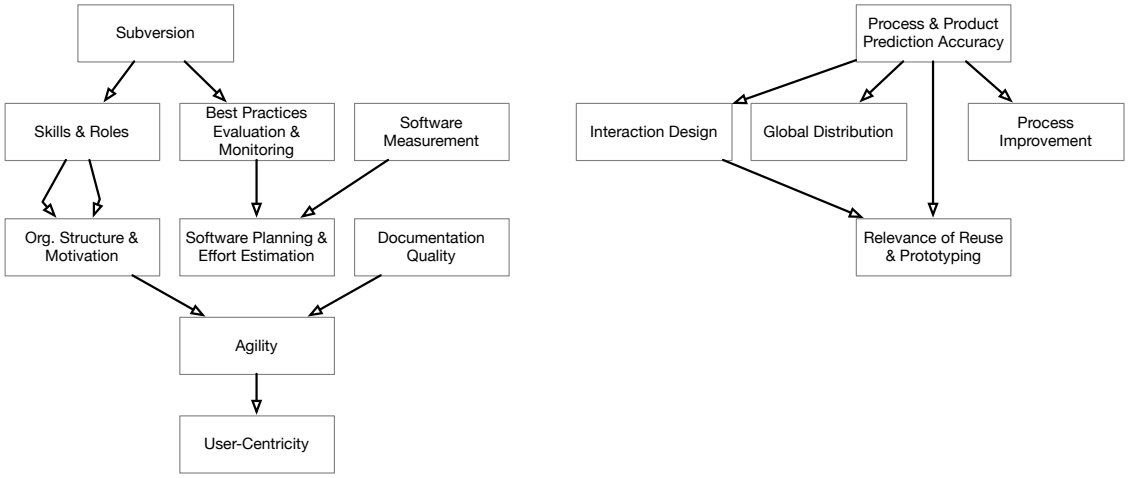

Fig. 5. Summary of Topic Modelling Results.

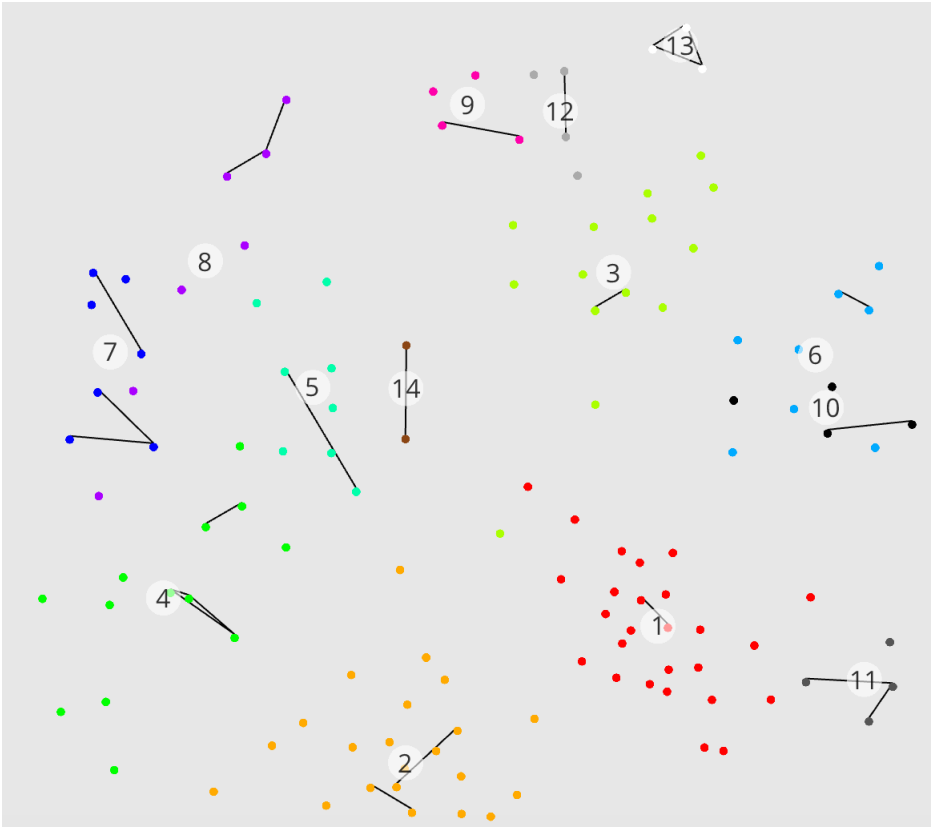

Fig. 4. Topic Modelling Clustering topology analysis [14] results.

Fig. 4 outlines the raw results of the 14 themes emerging from topic modelling. Dots indicate most-probable word radixes belonging to each theme (not reported on the figure for the sake of brevity but highlighted later in this section); edges reflect relations among core concepts from our grounded theory exercise.

On the other hand, figure 5 reports a manual representation of the themes emerging from the topic modelling exercise. Names for the themes were chosen independently by two analysts, with subsequent conflict resolution ($K_{alpha} = 0.89$). For the manual creation of the figure we also used

the relations previously reported as part of the grounded theory exercise (directed arrows in Fig. 5). However, for the sake of visualization, all relations were collapsed into a single (unweighted) arrow linking the clusters occurring in each relation. Based on the relations, the emerging sets of themes self-arranged into two domain areas that delimit the phenomena under study (success and failure of software engineering projects).

