


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

Damian A. Tamburri , Fabio Palomba , *Member, IEEE*, and 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. Recognizing the importance and impact of the topic, in this article we have executed a followup, more in-depth systematic literature review with additional analyses beyond what was previously provided. These new analyses offer: a grounded-theory of success and failure factors, harvesting over 500+ factors from the literature; 14 manually validated clusters of factors that provide relevant areas for success- and failure-specific measurement and risk-analysis; 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 this article. This article 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.

## I. 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 toward success. We pick up the challenge from where it was left off [3]. In this article, we refine and reexecute 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) 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 undersupported by measures. Third, we aim to elicit a rigorous quality model for these undersupported quantities.

Briefly, our results show that success and failure in software projects is mediated by more than 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 are 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 five 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 the data performed in this article.<sup>1</sup>

The rest of this article is organized as follows. Section II outlines the terminology used in this article. In Section III, we describe our research methods, while Section IV overviews the results achieved. Section V provides discussions on the key findings of the article. Finally, Section VI concludes this article.

Manuscript received September 12, 2019; revised January 14, 2020; accepted February 23, 2020. Date of publication June 1, 2020; date of current version February 5, 2021. This work was supported in part by the European Commission under Grant 0421 (Interreg ICT), Werkinzicht, in part by the European Commission under Grant 787061 (H2020), ANITA, in part by the European Commission under Grant 825040 (H2020), RADON, in part by the European Commission under Grant 825480 (H2020), SODALITE, and in part by the Swiss National Science Foundation through the SNF Project PZ00P2\_186090 (TED). Review of this manuscript was arranged by Department Editor Tugrul Daim. (Corresponding author: Damian Andrew Tamburri.)

Damian A. Tamburri is with the Eindhoven University of Technology, The Jheronimus Academy of Data Science, 5612 AZ Eindhoven, The Netherlands (e-mail: d.a.tamburri@tue.nl).

Fabio Palomba is with the University of Salerno, 84084 Fisciano SA, Italy (e-mail: fpalomba@unisa.it).

Rick Kazman is with the University of Hawaii & SEI/CMU, Honolulu, HI 96822 USA (e-mail: kazman@hawaii.edu).

Color versions of one or more of the figures in this article are available online at <https://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TEM.2020.2976642

<sup>1</sup>[Online]. Available: <https://figshare.com/s/e6f0968e55c2cd024389>

## II. SCOPE AND TERMINOLOGY

The scope intended for this article 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 the 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, this article 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 following section, we describe the research methodology we employed to conduct our followup systematic literature review.

## III. RESEARCH APPROACH

### A. Research Design

The *goal* of this article 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 (RQs)*.

**RQ<sub>1</sub>.** *What factors are reportedly connected to success or failure?*

**RQ<sub>2</sub>.** *What themes emerge across such factors?*

**RQ<sub>3</sub>.** *What themes are currently unobserved and what previously existing metrics can support these unobserved themes?*

Note that, in the scope of **RQ<sub>3</sub>**, by *unobserved* we mean the factors and themes emerging from **RQ<sub>1</sub>** and **RQ<sub>2</sub>** 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.

### B. 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 “practitioner”))

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:

- 1) IEEE Xplore digital library;<sup>2</sup>
- 2) ACM digital library;<sup>3</sup>
- 3) ScienceDirect;<sup>4</sup>
- 4) SpringerLink;<sup>5</sup>
- 5) Scopus;<sup>6</sup>
- 6) Engineering Village;<sup>7</sup>

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

<sup>2</sup>[Online]. Available: <http://ieeexplore.ieee.org>

<sup>3</sup>[Online]. Available: <https://dl.acm.org>

<sup>4</sup>[Online]. Available: <http://www.sciencedirect.com>

<sup>5</sup>[Online]. Available: <https://link.springer.com>

<sup>6</sup>[Online]. Available: <https://www.scopus.com>

<sup>7</sup>[Online]. Available: <https://www.engineeringvillage.com>

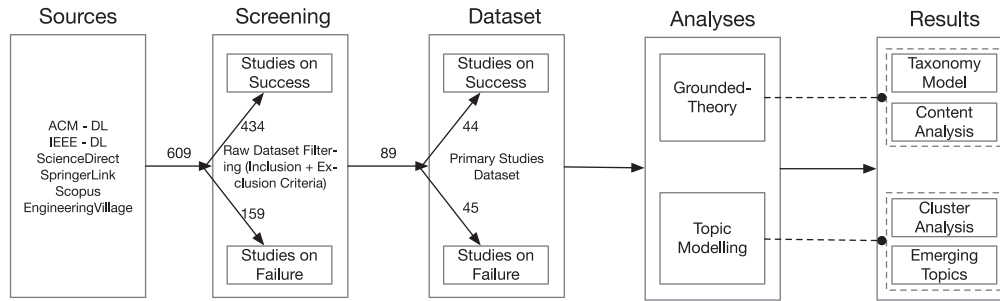


Fig. 1. Overview of our research design, from sources to results.

