Empir Software Eng (2017) 22:2281–2297
DOI 10.1007/s10664-017-9505-5

EDITORIAL

Guest editorial for special section on success and failure in software engineering

Mika V. Mäntylä[1] · **Magne Jørgensen**[2] · **Paul Ralph**[3] · **Hakan Erdogmus**[4]

Published online: 26 April 2017

Abstract Many papers investigate success and failure of software projects from diverse perspectives, leading to a myriad of antecedents, causes, correlates, factors and predictors of success and failure. This body of research has not yet produced a solid, empirically grounded body of evidence enabling actionable practices for increasing success and avoiding failure in software projects. The need for more evidence motivates this special issue, which includes four articles that contribute to our understanding of how software project success and failure relate to topics such as: requirements engineering, user satisfaction, start-up pivots and retrospective discussions. We moreover present a brief systematic review to both situate the accepted articles in existing literature and to explore enduring methodological and conceptual challenges in this area, including developing sound instruments for measuring success, representative sampling without population lists and creating both empirically sound and practically actionable taxonomies of success antecedents.

Keywords Success · Failure · Success factors · Failure factors · Software engineering · Project management · Systematic review

✉ Hakan Erdogmus
hakan.erdogmus@sv.cmu.edu

Mika V. Mäntylä
mika.mantyla@oulu.fi

Magne Jørgensen
magnej@simula.no

Paul Ralph
p.ralph@auckland.ac.nz

[1] M3S / ITEE / University of Oulu, Oulu, Finland

[2] Simula Research Laboratory, Fornebu, Norway

[3] University of Auckland, Auckland, New Zealand

[4] Carnegie Mellon University, Pittsburgh, PA 15213, USA

1 Introduction

In software projects, successes (Ralph and Paul 2014) and failures (Lehtinen et al. 2014) come not only in degrees but also in diverse shapes and forms. If we agree that software project success and failure is, if not a false dichotomy, at least a multidimensional spectrum, we can explore different types of successes and failures. Despite ongoing concerns over the failure rate of software projects, we still have not answered basic questions including "*How do we measure success?*" and "*How do we reduce failures?*" These questions motivate the organization of this special issue on success and failure in software engineering.

The topics of the special issue are predominately concerned with the performance of software engineering projects and processes (e.g., dev-ops, maintenance). While of course important, the performance of software products (e.g., market share), tools (e.g.,whether a certain UX modelling tool improves design quality) and related concerns (e.g., software engineering education, ethics) are beyond the present scope.

That said, at least three critical elements for understanding success and failure are evident:

1. *Operationalization,* i.e. definitions and measures of success and failure may vary among projects. Success measures may include measures of cost control, quality of deliverables, schedule control, productivity, profitability, client benefits, and user satisfaction. The relative emphasis on different measures and the threshold for deeming a project successful or failed may also vary among projects.
2. *Actionable Antecedents,* i.e. practices contributing to success or failure undertook by participants during a project that affect the project's outcome. Examples include processes, training, stakeholder involvement, tool support, and communication strategies.
3. *Non-Actionable Antecedents,* i.e. context factors, which are beyond the control of project actors, may enable or constrain outcomes and help predict success or failure. Examples include project environment, budget, client involvement, and development team skill level. Sometimes context factors and practices are linked. For example, the practice of training developers might mitigate the context factor of inexperienced developers.

With these elements in mind, the purposes of this introduction are to (1) provide an overview of existing literature on software project success and failure and (2) to introduce the articles in the special issue. We proceed in this order, and conclude with some thoughts on future research.

2 Review of Literature

We did not receive any submissions to the special issue that provided a systematic literature review (SLR) regarding software project success and failure. Partly for this reason, we include a limited, brief SLR of our own. We searched for relevant research and analyzed the results using text mining and bibliometrics. We then subjected a subset of the most influential (i.e., highly-cited) papers to manual qualitative analysis. Below, we provide the results of the quantitative and then the qualitative analysis, followed by a short discussion.

2.1 Literature Search

We used Scopus to search for literature on success and failure in software engineering. Scopus is marketed as "the largest abstract and citation database of peer-reviewed literature: scientific journals, books and conference proceedings."[1]

When iteratively refining the literature searches, we discovered that achieving high precision and recall was surprisingly difficult. Here, we use the terms precision and recall as they are defined in information retrieval; i.e.,precision gives the proportion of relevant search results, while recall gives the coverage of all relevant search results. Many articles that had very little to do with either success or failure in software engineering in general used the words "success" and "failure" simply to motivate their work. As a counter-measure, we required that the word "success" or "failure" appear in the paper title. We realize that this, among other search decisions, sacrifice recall for precision.