The area on the left-hand side of Fig. 5 incorporates *people* themes (subversion [15], organizational structure and motivation [16], agility [17]) as well as themes that discuss *internal software product characteristics*, that is, themes of characteristics of the product which are not perceived externally by end-users—specifically, the quality of documentation [18], user-centric design [19], software measurement [20], software planning and estimation [21]. Finally, this area contains *best-practices evaluation and monitoring* which is often considered orthogonal to all of the above but empirically is linked to the emergence of subversion [22] and is reportedly connected to software planning and effort estimations.

The area on the right-hand side of Fig. 5 incorporates *process-specific* themes (process improvement [23], accuracy of automated quality predictions [24]) as well as *external product* themes, that is, characteristics of the software product that are perceived externally—specifically, the distribution of its software process [25], its interaction design characteristics [26] as well as the extent to which external products have contributed to that product through prototyping and reuse [27].

The themes emerging in both domain areas are fleshed out in the following subsections, arranged left-to-right, and top-to-bottom following the contents of Fig. 5. We provide definitions of themes and, as is typical for LDA-based topic modelling, we offer the list of the most important terms as determined by the algorithm (arranged by decreasing rank with a cut-off below 20% probability) for each theme.

4.2.1 Domain Area 1: People, Internal Software Characteristics, Best Practices

1) **Subversion.** The concept of subversion refers to concepts and challenges of subversize stakeholders previously introduced by Ross and Glass [15]. In the scope of our topic modelling exercise, this theme corresponds to most recurrent words and substrings (using wildcards) specified as follows: *friction*, restrict*, *communication*, *cooperation*, disregardof*, lackof**;
2) **Skills & Roles.** The concept of software skills and appropriate role management is still under investigation from several perspectives [28], [29], though most prominently from an educational viewpoint. Concerning the theme, most recurrent words and substrings we reported are: *soft*, *motivation, coaching, *experience, domain*, trust, *size, core-*, connected**;
3) **Best Practices Evaluation & Monitoring.** We reported a strong presence of factors and themes relating to the application, appraisal, and effectiveness measurement of best practices, intended as *recurrent solutions for known and established problems* [30]. Most recurrent words and substrings for this topics are: *defect-*, accura*, change-*, *prediction, *accuracy, *success, *practice*;
4) **Software Measurement.** Software measurement [20] is a key activity in the scope of software engineering research. Most recurrent words and substrings concerning this theme within the scope of our results are as follows: *quality, define*, instrument*, customer*, stakeholder*, *community*, *interpretation;
5) **Org. Structure & Motivation.** Organizational structure refers to the graph of recurrent, explicit or implicit relations of coordination, co-operation, and communication relations occurring among individuals in an endeavor [16]. The terms occurring for this theme reflect a prominent role of motivation as a driving force. Specifically, recurrent words and substrings are: *turnover, *motivated, environment, feedback, recognition*, *motivator**;

6) **Software Planning & Effort Estimation.** A well-established area of software engineering research and practice, software planning and effort estimation are key activities in software engineering economics [31]. In the scope of our work, most recurrent words and substrings relating to this theme are: *misuse, earn*, staff*, governance*;
7) **Documentation Quality.** From the perspective of software maintenance & evolution, documentation is a discriminant in successful or failing software projects [32], [33], [34]. We obtained the following recurrent words and substrings for this theme: **knowledge*, domain*, requirements*, formal*, granularity, broker*, post-mortem*;
8) **Agility.** Agility clearly relates to the use, level of, and confidence around agile methods [35]. The adoption of agile methods is an established fact in software engineering literature [36]; however, the factors that lead to successful or failing attempts at harnessing agile methods are still left largely to speculation. In the scope of our topic modelling exercise, the following terms were reported: *self*, user*, value*, pressure*, pair*, test*, human**;
9) **User-Centric Design.** Finally, in the scope of topics relating to people, internal software characteristics, as well as best practices, we reported several factors and recurrent keywords relating to user-centric design [37], that is, the framework of engineering where usability goals, user characteristics, environment and workflows are given attention at each stage of the (software) design process. Many of the keywords reported for this theme relate to how practices from this framework lead to successful or failing engineering attempts. Specifically, words and substrings reported are: *persona*, communit*, organization*, usabilit*, integrat*, context**;

4.2.2 Domain Area 2: Processes and External Product Characteristics

1) **Process & Product Quality Prediction Accuracy.** This theme relates to the accuracy with which a quality prediction is made or appraised in the scope of software engineering research [38], [39]. Several works from the literature have touched upon this topic, most prominently along the lines of defect prediction [24] and similar endeavours. Words and substrings featured in this theme are: *histor*, objective*, improvement, additional*, technolog**;
2) **Interaction Design.** Interaction design refers to the design of interactive products and services in which design focus goes beyond the product under development and includes the ways users are likely to interact with that product [40]. Although not a common software engineering topic of focus, interaction design reflects several keywords occurring frequently in general software engineering literature, most prominently: *socio-*, man-machine*, cognitive*, anthropo*, bond*, operation**;
3) **Global Distribution.** Global distibrution in the scope of the themes emerging from our topic modelling refers to the general sub-field of software engineering that studies globally-dispersed teams as part of global software engineering and development [41], [42]. The most frequent words and substrings relating to this theme are: *remote*, geograph*, standard*, expan*, distribut*, multi*, organization**;
4) **Process Improvement.** Process improvement refers to the segment of software engineering research and practice dedicated to appraising and improving the quality of software processes [43], [44]. In the scope of our topic modelling, words and substrings relating to process improvement are: *progress*, train*, ad-hoc, capabilit*, principl*, chang*, need*, expectation*, assess**;
5) **Reuse & Prototyping.** The last emerging theme out of topic modelling reflects the role of software reuse and rapid prototyping as strategies for software engineering, where *reuse* indicates the recycling of existing software assets into a new or evolved version of a software product [45] while *prototyping* reflects the preparation of mock-ups for exploratory requirements engineering [46]. Key terms for this theme are: *decreas*, upgrad*, reverse, cost**.