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 nonrelevant sources. Specifically, we filtered out:

- 1) papers that were not written in English;
- 2) papers whose full text was not available;
- 3) short papers (up to four pages) just reporting preliminary results;
- 4) papers from workshops;
- 5) papers that adopted the term “failure” to indicate software faults;
- 6) 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;
- 7) papers that described the failure or success of introducing new tools or processes, but that did not relate this to project success or failure;
- 8) papers referring to just one development phase rather than the entire software lifecycle;
- 9) papers that were about project success and failure, but that did not provide research results;
- 10) 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 article, 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<sup>8</sup> conferences and journals sponsored by ACM, IEEE, Springer, Elsevier, and Wiley.

### C. Analysis and Synthesis Methods

1) *Qualitative Analysis*: Analysis and synthesis of results were carried out through the well-known grounded theory (GT) approach [9].

<sup>8</sup>either class A\* or A or B from the CORE Edu Rankings portal <http://portal.core.edu.au/>

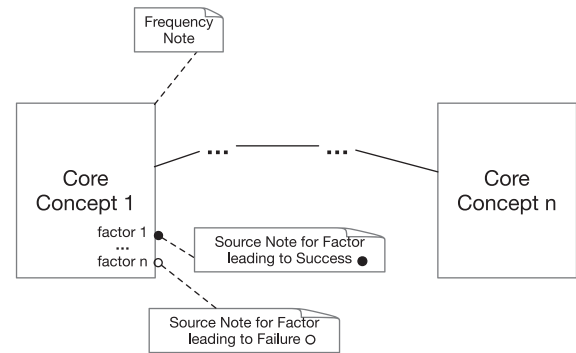


Fig. 2. Grounded theory as employed in this article.

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.

2) *Quantitative Analysis*: Following an approach similar to that proposed by Mäntylä *et al.* [3], we used the well-established topic modeling 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 Section III-C1.

For this topic modeling 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

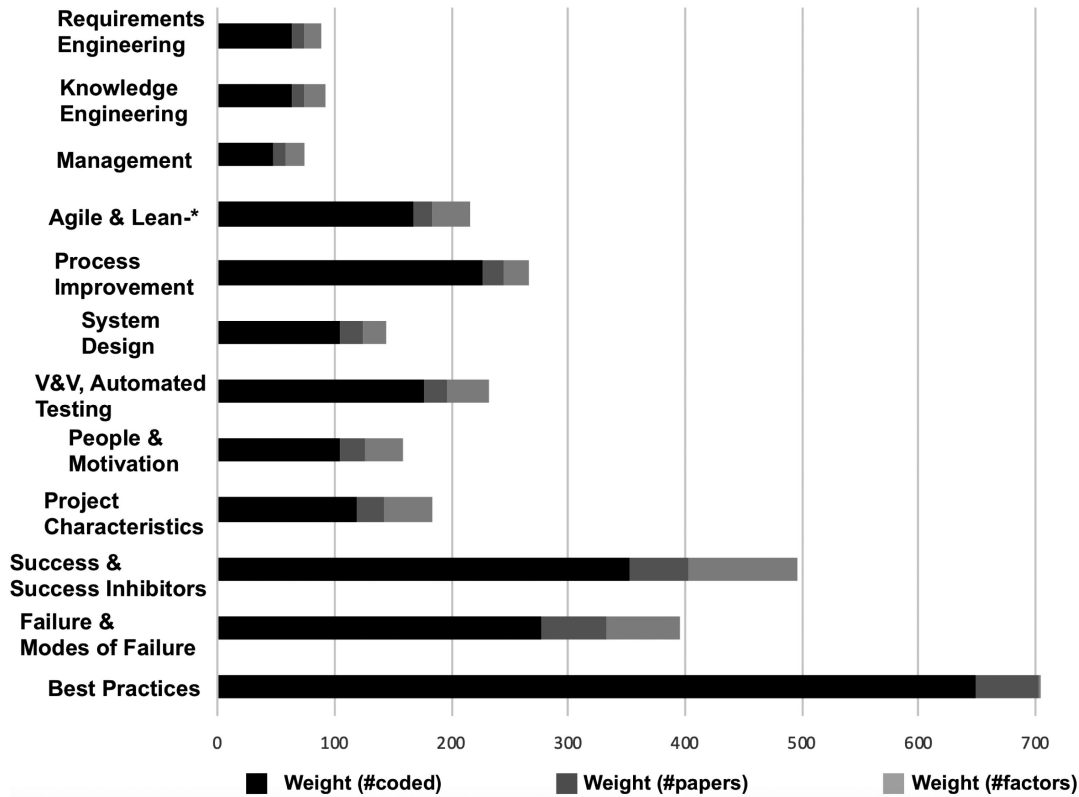


Fig. 3. Grounded theory content analysis.

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 preprocessing and analyses, we exploited the NetCulator bibliometric analytics tool<sup>9</sup> that supports LDA and a number of similar natural-language analyses and clustering techniques.

#### IV. RESULTS

##### 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 total, linked by 84 cooccurrence relations. However, an overview is available to browse as an online image.<sup>10</sup> 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 article and to address **RQ<sub>1</sub>**, we offer an outline of the core-concepts and their content analysis

[11]. Fig. 3 plots an overview of the most frequently occurring core-concepts captured with the method as described in Section III-C1. 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 nine 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, first, a gap in the levels of abstraction concerning both success and failure as evident from the state of the art and second, a relative distance in the depth of knowledge in the respective clusters found. Specifically, the clusters reflect the definitions outlined below:

- 1) *Requirements engineering*: Factors in this cluster address the creation, processing, resolution, traceability, or quality of requirements as well as any factors influencing

<sup>9</sup>[Online]. Available: <https://www.netculator.com/>

<sup>10</sup>[Online]. Available: <https://tinyurl.com/y79hfvby>



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 workflow awareness* as well as *human agile metrics*.
- 5) *Process improvement*: Factors in this cluster refer 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 and validation, automated testing*: Factors in this cluster refer 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 and 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 *dependence 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 the 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 subfactors: 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*.