We also discovered that many papers having nothing to do with industrial software engineering (e.g., a paper describing the development of research software) use "software engineering" as a keyword. We therefore targeted the search term "software engineering" only in paper abstracts and titles. Additionally, we included common synonyms to capture software engineering work conducted within related projects. This resulted in 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")

For papers about failure, we received many irrelevant hits that focused on failure engineering and reliability modelling, which consider software engineering failures only as part of software that is malfunctioning. As these are inconsistent with our focus on projects and processes, we removed them using additional exclusion rules. We similarly used exclusion rules to remove papers concerning failures in physical and biological systems. For failure papers only, we therefore excluded the papers matching the following search string:

TITLE-ABS-KEY ("heart failure" OR "failure mode" OR "stability failure" OR "failure mode" OR "ceramic" OR "steel" OR "mtbe" OR "coal" OR "fault diagnosis" OR "System failure engineering" OR "Failure simulation" OR "slope failure" OR "fault tolerance" OR "physiology" OR "reliability modeling" OR "software reliability")

This process resulted in 159 papers on software project failures and 434 papers on software project successes.

2.2 Manual Filtering

Next, we manually filtered all papers based on their titles and abstracts. We removed papers where: (i) the papers used the term "failure" to denote software faults (bugs); (ii) the papers introduced a method or tool they claimed would reduce the risk of project failure or increasing the probability of success, without really testing that this was the case; (iii) the papers described the failure or success of introducing, using or improving a method, tool or process, but did not relate this to project success or failure; (iv) the paper was about other types of projects than software projects, or just one phase of a software project; or (v) the paper was about project success and failure, but provided no research results.

[1] https://www.elsevier.com/solutions/scopus

Manual filtering resulted in 90 papers on failure and 260 papers on success, with some overlap. Removing duplicates and nearly identical works (identified with Jaccard similarity) left a total of 332 papers. This 44% reduction in papers demonstrates that textual search without manual filtering would probably have been misleading.

2.3 Analyzing Literature With Text Mining

To get a high-level overview of the literature, we generated word clouds using the R-package "wordcloud" (Fellows 2012). We then used a topic modelling technique called Latent Dirichlet Allocation (LDA) to cluster our papers. Clustering offers a more detailed view of the papers than word clouds. For the clustering, we performed standard text-mining pre-processing that improves results by removing unnecessary information. Specifically, we removed standard copyright information occasionally present in the abstract, removed punctuation and numbers, converted all the letters to lower case, removed common stop words for English, stemmed the document, created a document-term-matrix, and removed terms with scores below the median term frequency minus inverse document frequency (tf–idf) from our vocabulary.

For topic modelling, several measures of clustering quality have been proposed. Following the approach recommended by Griffiths and Steyvers (2004) for scientific corpuses, we selected log-likelihood as our measure of clustering quality. We used a fixed number of clusters ($k = 10$) to offer an easily understandable overview of the area and allow a sample of papers for the qualitative analysis to remain small, rather than generating large number of fine-grained clusters. We used the genetic algorithm Differential Evolution to tune LDA hyper-parameters alpha and beta as suggested by Agrawal et al. (2016).

2.4 Analyzing Literature With Qualitative Coding

From each of the ten clusters, we selected the top-cited paper, normalized for time. We used normalization for time so that both older and newer papers had equal chances of being selected for further analysis. This generated a sample of influential papers for more in-depth qualitative analysis to investigate what the papers are really saying about their topics. We manually analyzed these ten papers using an *a priori* defined coding scheme.

We created the coding using a three-step process: (1) collective brainstorming of points of inquiry; (2) synthesizing these ideas into a set of questions; and (3) review of the questions and suggested improvements. This process produced the coding scheme shown in the Appendix. Although we expected the coding scheme to evolve throughout the analysis, no major changes were made. However, we updated the wording and some of the examples in the Appendix for clarity.