5 Usage, Implications and Threats to Validity

5.1 Discussion

Our results indicate that the phenomena of software success and failure are extensive and span a large variety of factors and themes, not all of which are currently measured or tracked. Furthermore, there seems to be a mismatch or some form of *failure reticence* in the field, since the literature reports a majority of studies focused on software success as opposed to failure.

We conclude that further research should be dedicated into both the phenomena under study, but emphasize that such research should elaborate more on the phenomena associated with software failure, the factors entailed, and their many relations and ramifications.

Stemming from previous studies, we renew the conclusions of those studies with our own data and observations. In addition, we provide three other observations:

1) *Creating and Validating Instruments for Measuring Success* — we confirm this finding from multiple perspectives. For example, we discovered that the correct use and appraisal of best practices in software engineering is least understood and yet such understanding is urgent since it often mediates software failure and success altogether.
2) *Representative Sampling Without Population Lists* — although we did not conduct any specific analysis to confirm this finding, we did in fact report a relative paucity of methodological detail in about 70% of the papers that we surveyed. The lack of rigour and replicability compromises the generalisability of individual findings[5].
3) *Identifying Empirically Validated and Actionable Antecendents* — similarly to the previous point, we did not conduct any systematic analysis focusing on the antecedents in question but we did report a relative lack of dimensions, factors, and valid metrics from a considerable subset of the primary studies. Specifically about 60% of the primary studies do not conclude with measurable quantities to be tracked and improved.

Furthermore, the study highlights several other findings, most prominently on the importance of the dimensions of *subversion* around software, described in both this study and its precedent as the process whereby the values and principles of an established software engineering project are undermined, in an attempt to transform the social order and its structures of power, authority, hierarchy, and social norms in line with some desired end-state differing from the project goal. Our findings highlight a prominence of *subversive* dimensions. The existence and prominence of such dimensions further motivates streams of inquiry around social software engineering [47] and the quality of organisational and community structures [48], [49], [50] for software engineering.

5.2 Addressing the Research Questions

This study set out to address three research questions, namely: (1) What factors are reportedly connected to success or failure? (2) What themes emerge across such factors? And (3) What measurable quantities exist in themes that are not currently being measured? In addressing these research questions we reported, in the scope of $\mathbf{RQ}_1$, the following:

Answer to $\mathbf{RQ}_1$. There exist over 500 factors arranged in over 40 topical clusters of factors. Among these clusters, the most impactful in terms of occurrence and frequency (established via content analysis) range from software engineering phases such as *requirements engineering* to the use and effectiveness-appraisal of *best practices*. Further research can use the isolated

5. Nevertheless, to encourage replication, a comprehensive replication package is provided online here: https://figshare.com/s/e6f0968e55c2cd024389

clusters (and the factors therein) to devise tools and metrics for continuous monitoring and analysis.

Furthermore, in the scope of $\mathbf{RQ}_2$, we aimed to determine additional themes within the factors, beyond those found in our manual qualitative clustering. For this second endeavour, we reported the following:

Answer to $\mathbf{RQ}_2$. There exist 14 underlying themes among the over 500 factors in our analysis. Themes emerging from this analysis constitute essential risk engineering targets for successful software engineering.

Based on our results and the answers to both research questions, the two perspectives that may make practical use of the synthesis that we have provided reflect (1) practitioners' efforts in avoiding failure and (2) researchers' efforts in figuring out and measuring both success and failure.

On one hand, practitioners can focus on the factors (and clusters thereof, see Fig. 3) that reflect (1) *success and success inhibitors*, (2) *failure and failure modes* as well as (3) *best practices* and their evaluation. In so doing, practitioners can use the factors we provide as indicators to assess their project status and can plan and instrument corrective actions.

On the other hand, researchers can use the theoretical modelling exercise reported in Fig. 3 to further understand and potentially measure the factors, focusing on operationalising any factors that were not previously measured. At the same time, the topic modelling exercise we reported in Sec. 4.2 could be used as a basis to design, prototype, and evaluate automated computational intelligence [51] methods, tools, and techniques to automatically determine the status of software projects, e.g., analyzing data stemming from the DevOps pipelines around such projects.[6]

Finally, in the scope of $\mathbf{RQ}_3$, we set out to identify the dimensions emerging from the previous analyses which, to date, do not have any automated means of measurement, tracking, and improvement in software engineering research and practice. To address this gap, we elaborated a quality model [52] obtained by identifying the factors from our study ($\mathbf{RQ}_1$ and $\mathbf{RQ}_2$) which are currently *not* supported by any artefact corresponding to the definition of a quality model [53]. A quality model establishes relationships between project quality outcomes (e.g., bug rates, issue resolution time, size and vigor of the community, etc.) and characteristics of the product and its community. The next section outlines this contribution in more detail.