## B. Success and Failure Distilled: Topic Modeling Results

The results of the topic modeling exercise are recapped in Figs. 4 and 5.

Fig. 4 outlines the raw results of the 14 themes emerging from topic modeling. Dots indicate most-probable word radices 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, Fig. 5 reports a manual representation of the themes emerging from the topic modeling 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 sections, 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 modeling, we offer the list of the most important terms as determined by the algorithm (arranged by decreasing rank with a cutoff below 20% probability) for each theme.

1) *Domain Area 1: People, Internal Software Characteristics, Best Practices*:

- 1) *Subversion*: The concept of subversion refers to concepts and challenges of subversive stakeholders previously

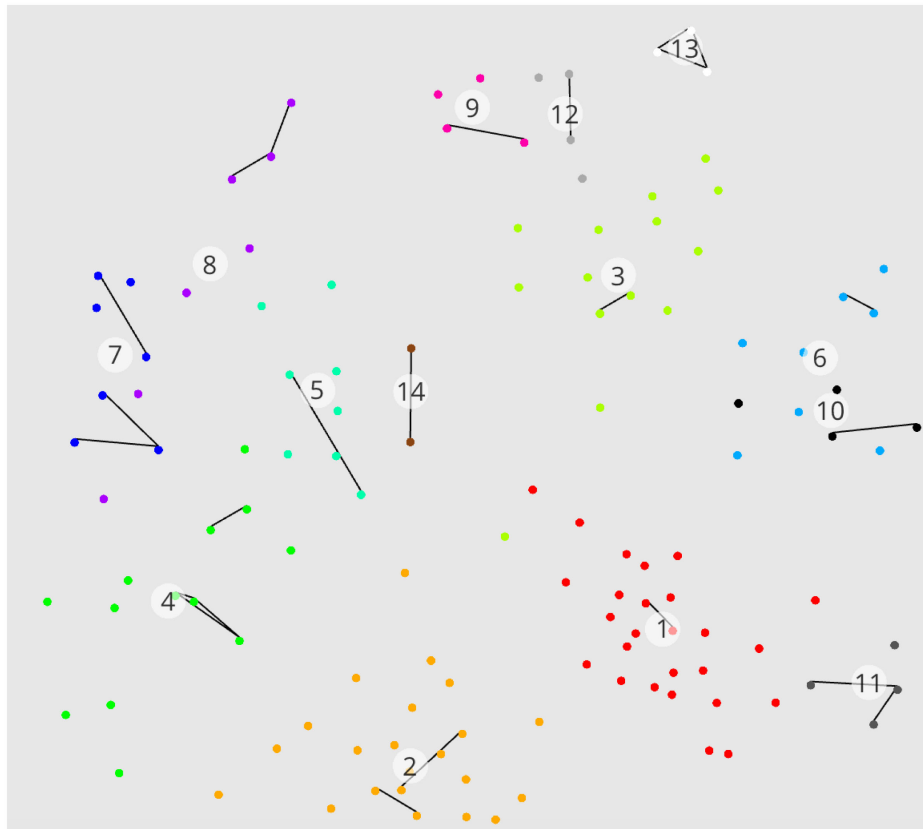


Fig. 4. Topic modeling clustering topology analysis [14] results.

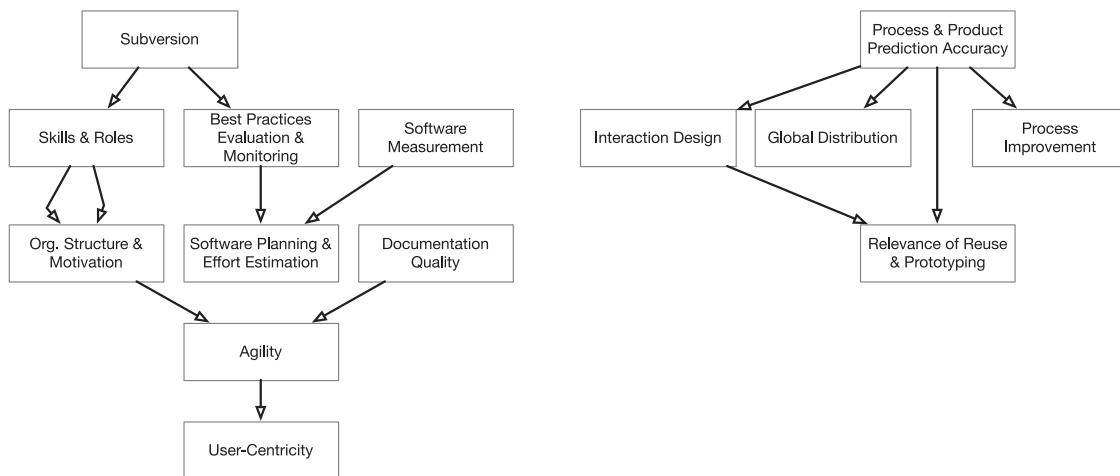


Fig. 5. Summary of topic modeling results.

introduced by Ross and Glass [15]. In the scope of our topic modeling exercise, this theme corresponds to most recurrent words and substrings (using wildcards) specified as follows: *friction\**, *restrict\**, *\*communication\**, *\*cooperation\**, *disregardof\**, and *lackof\**.