The ten-paper subset was analyzed against the coding scheme by one of the authors (Ralph) using an Excel spreadsheet. Two other authors reviewed this analysis. No particularly controversial analytical leaps were evident, except that it was difficult to specify a single core message or thesis for some papers.

2.5 Results: Bibliometrics

We performed a bibliometric analysis on our pool of 322 articles. Figure 1 shows the number of papers published in each year. We can see that, prior to 2005, fewer than eight papers were

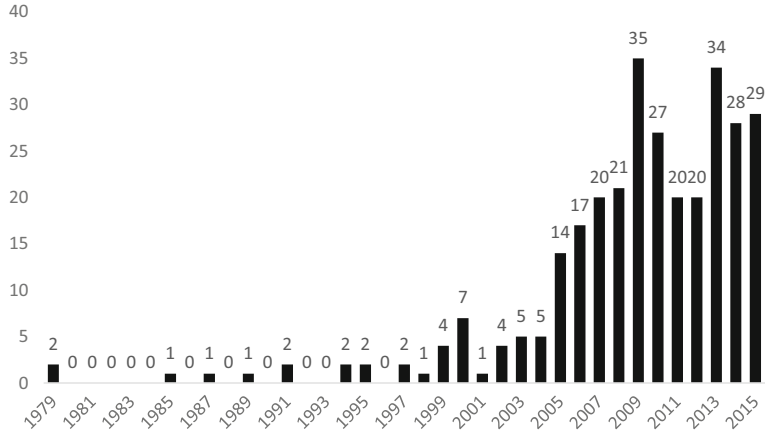

Fig. 1 Number of papers published per year

published per year, while between 2005 and 2015, an average of 24 papers were published per year. This trend is similar to that observed in a bibliometric study (Garousi and Mäntylä 2016) of the entire software engineering literature, which analyzed over 70,000 papers also using data from Scopus. In that study, prior to 2005, less than 2,000 papers had been published annually and after 2005, the annual number of paper was over 3,000. Overall, the general growth trends are similar in both works.

Table 1 lists the most popular publication outlets in our sample. It shows research coming from three research communities: software engineering, information systems, and project management, as well as venues we categorized as General Computer Science (CS). Although project management research has only one venue, *International Journal of Project Management*, it was the second most popular with ten papers. Moreover, the sources classed as General CS have large volumes of papers annually; for example, *Lecture Notes in Computer Science* published over 20,000 papers in 2015.

Table 1 Top-10 venues for success and failure in software engineering

Rank	Venue	Community	Count
1	Information and Software Technology	Software Engineering	11
2	International Journal of Project Management	Project Management	10
3	Journal of Systems and Software	Software Engineering	9
4	Americas Conference on Information Systems, AMCIS	Information Systems	8
4	International Conference on Information Systems, ICIS	Information Systems	8
6	Annual Hawaii International Conference on System Sciences, HICSS	Information Systems	7
7	IEEE Software	Software Engineering	6
7	Lecture Notes in Computer Science	General CS	6
9	ACM International Conference Proceeding Series	General CS	5
9	Advances in Intelligent Systems and Computing	General CS	5
9	Information Systems Management	Information Systems	5
9	International Conference on Software Engineering, ICSE	Software Engineering	5
9	Lecture Notes in Business Information Processing	General CS	5

2.6 Results: Research Topics

The top-level analysis of the research topics covered in the identified papers are givenby the world cloud in Fig. 2. We can see that a greater number of papers focused on success than on failure. The word cloud also reveals that the articles mostly considered success and failure as a software project management topic. Smaller terms reveal further information regarding the context, such as open source, agile, offshore, outsourcing, business, organizational, requirements, and teams.

We clustered the papers into ten topics to have a more detailed view of various success and failure factors. Ten topics offer a high-level view of the area and facilitate sampling for qualitative analysis. According to LDA, a topic cluster is represented by the most probable words for the topic. Table 2 shows the ten most probable words for the ten topics (clusters) we generated.

Computer-based clustering of natural language documents is hardly ever perfect, but it can still guide human interpretation in large text corpuses. Roughly speaking, we can see two dimensions in the topic words. We can see context variables under which the studies have been performed: enterprise resource planning (erp; topic 1), agile (topic 3), global software development (gsd; topic 4), egovernance (topic 6), student (topic 7), open source software (oss, topic 8), outsource and offshore (topic 9), and health and healthcare (topic 10). We can also see more detailed research topics, which may be important factors in SE successes and failures: collaboration, bid (topic 1); leadership, srs (software requirements specification, topic 2); deliveries, iterations (topic 3); decisions, risk, stress, motivation (topic 4); scope, coordination, maturity, strategy (topic 5); user, feature, task (topic 6); early prediction and estimation (topic 7); interaction (topic 8); goverance, contract (topic 9); and knowledge, skill, learning (topic 10).