5.3 A Quality Model for Unobserved Software Quality Dimensions

To address the gap identified by $\mathbf{RQ}_3$ we operated a simple systematic search of every keyword discovered as part of topic modelling (see $\mathbf{RQ}_2$, Sec. 4) along with the additional search string defined as follows:

$$\text{quality} \wedge (\text{model } V \text{ framework } V \text{ metric } V \text{ measure } V \text{ measurement } V \text{ analysis } V \text{ parameter});$$

As a result of this exercise, our model addresses 3 unobserved themes: (1) subversion; (2) organizational structure and motivation; (3) skills and roles.

We aggregated all metrics and empirically-investigated quantities from software engineering research that emerged from the systematic search above. The metrics and quantities involved are all related to features and characteristics of a social graph construct, known as *Developer Social*

6. https://dzone.com/articles/role-of-predictive-analytics-in-devops

Network, loosely defined by Meneely and Williams [54] as the superimposed communication and collaboration networks structures emerging during software development. The aforementioned construct was previously touched upon by several other research attempts, also in relation to software failure [55]. We reuse this construct as a reference to flesh out the metrics we discovered in literature that address the aforementioned observation gaps. A total of 38 metrics were found.

An elaboration in full detail of all the 38 metrics for of each quality category featured in the model is outside of the scope of this contribution, which is aimed at offering an aggregate quality model rather than a detailed treatise or synthesis of each factor.[7]

The emerging quality model features 5 categories of previously-defined, validated metrics that can aid the observability of *subversion*, *organizational structures & motivation*, as well as *skills & roles*. These metrics span: (1) developer social networks (DSNs) — these mainly reflect population metrics applied in the context of DSNs [57]; (2) socio-technical — these mainly reflect quantities that were introduced to relate communication (i.e., information interchange) and collaboration (i.e., co-operated action over software artefacts) together, most prominently socio-technical congruence [58]; (3) core-community members — these mainly reflect the difference between features in the *core* and *periphery* of the network structure [59], [60]; (4) turnover — these mainly reflect the degrees of freedom or variability of members within the DSN; (5) social networks analysis (SNA) — these mainly reflect the use of "classical" SNA metrics that were previously applied in the context of software engineering [61]. To address $\mathbf{RQ}_3$ we argue as follows:

Summary for $\mathbf{RQ}_3$. There exist three themes emerging from our systematic literature analysis that are currently not supported by a full-fledged quality model. They are: (1) subversion; (2) organizational structure and motivation; (3) skills and roles. Nevertheless, there exist in the literature a considerable number of metrics to address the aforementioned gaps. These metrics are openly available online [56] and reflect 5 categories of quality that need to be explicitly tracked to monitor the extent of software success and to ward off software failure. The proposed quality model can be used in conjunction with established technical, process or other quality models for software engineering practice.

5.4 Observations and Implications

First, from a purely statistical perspective, the clusters and themes discussing *best practices*—their evaluation and monitoring as well as software success and failure—were the most popular ones emerging from this study. Furthermore, these themes and clusters emerged *both* from (1) topic modelling and (2) grounded theory. And this topic by far outweighed all others in terms of software engineering research and practice. This finding confirms what was previously reported in Mäntylä et al. [3]. *Further research should thus be dedicated to establishing this research cluster/theme as a research topic in its own right.*

Second, based on the extent of our data (500+ factors over 40+ clusters), software success and failure are vast phenomena which deserve dedicated software engineering research on their own. Specifically, the dimensions and factors along which success (or failure) unfold need statistically significant factor analysis using time-series analysis [62] or similar approaches to effectively establish what factors and dimensions contribute to or facilitate success. Conversely, our data indicates that we know much more about success than we do about failure (e.g., see Tab. 3). The number of codes applied for the core concepts of *success* and *failure* differ by almost 2 to 1 and the

7. For complete details, the reader may refer to [56] which contains a complete overview of all factors in the quality model, their operationalisation, and their implementations in practice.

number of papers in which these codes were applied is 1.7 times higher for *success*. To address this gap like other engineering disciplines, software engineering research should dedicate research to establish more background knowledge on software failure (e.g., reflecting post-mortem analysis [63], empirical software failure research, fault lines [64], etc.). *In summary, further research along this line should be dedicated to better understanding software failure, perhaps starting from well-known cases of software failure, e.g., in open-source. Specifically, open-source phenomena such as forge failure, community forking, and sustainability beyond forks are still not widely studied and thus deserve further empirical and experimental research.*

Finally, our 3 **RQ**s together amount to a single key message: *software engineering is a perilous game of equilibrium over as many as 500+ degrees of freedom. Constant feedback loops between all areas of the organisational and technical structures involved, be they open- or closed-source, is required to maintain this equilibrium. Sustaining these feedback loops by any means necessary should be a key goal for future software engineering research.*

5.5 Threats to Validity

The conclusion provided by our study might have been threatened by a two main factors: the collection of a complete set of papers on the subject of interest and the way we analyzed the collected sources to provide new knowledge.

In the first instance, the major challenge of a systematic literature review is that of finding a comprehensive set of papers to study and analyze. In our case, we built a search string that not only included keywords coming from the reference work of Mäntylä et al. [3], but also aimed at retrieving papers offering results stemming from industrial practice and experience. Using this strategy, we were able to survey the literature on success and failure more comprehensively and from different perspectives. In so doing, we queried all major databases currently available in software engineering research, hence increasing the comprehensiveness of our research. Furthermore, it is worth noting that two authors jointly scanned each of the papers coming from the application of the search string with the aim of (1) assessing its fitting to the goals of the paper, thus discarding non-relevant ones by means of the exclusion criteria defined in Section **??** and (2) increasing the overall reliability of the methodological procedure, by conducting a joint effort in evaluating it.