- 2) *Skills and 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-\**, and *connected\**.

- 3) *Best practices evaluation and 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 as follows: *defect-\**,

- accura\**, *change-\**, *\*prediction*, *\*accuracy*, *\*success*, and *\*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\**, and *\*interpretation*.
  - 5) *Organizational structure and 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\**, and *\*motivator\**.
  - 6) *Software planning and 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\**, and *governance*.
  - 7) *Documentation quality*: From the perspective of software maintenance and evolution, documentation is a discriminant in successful or failing software projects [32]–[34]. We obtained the following recurrent words and substrings for this theme: *\*knowledge\**, *domain\**, *requirements\**, *formal\**, *granularity*, *broker\**, and *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\**, and *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\**, and *context\**;
- 2) *Domain Area 2: Processes and External Product Characteristics*:
- 1) *Process and 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\**, and *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\**, and *operation\**.
  - 3) *Global distribution*: Global distribution in the scope of the themes emerging from our topic modeling refers to the general subfield 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\**, and *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 modeling, words and substrings relating to process improvement are: *progress\**, *train\**, *ad-hoc*, *capabilit\**, *principl\**, *chang\**, *need\**, *expectation\**, and *assess\**.
  - 5) *Reuse and Prototyping*: The last emerging theme out of topic modeling 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*, and *cost\**.

## V. USAGE, IMPLICATIONS, AND THREATS TO VALIDITY

### A. Discussion

Our results indicate that the phenomena of software success and failure is 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 generalizability of individual findings.<sup>11</sup>
- 3) *Identifying Empirically Validated and Actionable Antecedents*: 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 article 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 organizational and community structures [48]–[50] for software engineering.

## B. Addressing the Research Questions

This article 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?
- 3) What measurable quantities exist in themes that are not currently being measured?

In addressing these research questions we reported, in the scope of **RQ<sub>1</sub>**, the following:

**Answer to RQ<sub>1</sub>**. There exist more than 500 factors arranged in more than 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 clusters (and the factors therein) to devise tools and metrics for continuous monitoring and analysis.

Furthermore, in the scope of **RQ<sub>2</sub>**, 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 RQ<sub>2</sub>**. There exist 14 underlying themes among the more than 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 practitioners' efforts in avoiding failure and 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 modeling exercise reported in Fig. 3 to further understand and potentially measure the factors, focusing on operationalizing any factors that were not previously measured. At the same time, the topic modeling exercise we reported in Section IV-B 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.<sup>12</sup>

Finally, in the scope of **RQ<sub>3</sub>**, 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 (**RQ<sub>1</sub>** and **RQ<sub>2</sub>**), 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 following section outlines this contribution in more detail.

## C. Quality Model for Unobserved Software Quality Dimensions

To address the gap identified by **RQ<sub>3</sub>**, we operated a simple systematic search of every keyword discovered as part of topic modeling (see **RQ<sub>2</sub>**, Section IV) along with the additional search string defined as follows:

quality  $\wedge$  (model  $\vee$  framework  $\vee$  metric  $\vee$  measure  $\vee$  measurement  $\vee$  analysis  $\vee$  parameter).

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

<sup>11</sup>Nevertheless, to encourage replication, a comprehensive replication package is provided online here: <https://figshare.com/s/e6f0968e55c2cd024389>

<sup>12</sup>[Online]. Available: <https://dzone.com/articles/role-of-predictive-analytics-in-devops>



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 Network*, loosely defined by Meneely and Williams [54] as the superimposed communication and collaboration networks structures emerging during the 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 the 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.<sup>13</sup>

The emerging quality model features five categories of previously defined, validated metrics that can aid the observability of *subversion*, *organizational structures and motivation*, as well as *skills and 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., cooperated 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 **RQ<sub>3</sub>** we argue as follows:

*Summary for RQ<sub>3</sub>.* 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 five 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.

<sup>13</sup>For complete details, the reader may refer to [56], which contains a complete overview of all factors in the quality model, their operationalization, and their implementations in practice.

#### D. 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 article. Furthermore, these themes and clusters emerged *both* from topic modeling and 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 indicate that we know much more about success than we do about failure (e.g., see Table III). The number of codes applied for the core concepts of *success* and *failure* differ by almost 2 to 1 and the 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 postmortem analysis [63], empirical software failure research, fault lines [64], etc.). *In summary, further research along this line should be dedicated to better understand 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 three **RQs** 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 organizational 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.*

#### E. 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 doing so, we queried all major databases currently available in the 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 article, thus discarding nonrelevant 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 article 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 article. 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.

## VI. CONCLUSION

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

### A. 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 article 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<sup>14</sup> and is currently under experimentation.<sup>15</sup>

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 **RQs** 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 **RQ<sub>3</sub>**.

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.<sup>16</sup>
- 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.<sup>17</sup>
  - 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*:

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

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 preparedness 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 article. 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.