Comparison between Table 2 and Fig. 2 shows similarities: for example, agile, outsourcing, organizational, oss, and csfs (critical success factors) can be seen in both. Table 2, which was created with LDA topic modelling, provides more information than Fig. 2. An additional benefit of LDA topic modelling is that each paper can be assigned to one cluster/topic. We take

Fig. 2 Word cloud based on the titles and abstracts of success and failure papers

Table 2 The ten most probable words (stems) for each topic

Term Rank	T1: ERP + Collaboration	T2: Success/Failure attributes	T3: Agile + Criteria	T4: Risk + GSD	T5: Organization	T6: User	T7: Prediction	T8: Open-source	T9: Outsourcing	T10: Competences + Healthcare
1	Erp	Attribut	Agil	Risk	Organis	User	Predict	Oss	Outsource	Knowledg
2	Describ	Profession	Criteria	Strategi	Scope	Similar	Estim	Sourc	Offshore	Skill
3	Trend	Leadership	Decis	Gsd	Coordin	Featur	Earli	Open	Csfs	Learn
4	Variabl	Srs	Deliv	Probabl	Public	Task	Stage	Network	Vendor	Health
5	Collabor	Poor	Correl	Optim	Start	Tempor	Classifi	Social	Govern	Isit
6	Enterpris	Person	Iter	Stress	Matur	Isd	Metric	Interest	Appropri	Domain
7	Capabl	Util	Deliveri	Mitig	Extern	Viewpoint	Standard	Interact	Contract	Uncertainti
8	Bid	Overrun	Core	Motiv	Congruenc	Believ	Student	Download	Infrastructur	Healthcar
9	Teamwork	Situat	Explain	Proposit	Strateg	Causal	Mine	Pattern	Oosd	Innov
10	Durat	Element	Given	Sdo	Deploy	Egovern	Accur	Sustain	Fit	Illustr

advantage of this benefit in the following section to investigate the papers by examining a prominent representative from each cluster.

Table 3 lists the most-cited paper for each topic.

2.7 Results: Qualitative Analysis

As described above, we analyzed the top cited paper of each text clustering topic using the coding scheme in the Appendix to achieve a diverse sample while highlighting the most influential work in the area. The sample of ten papers comprise four questionnaire-based studies, two case studies, a social network analysis, a systematic literature review (SLR), an experience report, and a workshop report. Eight papers are empirical (we consider SLRs empirical). Six papers investigate software development in general, while four papers focus on a specific context or domain, namely, offshoring, open source, health IT and ERP implementation. Our analysis of the sampled papers revealed three main research challenges, as well as a couple of common shortcomings and some positive insights.

2.7.1 Challenge 1: Creating and Validating Instrument for Measuring Success

Software project success (or failure) is multidimensional in the sense that projects can succeed or fail in different ways, from different perspectives. Trying to measure success using one- or two-dimensional spectra therefore oversimplify and overrationalize reality.

Table 3 Top cited paper (citations per year) for each topic

Topic	Paper	Citations	Citation per year
T1: ERP + Collaboration	Managing ERP implementation failure: A project management perspective (Chen et al. 2009)	75	10.71
T2: Success/ Failure attributes	Software developer perceptions about software project failure: A case study (Linberg 1999)	146	8.59
T3: Agile + Criteria	Project management: Cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria (Atkinson 1999)	515	30.29
T4: Risk + GSD	Benefits Realisation Management and its influence on project success and on the execution of business strategies (Serra and Kunc 2015)	20	20
T5: Organization	Defining 'success' for software projects: An exploratory revelation (Agarwal and Rathod 2006)	108	10.80
T6: User	A systematic review on the relationship between user involvement and system success (Bano and Zowghi 2015)	16	16.00
T7: Prediction	Early warning signs of IT project failure: The dominant dozen (Kappelman, McKeeman, and Zhang 2006)	129	12.90
T8: Open-source	Location, location, location: How network embeddedness affects project success in open source systems (Grewal, Lilien, and Mallapragada 2006)	216	21.60
T9: Outsourcing	Empirical investigation of success factors for offshore software development outsourcing vendors (Khan, Niazi, and Ahmad 2012)	21	5.25
T10: Competences + Healthcare	Health IT Success and Failure: Recommendations from Literature and an AMIA Workshop (Kaplan and Harris-Salamone 2009)	157	22.43