When analyzing the sources retrieved after the application of the search string, we applied formal grounded theory methods to let emerge themes related to software engineering success and failures. To increase the reliability of the applications of such a methodology in our context, two authors of this paper have jointly performed the task: they analyzed each of the retrieved sources to understand concepts and assign codes. Furthermore, to ensure internal and construct validity even further, the set of codes for grounded theory was later double-checked by an external researcher having more than 10 years of research experience, who fully confirmed the initial codes assigned by the two authors of this paper. With these steps, we aimed at increasing the overall validity and reliability of the reported results; nevertheless, we cannot exclude possible imprecision and/or subjective judgment that may have played a role in the elaboration of the codes. For these reasons, we make our data publicly available to enable further replications and verification of our analyses.

6 Conclusions

This section reports on the practical usage of the results achieved in our study and outlines our future research agenda on the topic.

6.1 Results Usage in Practice

From a more practical perspective, the results provided in the previous pages can be used in at least four practical scenarios.

First, practitioners steering their own software engineering endeavours can use the overview provided in Fig. 3 and 5 to understand the potential areas at risk within their software projects. Later, once these areas are understood, practitioners can use the more fine-grained and detailed grounded theory to pick and choose which factors are known inside those sensitivity areas. In the same vein, practitioners can also bootstrap new software engineering endeavours providing an appropriate software risk analysis starting from the results we have provided.

Second, practitioners can use the metrics and indicators accounted for in our grounded-theory or any of its syntheses in this manuscript as input for organizational quality tracking and continuous improvement, just as technical metrics are used to track and improve software coding practices. In line with this contribution, we have designed and implemented a research tool to automate the elicitation and analysis of such metrics. This tool is being refined based on a fork of the Siemens CodeFace tool [8] and is currently under experimentation.[9]

Third, practitioners and software vendors active in the quality assurance software tools market segment can use the factors and reference analyses in the scope of our **RQ**s to refine their tools in line with the findings of this study or even devise new tools to support the unobservable dimensions isolated as part of our response to $\mathbf{RQ}_3$.

Fourth, practitioners can conduct a self-assessment of their software projects with respect to the factors we summarized in the previous sections. A rudimentary risk self-assessment methodology entails at least the following steps:

1) Download the grounded-theory model we have provided online;[10]
2) Use the model as a checklist to assess whether failure-inducing factors (those marked with an empty circle linked note reporting the papers discussing them) may be leading to risks of failure;
3) Use the model as a checklist to assess whether success-facilitating factors (those marked with a filled circle linked note reporting the papers discussing them) are reflected in the project under study;
4) Elaborate the total risk of failure as follows:
   a) **Elaborating the Known Risks.** Subtract the *positive knowns*, that is, the sum of known success-facilitating factors exerting an observable effect on the project from the *negative knowns* exerting an observable effect on the same project. This is reasonable since risk is higher if negative factors are manifested but can be lowered to the degree that positive and success-inducing factors are manifested;[11]
   b) **Elaborating the Unknown Risks.** Sum together any remaining negative and positive *unknowns* from the model. This is reasonable since the risk of failure is higher the more factors' effects are unknown to an observer, regardless of whether those effects are positive or negative;
   c) **Elaborating a Grand Total.** Sum together the two compounding quantities above.

The steps entailed in (4.a-c) allow practitioners to get a rough evaluation of the risk coverage for the project under study. More formally:

Software Failure Risk:

8. http://siemens.github.io/codeface/

9. The tool is available and free to use under the Apache 2 license agreement: https://github.com/maelstromdat/CodeFace4Smells

10. https://tinyurl.com/y79hfvby

11. This assumes that all factors have an equal mutual effect, which is obviously an open research question.

$$\rho = \sum[(P_n - N_n) + U];$$

where P_n indicates the *positive knowns*, while N_n indicates the *negative knowns* and U indicates any remaining *unknowns*, e.g., accounting for contingency management and preparedeness planning. The above methodology and the basic formula are to be seen as a rudimentary starting point for further experimentation, which is beyond the scope of this study. However, we are planning several applications of the aforementioned methodology and formula in action in industry to elaborate more on its construct and external validity.

6.2 Synthesis and future work

This paper builds upon previous studies of the complex phenomena of software success and failure. The literature in question focuses on the software engineering domain and covers a broad range of perspectives over the discipline. In this paper we have presented a more extensive and rigorous analysis of the literature, by executing 3 analyses aimed at deepening our understanding of software success and failure. The 3 analyses reflect: (1) a grounded theory of the phenomena under study; (2) the emergent themes hidden beneath such a theory; (3) the measurable quantities from software engineering research that account for previously unobserved themes and factors from analyses (1) and (2) above. In the future we plan to further analyse the data and factors produced as part of our research question 1, e.g., to offer automated means of classification for the factors. Furthermore, we plan to analyse the data in our replication bundle for the purpose of generalising a more refined taxonomy or *ontology* for the purpose of instrumenting automated reasoning and risk analysis (e.g., to support post-mortem analysis).