<sup>14</sup>[Online]. Available: <http://siemens.github.io/codeface/>

<sup>15</sup>The tool is available and free to use under the Apache 2 license agreement: <https://github.com/maelstromdat/CodeFace4Smells>

<sup>16</sup>[Online]. Available: <https://tinyurl.com/y79hfvby>

<sup>17</sup>This assumes that all factors have an equal mutual effect, which is obviously an open research question.

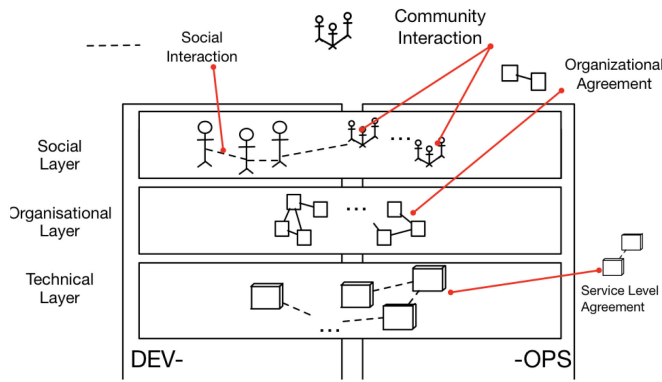


Fig. 6. Omniscient DevOps Analytics; concept tailored from [65].

### B. Synthesis and Future Work

This article 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 article, we have presented a more extensive and rigorous analysis of the literature, by executing three analyses aimed at deepening our understanding of software success and failure. The three 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 analyze 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 analyze the data in our replication bundle for the purpose of generalizing a more refined taxonomy or *ontology* for the purpose of instrumenting automated reasoning and risk analysis (e.g., to support postmortem 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 toward modeling 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 modeling computational intelligence while a holistic ALM suite back-end could be trained as an *ensemble* method to assemble the individual predictions toward 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 toward a DevOps analytics suite, which could be considered *omniscient*, that is, acting toward most if not all

of the dimensions accounted in the grounded-theory proposed in this article across all dimensions we highlighted, namely, the individuals dimension, their social-interactive communitarian dimension, the organizational layer combining them as well as the technical layer toward which their work is aimed.

### ACKNOWLEDGMENT

The authors would like to thank Dr. F. Castri for his invaluable effort and work in producing and allowing for the interrater reliability assessment of the grounded theory reported in this article. Furthermore, the authors would like to thank Dr. M. Di Penta, Dr. H. Muccini, Dr. P. Pelliccione, and Dr. M. Di Memmo for their precious insights, explanatory, and confirmatory discussions in the scope of addressing the threats to validity of this work. Finally, they would like to thank the associate editor and anonymous reviewers for the detailed and constructive comments on the preliminary version of this article, which were instrumental to improving the quality of our work.

- 1) Downloadable Dataset: [figshare.com/s/e6f0968e55c2cd024389](https://figshare.com/s/e6f0968e55c2cd024389)
- 2) Primary studies:<sup>18</sup> [www.dropbox.com/s/3lj4h0otpt7jrxe/SLRWorkingPortfolio.pdf?dl=0](https://www.dropbox.com/s/3lj4h0otpt7jrxe/SLRWorkingPortfolio.pdf?dl=0)
- 3) GT codes list and definition: [www.dropbox.com/s/ia0yv225r8jcggr/code%20list.xlsx?dl=0](https://www.dropbox.com/s/ia0yv225r8jcggr/code%20list.xlsx?dl=0)

### REFERENCES

- [1] P. Ralph and P. Kelly, "The dimensions of software engineering success," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 24–35. [Online]. Available: <http://dblp.uni-trier.de/db/conf/icse/icse2014.html#RalphK14>
- [2] T. O. Lehtinen, M. V. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius, "Perceived causes of software project failures—An analysis of their relationships," *Inf. Softw. Technol.*, vol. 56, no. 6, pp. 623–643, 2014.
- [3] M. V. Mäntylä, M. Jørgensen, P. Ralph, and H. Erdogmus, "Guest editorial for special section on success and failure in software engineering," *Empirical Softw. Eng.*, vol. 22, no. 5, pp. 2281–2297, 2017. [Online]. Available: <http://dblp.uni-trier.de/db/journals/ese/ese22.html#MantylaJRE17>
- [4] N. Iivari, *Enculturation of User Involvement in Software Development Organizations—An Interpretive Case Study in the Product Development Context*. Tampere, Finland: ACM Press, 2004, pp. 287–296.
- [5] ISO/IEC Standard, *Software Engineering—Product Quality—Part 1: Quality Model*, ISO/IEC, ISO Standard 9126-1, 2003.
- [6] M. Torchiano, F. Ricca, and A. Marchetto, "Is my project's truck factor low?: Theoretical and empirical considerations about the truck factor threshold," in *Proc. 2nd Int. Workshop Emerg. Trends Softw. Metric*, 2011, pp. 12–18. [Online]. Available: <http://dblp.uni-trier.de/db/conf/icse/wetsom2011.html#TorchianoRM11>
- [7] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2008, pp. 2–11. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1414008>
- [8] B. Kitchenham and S. Charters, "Guidelines for Performing systematic literature reviews in software engineering," School of Computer Science and Mathematics, Keele Univ., Keele, Newcastle, U.K., Jul. 2007.
- [9] J. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociol.*, vol. 13, no. 1, pp. 3–21, 1990.
- [10] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? (and how to fix it using search-based software engineering)," *CoRR*, 2016, *arXiv:1608.08176*. [Online]. Available: <http://dblp.uni-trier.de/db/journals/corr/corr1608.html#AgrawalFM16>
- [11] H.-F. Hsieh and S. E. Shannon, "Three approaches to qualitative content analysis," *Qualitative Health Res.*, vol. 15, no. 9, pp. 1277–1288, 2005.