Furthermore, success is a construct —a hypothesized variable that cannot be measured directly. Success is real in the same way trust and preference are real, but not in the same way length and mass are real. We must therefore operationalize it using indicators which vary in their construct validity—"the degree to which inferences can legitimately be made from the operationalizations in [a] study to the theoretical constructs on which those operationalizations were based" (Trochim 2001).

In one of the analyzed papers, Kaplan et al. emphasize that "there is little agreement about what *success* or *failure* is" (2009, p. 294); in another, Agarwal and Rathod claim that "in practice, it may be very difficult to claim that the project was really successful or not" (2006, p. 358). None of the analyzed papers use or present a validated instrument for measuring success or failure. Ironically, Kappelman et al. (2006) point out that *lack of documented success criteria* is one of the dominant warning signs preceding project failure.

Of the papers that measure success, four use participant's subjective judgements. Both the type of participant (e.g. developer, manager, user) and the dimensions on which judgements were sought vary. Serra and Kunc (2015) have the most detailed evaluation of success. They used a questionnaire with Likert scales covering budget, schedule, scope, business goals, undesired outcomes and return on investment. While the subjective judgments of informants may be valuable for assessing success, they are intrinsically from the point of view of the informant. For example, the developers' judgments of user satisfaction are unlikely to be reliable. Meanwhile, Grewal et al. (2006) operationalize success using page-view and download frequency; that is, they use popularity as a surrogate for success. While this is reasonable given their open source focus, popularity is still only a small part of success.

In summary, the meaning of success is contested and not agreed upon. No well-validated measures of success are available (as far as we know); consequently, researchers either use fairly simple, non-validated measures or rely on the subjective judgment of participants. Research rigor will continue to be hampered until better instruments are created. The lack of agreement around and complexity of what we should mean with software project success entail serious challenges for creating such measures.

2.7.2 Challenge 2: Representative Sampling Without Population Lists

Without effective sampling, even the most extensive study on software projects can give misleading results. An infamous example of this is the Standish Group's 1994 survey (The Standish Group International Inc 1994) claiming a software crisis, due their finding that only 16.9% of all software projects were successful, measured as within budget, on schedule, and with specified functionality. This survey may be the most frequently cited survey on software project success. The sampling process is not well described, but (page 13) of their report states: "*We then called and mailed a number of confidential surveys to a random sample of top IT executives, asking them to share failure stories. During September and October of that year, we collected the majority of the 365 surveys we needed to publish the CHAOS research.*" While we can learn a lot from failure stories, they should clearly not be presented as representing the software industry.

In our evaluation, we found eight papers with a sampling strategy. Of these eight papers, seven employ convenience or purposive sampling. While convenience and purpose sampling are appropriate or even preferable for some studies, lack of representative sampling in a body of literature is of concern. A body of (social and applied) research typically combines purposive sampling for theory-building case studies, convenience sampling for exploratory

studies, and random sampling for confirmatory studies. We come to a deep and generalizable understanding of a phenomenon by synthesizing results from these different strategies.

Software engineering, however, often lacks comprehensive population lists for the phenomena of interest. For example, we cannot randomly sample Australian mobile video game development projects because there is no comprehensive list of such projects to sample from. This leads to two methodological challenges.

First, many studies sample from surrogate populations that may differ from target populations in unknown ways. Grewal et al. (2006) employed a multi-stage sampling strategy: they purposively chose sourceforge, then randomly selected a subset of sourceforge projects that met specific conditions, and then used all the developers involved in that foundry. In the context of their study, this is a perfectly reasonable approach. There is no comprehensive list of open source projects, so they sample from a surrogate population. However, we do not understand the differences between the surrogate population and the population to which we want to generalize; e.g. a random sample of sourceforge projects with more than 100 developers may or may not be representative of large open source projects. We therefore have no theoretical basis to claim that such a "random" sample is any more representative than a convenience sample.