Finally, we plan to refine and further evaluate tool support to track as many factors from our grounded-theory and themes as possible, automating their investigation from openly available application lifecycle management (ALM) tools of common use during software development, such as quality metrics suites, issue-tracking systems, CI/CD pipelines, and more. A vision for how this might be realized is presented in previous work [65]. Specifically, the unobserved themes emerging from this study could be supported for specifically-tailored holistic DataOps [66] software process, product, and people analysis ALM suite [67] acting as an integrated predictive analytics solution working continuously towards modelling success and failure by means of machine-learning and similar advanced computational intelligence. In this vein, all the dimensions elaborated in our grounded-theory could be supported by specific predictive modelling computational intelligence while a holistic ALM suite back-end could be trained as an *ensemble* method to assemble the individual predictions towards an aggregated series of fundamental scores, thus instructing all software stakeholders in their next steps. For example, see the recap in Fig. 6; the figure outlines our future work towards a DevOps analytics suite which could be considered *omniscient*, that is, acting towards most if not all of the dimensions accounted in the grounded-theory proposed in this work across all dimensions we highlighted, namely, the individuals dimension, their social-interactive communitarian dimension, the organisational layer combining them as well as the technical layer towards which their work is aimed.

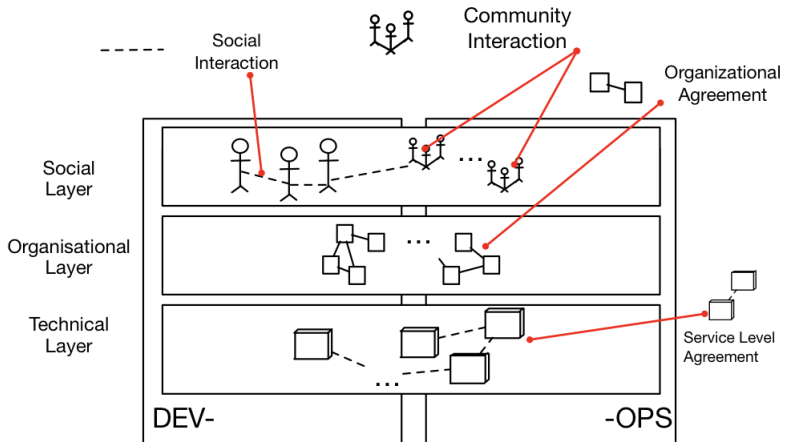

Fig. 6. Omniscient DevOps Analytics; concept tailored from [65].

ACKNOWLEDGEMENT

The authors would like to thank Dr. Francesco Castri for his invaluable effort and work in producing and allowing for the inter-rater reliability assessment of the grounded theory reported in this paper. Furthermore, the authors would like to thank Dr. Massimiliano Di Penta, Dr. Henry Muccini, Dr. Patrizio Pelliccione, and Dr. Marco Di Memmo for their precious insights, explanatory, and confirmatory discussions in the scope of addressing the threats to validity of this work. Finally, some of the authors' work is partially supported by the European Commission grant no. 0421 (Interreg ICT), Werkinzicht, the European Commission grant no. 787061 (H2020), ANITA, European Commission grant no. 825040 (H2020), RADON, European Commission grant no. 825480 (H2020), SODALITE, and the Swiss National Science Foundation through the SNF Project No. PZ00P2_186090 (TED). Finally, we would like to thank the associate editor and anonymous reviewers for the detailed and constructive comments on the preliminary version of this paper, which were instrumental to improving the quality of our work.

APPENDIX

- Downloadable Dataset: figshare.com/s/e6f0968e55c2cd024389;
- Primary studies[12]: www.dropbox.com/s/3lj4h0otpt7jrxe/SLRWorkingPortfolio.pdf?dl=0;
- GT codes list and definition: www.dropbox.com/s/ia0yv225r8jcggr/code\%20list.xlsx?dl=0

REFERENCES

[1] Paul Ralph and Paul Kelly. The dimensions of software engineering success. In Pankaj Jalote, Lionel C. Briand, and Andr van der Hoek, editors, *ICSE*, pages 24–35. ACM, 2014.
[2] Timo OA Lehtinen, Mika V Mäntylä, Jari Vanhanen, Juha Itkonen, and Casper Lassenius. Perceived causes of software project failures–an analysis of their relationships. *Information and Software Technology*, 56(6):623–643, 2014.
[3] Mika V. Mäntylä, Magne Jørgensen, Paul Ralph, and Hakan Erdogmus. Guest editorial for special section on success and failure in software engineering. *Empirical Software Engineering*, 22(5):2281–2297, 2017.
[4] Netta Iivari. Enculturation of user involvement in software development organizations - an interpretive case study in the product development context. pages 287 – 296, Tampere, Finland, 2004. ACM Press.
[5] ISO/IEC Standard. Software engineering – product quality – part 1: Quality model. ISO Standard 9126-1, ISO/IEC, 2003.

12. please consider that all copies are copyrighted and were downloaded for research purposes only.

[6] Marco Torchiano, Filippo Ricca, and Alessandro Marchetto. Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold. In Giulio Concas, Ewan D. Tempero, Hongyu Zhang, and Massimiliano Di Penta, editors, *WETSoM*, pages 12–18. ACM, 2011.

[7] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11, New York, NY, USA, 2008. ACM.

[8] Barbara Kitchenham and Stuart Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. School of Computer Science and Mathematics, Keele University, UK, July 2007.

[9] Juliet Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

[10] Amritanshu Agrawal, Wei Fu, and Tim Menzies. What is wrong with topic modeling? (and how to fix it using search-based se). *CoRR*, abs/1608.08176, 2016.