<sup>18</sup>Please consider that all copies are copyrighted and were downloaded for research purposes only.



- [12] K. Schwaber, *Agile Project Management With Scrum*. Redmond, WA, USA: Microsoft Press, 2004. [Online]. Available: <http://my.safaribooksonline.com/9780735619937>
- [13] D. Kumar, "Lean software development," *The PROJECT PERFECT White Paper Collection*, 2005. [Online]. Available: [http://www.projectperfect.com.au/downloads/Info/info\\_lean\\_development.pdf](http://www.projectperfect.com.au/downloads/Info/info_lean_development.pdf)
- [14] R. Kwitt, S. Huber, M. Niethammer, W. Lin, and U. Bauer, "Statistical topological data analysis—A kernel perspective," in *Proc. Adv. Neural Inf. Process. Syst.*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. 2015, pp. 3070–3078. [Online]. Available: <http://dblp.uni-trier.de/db/conf/nips/nips2015.html#KwittHNLB15>
- [15] J. Rost and R. L. Glass, "The impact of subversive stakeholders on software projects," *Commun. ACM*, vol. 52, no. 7, pp. 135–138, 2009. [Online]. Available: <http://dblp.uni-trier.de/db/journals/cacm/cacm52.html#RostG09>
- [16] D. A. Tamburri, P. Lago, and H. v. Vliet, "Organizational social structures for software engineering," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 3:1–3:35, Jul. 2013. [Online]. Available: <http://doi.acm.org/10.1145/2522968.2522971>
- [17] N. Abbas, "Software quality and governance in agile software development." Ph.D. dissertation, Univ. Southampton, U.K., 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 *Proc. 17th Int. Conf. Eval. Assessment Softw. Eng.*, Porto de Galinh, Brazil, 2013, pp. 24–35. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2461003>
- [19] A. Soylu, P. D. Causmaecker, D. Preuveneers, Y. Berbers, and P. Desmet, "Formal modelling, knowledge representation and reasoning for design and development of user-centric pervasive software: A meta-review," *Int. J. Metadata Semantics Ontol.*, vol. 6, no. 2, pp. 96–125, 2011. [Online]. Available: [http://www.ahmetsoylu.com/wp-content/uploads/2011/05/IJMSO\\_62\\_Soylu\\_et\\_al.pdf](http://www.ahmetsoylu.com/wp-content/uploads/2011/05/IJMSO_62_Soylu_et_al.pdf)
- [20] N. E. Fenton, "Software measurement: A necessary scientific basis," *IEEE Trans. Softw. Eng.*, vol. 20, no. 3, pp. 199–206, Mar. 1994. [Online]. Available: <http://dblp.uni-trier.de/db/journals/tse/tse20.html#Fenton94>
- [21] M. Nasir, "A survey of software estimation techniques and project planning practices," in *Proc. 7th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw., Parallel/Distrib. Comput.*, Jun. 2006, pp. 305–310.
- [22] D. A. Tamburri, R. De Boer, E. Di Nitto, P. Lago, and H. van Vliet, "Dynamic networked organizations for software engineering," in *Proc. Int. Workshop Social Softw. Eng.*, 2013, pp. 5–12.
- [23] H. Ogasawara, T. Kusanagi, and M. Aizawa, "Proposal and practice of software process improvement framework—Toshiba's software process improvement history since 2000," *J. Softw., Evol. Process*, vol. 26, no. 5, pp. 521–529, 2014. [Online]. Available: <http://dblp.uni-trier.de/db/journals/smr/smr26.html#OgasawaraKA14>
- [24] S. Assar, M. Borg, and D. Pfahl, "Using text clustering to predict defect resolution time: A conceptual replication and an evaluation of prediction accuracy," *Empirical Softw. Eng.*, vol. 21, no. 4, pp. 1437–1475, 2016. [Online]. Available: <http://dblp.uni-trier.de/db/journals/ese/ese21.html#AssarBP16>
- [25] C. Ebert, M. Kuhrmann, and R. Prikladnicki, "Global software engineering: An industry perspective," *IEEE Softw.*, vol. 33, no. 1, pp. 105–108, Jan./Feb. 2016. [Online]. Available: <http://dblp.uni-trier.de/db/journals/software/software33.html#EbertKP16>
- [26] M. Bellingham, S. Holland, and P. Mulholland, "A cognitive dimensions analysis of interaction design for algorithmic composition software," in *Proc. Psychol. Program. Interest Group Annu. Conf.*, B. du Boulay and J. Good, Eds. Psychology of Programming Interest Group, 2014, p. 18. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ppig/ppig2014.html#BellinghamHM14>
- [27] S. H. Rubin, "Knowledge-based software prototyping and reuse," *Int. J. Artif. Intell. Tools*, vol. 6, no. 1, pp. 127–147, 1997. [Online]. Available: <http://dblp.uni-trier.de/db/journals/ijait/ijait6.html#Rubin97>
- [28] R. Pham, "Improving the software testing skills of novices during onboarding through social transparency," Ph.D. dissertation, University of Hanover, Hanover, Germany, 2016.
- [29] H.-D. Yang, H.-R. Kang, and R. M. Mason, "An exploratory study on meta skills in software development teams: Antecedent cooperation skills and personality for shared mental models," *Eur. J. Inf. Syst.*, vol. 17, no. 1, pp. 47–61, 2008. [Online]. Available: <http://dblp.uni-trier.de/db/journals/ejis/ejis17.html#YangKM08>