Second, a diverse sample may now be representative. Serra &Kunc (2015) report using stratified random sampling, but may have actually used stratified convenience sampling as the LinkedIn website does not appear to support random selection. *Random* means that every element in the population has an equal probability of selection, not that elements are selected haphazardly. This is important because it is the randomness, not the stratification, that underlies the argument to representativeness. Again, a stratified convenience sample is appropriate for this particular study; the problem is the lack of representative sampling in the literature overall.

In summary, because we lack population lists for many software engineering phenomena, researchers are forced to use non-random sampling or surrogate populations that differ from the target population in unknown ways. This presents major challenges for statistically generalizing results.

2.7.3 Challenge 3: Identifying Empirically Validated and Actionable Antecendents

Unsurprisingly, software project success typically has no single dominant cause, and while a particular failure could have a single dominant cause, different projects fail for different reasons. Some studies therefore investigate the relationship between success and one or a few antecedents, while others attempt to classify antecedents into a meaningful list or taxonomy. In our sample, three papers focus on a single antecedent: user involvement (Bano and Zowghi 2015), benefits realization management (Serra and Kunc 2015) and network embeddedness (Grewal et al. 2006). Meanwhile, four of the papers present an extensive list or taxonomy of antecedents, causes, correlates, factors or predictors of success.

Three research challenges with these taxonomies are evident:

1. There is little agreement as to the composition of, structure of or relative importance of items within taxonomies of success antecedents. Their veracity and reliability are therefore in doubt.
2. Many taxonomies include non-actionable "success factors." For example, suppose that network embeddedness is a primary driver of success in open source software projects.

Researchers can use this knowledge to improve predictive models, but it is simply not actionable for practitioners.

3. Above, we differentiate between actionable antecedents (practices) and non-actionable antecedents (contextual factors). Existing taxonomies tend not to distinguish between kinds of antecedents; for instance, actionable practices vs. contextual factors; predictors vs. predictions; correlates vs. causes.

2.7.4 Other Findings

Other common shortcomings detected include lack of explicit research questions and of descriptions of research methods. Six of the ten papers do not explicitly state a research question. In two cases, one can easily infer the research question from the paper, but in the other four, it is less clear. Three of the papers analyze data, but do not give details of their method of analysis *at all*. This low bar for methodological rigor may be hurting the field's perceived legitimacy.

Notwithstanding these challenges and shortcomings, there is some good news. The Iron Triangle (i.e., the idea that quality is constrained by time, budget and scope) is often inappropriately used as a model of success in software engineering. That is, many studies measure success in terms of meeting schedules, budgets and requirements. Of the three papers that mention the Iron Triangle, Atkinson (1999) and Kaplan et al. (2009) reject it as oversimplified and inappropriate.

Furthermore, the papers bring up many interesting ideas. For example, Linberg (1999) describes how developers had to break rules and bypass management to succeed, with other developers secretly helping them behind management's back. This kind of constructive, deviant behavior exemplifies the importance of rejecting bureaucracy—a key tenant of contemporary management research, not to mention the underlying principle of Agile and Lean development.

3 Introducing the Special Issue Papers

For this special issue, we have selected four papers focusing on failure factors and one focusing on success factors. Two papers in our selection are longitudinal case studies, based on data collected over an extended period of time from the same organization in a limited context. Such studies have the capacity to reveal deep insights not easily attainable in larger, more diverse samples. The remaining two are surveys conducted with considerably more diverse samples across multiple organizations, and aiming at capturing commonalities with a better chance for generalizability within their targeted scopes.

These studies are neither immune to the inevitable construct validity and sampling bias issues that were pervasive in our own literature review, nor do they agree on the same set of antecedents, with each being sourced in a different context and focusing on different measures, factors, triggers, and effects. They nonetheless offer many worthwhile, relevant takeaways, which we hope you, the readers, can use in your own contexts. The selections also raise numerous interesting questions to inspire further research on software engineering success and failure.