[11] Hsiu-Fang Hsieh and Sarah E Shannon. Three approaches to qualitative content analysis. *Qualitative Health Research*, 15(9):1277–1288, 2005.

[12] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, Redmond, WA, 2004.

[13] D.R. Kumar. Lean Software Development. *The PROJECT PERFECT White Paper Collection*, 2005. http://www.projectperfect.com.au/downloads/Info/info_lean_development.pdf.

[14] Roland Kwitt, Stefan Huber, Marc Niethammer, Weili Lin, and Ulrich Bauer. Statistical topological data analysis - a kernel perspective. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *NIPS*, pages 3070–3078, 2015.

[15] Johann Rost and Robert L. Glass. The impact of subversive stakeholders on software projects. *Commun. ACM*, 52(7):135–138, 2009.

[16] Damian A. Tamburri, Patricia Lago, and Hans van Vliet. Organizational social structures for software engineering. *ACM Comput. Surv.*, 46(1):3:1–3:35, July 2013.

[17] Noura Abbas. *Software quality and governance in agile software development.* PhD thesis, University of Southampton, UK, 2009. British Library, EThOS.

[18] G. Garousi, V. Garousi, M. Moussavi, G. Ruhe, and B. Smith. Evaluating usage and quality of technical software documentation: an empirical study. In *17th International Conference on Evaluation and Assessment in Software Engineering*, Porto de Galinh, Brazil, 2013.

[19] Ahmet Soylu, Patrick De Causmaecker, Davy Preuveneers, Yolande Berbers, and Piet Desmet. Formal modelling, knowledge representation and reasoning for design and development of user-centric pervasive software: a meta-review. *IJMSO*, 6(2):96–125, 2011.

[20] Norman E. Fenton. Software measurement: A necessary scientific basis. *IEEE Trans. Software Eng.*, 20(3):199–206, 1994.

[21] M. Nasir. A Survey of Software Estimation Techniques and Project Planning Practices. In *Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006*, pages 305–310. IEEE, June 2006.

[22] Damian A. Tamburri, Remco De Boer, Elisabetta Di Nitto, Patricia Lago, and Hans van Vliet. Dynamic networked organizations for software engineering. In *Proceedings of the 2013 International Workshop on Social Software Engineering*, SSE 2013, pages 5–12, New York, NY, USA. ACM.

[23] Hideto Ogasawara, Takumi Kusanagi, and Minoru Aizawa. Proposal and practice of software process improvement framework - toshiba's software process improvement history since 2000. *Journal of Software: Evolution and Process*, 26(5):521–529, 2014.

[24] Sad Assar, Markus Borg, and Dietmar Pfahl. Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy. *Empirical Software Engineering*, 21(4):1437–1475, 2016.

[25] Christof Ebert, Marco Kuhrmann, and Rafael Prikladnicki. Global software engineering: An industry perspective. *IEEE Software*, 33(1):105–108, 2016.

[26] Matt Bellingham, Simon Holland, and Paul Mulholland. A cognitive dimensions analysis of interaction design for algorithmic composition software. In Benedict du Boulay and Judith Good, editors, *PPIG*, page 18. Psychology of Programming Interest Group, 2014.

[27] Stuart Harvey Rubin. Knowledge-based software prototyping and reuse. *International Journal on Artificial Intelligence Tools*, 6(1):127–147, 1997.

[28] Raphael Pham. *Improving the software testing skills of novices during onboarding through social transparency.* PhD thesis, University of Hanover, Germany, 2016.

[29] Hee-Dong Yang, Hye-Ryun Kang, and Robert M. Mason. An exploratory study on meta skills in software development teams: antecedent cooperation skills and personality for shared mental models. *EJIS*, 17(1):47–61, 2008.

[30] Christof Ebert and Reiner Dumke. Best practices in software measurement establish - extract - evaluate - execute, 2007.

[31] B.W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[32] Andrew Boyd. A goal-based approach to the evaluation and documentation of business process re-engineering. *Aslib Proceedings*, 56(5):286–300, 2004.

[33] V. Borja, J.A. Harding, and K.T.K. Toh. Product re-engineering process using an enterprise modelling architecture. *International Journal of Agile Management Systems*, 2(3):214–225, 2000.

[34] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *SIGDOC '05: Proceedings of the 23rd annual international conference on Design of communication*, pages 68–75, New York, NY, USA, 2005. ACM.

[35] Amr Elssamadisy and David West. Adopting agile practices: An incipient pattern language. In *Proceedings of the 2006 Conference on Pattern Languages of Programs*, PLoP '06, pages 1:1–1:9, New York, NY, USA, 2006. ACM.

[36] Anna Rohunen, Pilar Rodriguez, Pasi Kuvaja, Lech Krzanik, Jouni Markkula, and Burak Turhan. Agile adoption strategies in the context of agile in the large: Flexi agile adoption industrial inventory. In Alberto Sillitti, Angela Martin, Xiaofeng Wang, and Elizabeth Whitworth, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 48 of *Lecture Notes in Business Information Processing*, pages 397–398. Springer Berlin Heidelberg, 2010.

[37] Jeffrey Rubin. *Handbook of Usability Testing. How to Plan, Design, and Conduct Effective Tests*. Wiley, New York, 1994.

[38] M. Jörgensen. Experience with the accuracy of software maintenance task effort prediction models. 21(8):674–681, 1995.

[39] Li-Wei Chen and Sun-Jen Huang. Accuracy and efficiency comparisons of single- and multi-cycled software classification models. *Information & Software Technology*, 51(1):173–181, 2009.