- [30] C. Ebert and R. Dumke, *Best Practices in Software Measurement Establish - Extract - Evaluate - Execute*. Berlin, Germany: Springer-Verlag, 2007. [Online]. Available: <http://dx.doi.org/10.1007/978-3-540-71649-5>
- [31] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.
- [32] A. Boyd, "A goal-based approach to the evaluation and documentation of business process re-engineering," *Aslib Proc.*, vol. 56, no. 5, pp. 286–300, 2004. [Online]. Available: <http://dblp.uni-trier.de/db/journals/aslib/aslib56.html#Boyd04b>
- [33] V. Borja, J. Harding, and K. Toh, "Product re-engineering process using an enterprise modelling architecture," *Int. J. Agile Manag. Syst.*, vol. 2, no. 3, pp. 214–225, 2000.
- [34] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proc. 23rd Annu. Int. Conf. Des. Commun.*, 2005, pp. 68–75. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1085313.1085331&coll=Portal&dl=ACM&CFID=1901909&CFTOKEN=99893798>
- [35] A. Elssamadisy and D. West, "Adopting agile practices: An incipient pattern language," in *Proc. Conf. Pattern Lang. Programs*, 2006, pp. 1:1–1:9. [Online]. Available: <http://doi.acm.org/10.1145/1415472.1415474>
- [36] A. Rohunen, P. Rodriguez, P. Kuvaja, L. Krzanik, J. Markkula, and B. Turhan, "EnglishAgile adoption strategies in the context of agile in the large: Flexi agile adoption industrial inventory," in *EnglishAgile Processes in Software Engineering and Extreme Programming*, ser. Lecture Notes in Business Information Processing, A. Sillitti, A. Martin, X. Wang, and E. Whitworth, Eds. Berlin, Germany: Springer, 2010, vol. 48, pp. 397–398. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-13054-0\\_50](http://dx.doi.org/10.1007/978-3-642-13054-0_50)
- [37] J. Rubin, *Handbook of Usability Testing. How to Plan, Design, and Conduct Effective Tests*. New York, NY, USA: Wiley, 1994.
- [38] M. Jörgensen, "Experience with the accuracy of software maintenance task effort prediction models," *IEEE Trans. Softw. Eng.*, vol. 21, no. 8, pp. 674–681, Aug. 1995.
- [39] L.-W. Chen and S.-J. Huang, "Accuracy and efficiency comparisons of single- and multi-cycled software classification models," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 173–181, 2009. [Online]. Available: <http://dblp.uni-trier.de/db/journals/infsof/infsof51.html#ChenH09>
- [40] A. Cooper and R. Reimann, *About Face 3.0: The Essentials of Interaction Design*, 3rd ed. Hoboken, NJ, USA: Wiley, 2007. [Online]. Available: <http://www.amazon.de/About-Face-3-0-Essentials-Interaction/dp/0470084111%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0470084111>
- [41] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *Proc. Future Softw. Eng. Conf.*, 2007, pp. 188–198. [Online]. Available: <http://dblp.uni-trier.de/db/conf/icse/fose2007.html#Herbsleb07>
- [42] D. A. Tamburri, E. di Nitto, P. Lago, and H. van Vliet, "On the nature of the GSE organizational social structure: An empirical study," *Proc. 7th IEEE Int. Conf. Global Softw. Eng.*, 2012, pp. 1–10.
- [43] R. B. Grady, *Successful Software Process Improvement*, 1st ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1997.
- [44] M. Niazi, D. Wilson, and D. Zowghi, "Critical success factors for software process improvement implementation: An empirical study," *Softw. Process, Improvement Pract.*, vol. 11, no. 2, pp. 193–211, 2006.
- [45] W. Frakes, "Systematic software reuse: A paradigm shift," in *Proc. Int. Conf. Softw. Reuse: Adv. Softw. Reusability*, 1994, pp. 2–3.
- [46] M. Thompson and N. Wishbow, "Prototyping: Tools and techniques: improving software and documentation quality through rapid prototyping," in *Proc. 10th Annu. Int. Conf. Syst. Documentation*, R. MacLean, Ed., 1992, pp. 191–199. [Online]. Available: <http://dblp.uni-trier.de/db/conf/sigdoc/sigdoc1992.html#ThompsonW92>
- [47] J. Keyes, *Social Software Engineering*. Boca Raton, FL, USA: Taylor & Francis, Auerbach Series, 2011.
- [48] F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana, and R. Oliveto, "How do community smells influence code smells?" in *Proc. 40th Int. Conf. Softw. Eng., Companion*, 2018, pp. 240–241. [Online]. Available: <http://doi.acm.org/10.1145/3183440.3194950>
- [49] D. A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman, "Discovering community patterns in open-source: A systematic approach and its evaluation," *Empirical Softw. Eng.*, vol. 24, pp. 1369–1417, 2018.
- [50] D. A. Tamburri, R. Kazman, and H. Fahimi, "The architect's role in community shepherding," *IEEE Softw.*, vol. 33, no. 6, pp. 70–79, 2016. [Online]. Available: <http://dblp.uni-trier.de/db/journals/software/software33.html#TamburriKF16>



- [51] J. Fulcher and L. C. Jain, Eds. *Computational Intelligence: A Compendium*, ser. Studies in Computational Intelligence. New York, NY, USA: Springer, 2008, vol. 115. [Online]. Available: <http://dblp.uni-trier.de/db/series/sci/sci115.html>
- [52] International Standard Organization (ISO), *International Standard ISO/IEC 9126, Information Technology Product Quality Part 1: Quality Model*, 2001.
- [53] R. G. Dromey, "A model for software product quality," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 146–162, Feb. 1995. [Online]. Available: <http://dblp.uni-trier.de/db/journals/tse/tse21.html#Dromey95>
- [54] A. Meneely and L. Williams, "Socio-technical developer networks: should we trust our measurements?" in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 281–290. [Online]. Available: <http://doi.acm.org/10.1145/1985793.1985832>
- [55] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer social networks predict failures?" in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2008, pp. 2–12.
- [56] S. Magnoni, D. A. Tamburri, and E. D. Nitto, "A socio-technical quality model for software engineering organisational structures: An empirical study," Politecnico di Milano, Master Thesis Series, <https://goo.gl/Y9R4KY>, 2016.
- [57] C. Amrit and J. van Hillegersberg, "Mapping social network to software architecture to detect structure clashes in agile software development," in *Proc. 15th Eur. Conf. Inf. Syst.*, 2007, pp. 334–345. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ecis/ecis2007.html#AmritH07>
- [58] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2008, pp. 2–11. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1414008>
- [59] K. Crowston, K. Wei, Q. Li, and J. Howison, "Core and periphery in free/libre and open source software team communications," in *Proc. 39th Annu. Hawaii Int. Conf. Syst. Sci.*, 2006, Art. no. 118a. [Online]. Available: <http://dblp.uni-trier.de/db/conf/hicss/hicss2006-6.html#CrowstonWLH06>
- [60] C. Amrit and J. van Hillegersberg, "Exploring the impact of socio-technical core-periphery structures in open source software development," *J. Inf. Technol.*, vol. 25, no. 2, pp. 216–229, 2010. [Online]. Available: <http://dblp.uni-trier.de/db/journals/jitech/jitech25.html#AmritH10>
- [61] M. Y. Allaho and W.-C. Lee, "Analyzing the social networks of contributors in open source software community," in *Proc. Appl. Social Media Social Netw. Anal.*, 2015, pp. 57–75. [Online]. Available: <http://dblp.uni-trier.de/db/series/lnsn/kc2015.html#AllahoL15>
- [62] C. Couto, C. A. Maffort, R. Garcia, and M. T. Valente, "Comets: A dataset for empirical research on software evolution using source code metrics and time series analysis," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 1, pp. 1–3, 2013. [Online]. Available: <http://dblp.uni-trier.de/db/journals/sigsoft/sigsoft38.html#CoutoMGV13>
- [63] J. A. Hager, "Software cost reduction methods in practice: A post-mortem analysis," *J. Syst. Softw.*, vol. 14, no. 2, pp. 67–77, 1991. [Online]. Available: <http://dblp.uni-trier.de/db/journals/jss/jss14.html#Hager91>
- [64] K. Bahmani, Z. Semnani-Azad, K. P. Sycara, and M. Lewis, "Team faultline measures: The effect of rescaling weights," in *Proc. 51st Hawaii Int. Conf. Syst. Sci.*, 2018. [Online]. Available: <http://dblp.uni-trier.de/db/conf/hicss/hicss2018.html#BahmaniSS018>
- [65] D. A. Tamburri, D. Di Nucci, L. Di Giacomo, and F. Palomba, "Omniscient devops analytics," in *Proc. Int. Symp. Softw. Eng. Aspects Continuous Develop. New Paradigms Softw. Prod. Deployment*, 2018.
- [66] J. Ereth, "Dataops—Towards a definition," in *Proc. Conf. "Lernen, Wissen, Daten, Analysen,"* R. Gemulla, S. P. Ponzetto, C. Bizer, M. Keuper, and H. Stuckenschmidt, Eds., vol. 2191, CEUR-WS.org, 2018, pp. 104–112. [Online]. Available: <http://dblp.uni-trier.de/db/conf/lwa/lwda2018.html#Ereth18>
- [67] J. Klespitz, M. Bir, and L. Kovcs, "Cross-tool interoperability in heterogeneous application lifecycle management systems," in *SoMeT*, ser. Frontiers in Artificial Intelligence and Applications, H. Fujita and G. A. Papadopoulos, Eds., vol. 286, Amsterdam, The Netherlands: IOS Press, 2016, pp. 213–220. [Online]. Available: <http://dblp.uni-trier.de/db/conf/somet/somet2016.html#KlespitzBK16>