[40] Alan Cooper and Robert Reimann. *About Face 3.0: The Essentials of Interaction Design*. Wiley & Sons, 3rd revised edition edition, 2007.

[41] James D. Herbsleb. Global software engineering: The future of socio-technical coordination. In Lionel C. Briand and Alexander L. Wolf, editors, *FOSE*, pages 188–198, 2007.

[42] Damian A. Tamburri, Elisabetta di Nitto, Patricia Lago, and Hans van Vliet. On the nature of the GSE organizational social structure: an empirical study. *proceedings of the 7th IEEE International Conference on Global Software Engineering*, pages 1–10, 2012.

[43] Robert B. Grady. *Successful Software Process Improvement*. Prentice Hall, 1st edition, 1997.

[44] Mahmood Niazi, David Wilson, and Didar Zowghi. Critical success factors for software process improvement implementation: an empirical study. *Software Process: Improvement and Practice*, 11(2):193–211, 2006.

[45] W. Frakes. Systematic software reuse: a paradigm shift. In *Proc. Int'l Conf. Software Reuse: Advances in Software Reusability*, pages 2–3. IEEE Press, 1994.

[46] Michael Thompson and Nina Wishbow. Prototyping: tools and techniques: improving software and documentation quality through rapid prototyping. In Roy MacLean, editor, *SIGDOC*, pages 191–199. ACM, 1992.

[47] Jessica Keyes. *Social software engineering*. Taylor & Francis, Auerbach Series, Boca Raton, FL, 2011.

[48] Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, Andy Zaidman, Francesca Arcelli Fontana, and Rocco Oliveto. How do community smells influence code smells? In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, pages 240–241, New York, NY, USA, 2018. ACM.

[49] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering*, pages 1–49, 2018.

[50] Damian A. Tamburri, Rick Kazman, and Hamed Fahimi. The architect's role in community shepherding. *IEEE Software*, 33(6):70–79, 2016.

[51] John Fulcher and Lakhmi C. Jain, editors. *Computational Intelligence: A Compendium*, volume 115 of *Studies in Computational Intelligence*. Springer, 2008.

[52] International Standard Organization (ISO). International standard iso/iec 9126, information technology product quality part1: Quality model, 2001.

[53] R. Geoff Dromey. A model for software product quality. *IEEE Trans. Software Eng.*, 21(2):146–162, 1995.

[54] Andrew Meneely and Laurie Williams. Socio-technical developer networks: should we trust our measurements? In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 281–290, New York, NY, USA, 2011. ACM.

[55] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. Can developer social networks predict failures? In *FSE '08: Proceedings of the 16th ACM SIGSOFT international symposium on Foundations of software engineering*, 2008.

[56] Simone Magnoni, Damian A. Tamburri, and Elisabetta Di Nitto. A socio-technical quality model for software engineering organisational structures: An empirical study. *Copyright - Politecnico di Milano Master Thesis Series*, https://goo.gl/Y9R4KY, 2016.

[57] Chintan Amrit and Jos van Hillegersberg. Mapping social network to software architecture to detect structure clashes in agile software development. In Hubert sterle, Joachim Schelp, and Robert Winter, editors, *ECIS*, pages 334–345. University of St. Gallen, 2007.

[58] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11, New York, NY, USA, 2008. ACM.

[59] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. Core and periphery in free/libre and open source software team communications. In *HICSS*. IEEE Computer Society, 2006.

[60] Chintan Amrit and Jos van Hillegersberg. Exploring the impact of socio-technical core-periphery structures in open source software development. *JIT*, 25(2):216–229, 2010.

[61] Mohammad Y. Allaho and Wang-Chien Lee. Analyzing the social networks of contributors in open source software community. In Przemyslaw Kazienko and Nitesh V. Chawla, editors, *Applications of Social Media and Social Network Analysis*, Lecture Notes in Social Networks, pages 57–75. Springer, 2015.

[62] Cesar Couto, Cristiano Amaral Maffort, Rogel Garcia, and Marco Tulio Valente. Comets: a dataset for empirical research on software evolution using source code metrics and time series analysis. *ACM SIGSOFT Software Engineering Notes*, 38(1):1–3, 2013.

[63] James A. Hager. Software cost reduction methods in practice: A post-mortem analysis. *Journal of Systems and Software*, 14(2):67–77, 1991.

[64] Keivan Bahmani, Zhaleh Semnani-Azad, Katia P. Sycara, and Michael Lewis. Team faultline measures: The effect of rescaling weights. In *HICSS*. AIS Electronic Library (AISeL), 2018.

[65] Damian A Tamburri, Dario Di Nucci, Lucio Di Giacomo, and Fabio Palomba. Omniscient devops analytics. In *International Symposium on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS)*. Springer, 2018.

[66] Julian Ereth. Dataops - towards a definition. In Rainer Gemulla, Simone Paolo Ponzetto, Christian Bizer, Margret Keuper, and Heiner Stuckenschmidt, editors, *LWDA*, volume 2191 of *CEUR Workshop Proceedings*, pages 104–112. CEUR-WS.org, 2018.

[67] Jzsef Klespitz, Mikls Bir, and Levente Kovcs. Cross-tool interoperability in heterogeneous application lifecycle management systems. In Hamido Fujita and George Angelos Papadopoulos, editors, *SoMeT*, volume 286 of *Frontiers in Artificial Intelligence and Applications*, pages 213–220. IOS Press, 2016.