The first paper "Naming the Pain in Requirements Engineering: Contemporary Problems, Causes, and Effects in Practice" (DOI 10.1007/s10664-016-9451-7) is the culmination of a still

ongoing, multi-year, multi-country initiative by the requirements engineering research community. It is based on survey data collected using a common instrument by Mendez Fernandez and his many collaborators from Europe, North America, and South America. This paper fits best under Topic 2 from our text clustering (Table 2),which included "SRS" (software requirements specification). The data collected from over 200 companies investigates requirements-related failure factors using multiple measures separated into different categories: customer measures (dissatisfaction, low acceptance), product measures (missing functionality, poor quality, low business value), project/organizational measures (schedule and budget overrun, high lifecycle costs), validation and verification measures (test inefficiency, validation difficulty), and design and implementation measures (irrelevant requirements, low traceability, requirements volatility). No explicit practices are identified, but many are implied by failure factors related to organizational attributes, methods and tools used, and quality and nature of software development inputs. Size of organization (small, medium or large) and software process used (dichotomized as plan-driven or agile) are two antecedents that were considered. Failure measures that were found to be dominant include incomplete/hidden requirements, insufficient communication between project team and customer, and moving targets (project volatility). While incomplete/hidden requirements applied to all types of organizations, communication issues were not found to be widespread in small, plan-driven organizations and moving targets were prevalent mainly in plan-driven and large organizations. Furthermore, the authors identified lack of experience and skills, time pressure, lack of business vision, poor elicitation techniques, overly abstract specifications, and missing completeness checks as the main causes of dominant failure modes.

The second paper "User Satisfaction and System Success: An Empirical Exploration of User Involvement in Software Development" (DOI 10.1007/s10664-016-9465-1) by Bano, Zowghi, and Da Rimini is a longitudinal case study investigating, this time, success rather than failure, again in a requirements-related context. This paper fits well under Topic 6, "User", in our text clustering (Table 2). The study has a much more focused measure than the paper by Mendez-Fernandez et al. The adopted measure of success is user satisfaction, with the main hypothesized practice affecting this measure being user involvement. The data were collected from two projects in a single organization over a period of three years and analyzed using a qualitative approach combining interviews, workshops and artefact analysis. Context factors that were considered include schedule, budget, and management style. The study has three main conclusions. First, user satisfaction, and hence the notion of success, is not static and varies through the different stages of a project. Second, success, as defined, is achievable even when schedule and budget goals are not met. Third, user involvement, leading to acceptable level of user satisfaction, is associated with effective management strategies and high levels of user representation. The second insight is notable because being on-budget and on-schedule have traditionally been considered among primary success criteria for software projects. The authors thus downgrade their influence from a primary to contextual role and posit that the right notion of success be independent of budget and schedule concerns.

The third paper "Failures to be Celebrated: An Analysis of Major Pivots of Software Startups" (DOI 10.1007/s10664-016-9458-0) by Bajwa, Wang, Nguyen Duc, and Abrahamsson change the context to that of software startups. In a rare treatment, it adopts pivots, or strategic changes in business model or product characteristics, as preventive actions that pre-empt potential failures. Thus, there is no explicit failure measure, but pivots are

thought of as failure signals, the absence of which would presumably have resulted in a downstream failure. Then possible triggers for pivots are interpreted as proxies for failure factors. The data is collected from nearly 50 organizations using the case survey method to identify major types of pivots and their triggering factors. The authors found that changing or wrongly understood customer needs along with changes in targeted market segments were the most common market-related pivots, and changing the product functionality to focus on a specific feature and changing the solution to use a different technology or platform were the most common product-related pivots. Most pervasive triggers associated with these pivots were, unsurprisingly, negative customer reaction and flawed business model. The unique aspect of this work is its treatment of failures in a positive light as opportunities for validated learning for future success, consistent with the startup culture. While our text clustering identified several contexts and domains (e.g., ERP, agile, healthcare, outsourcing), none of these is a match for this paper, highlighting the novelty of its context.

The last paper, by Lehtinen, Itkonen, and Lassenius and titled "Recurring Opinions or Productive Improvements—What Agile Teams Actually Discuss in Retrospectives" (DOI 10.1007/s10664-016-9464-2), is another longitudinal case study. It fits well under Topic 3 from our text clustering— "Agile + Decisions" (Table 2). The paper's main focus is process improvement. The study is sourced on data collected from retrospective meetings over a period of close to three years in the same organization that used an agile software development process. The authors do not explicitly identify measures of success or failure, or associate such measures with specific antecedents (practices or context factors). Rather, they explore themes that recur in retrospective meetings based on subjective impressions of the team members. Positive impressions are implicitly interpreted as proxies for success outcomes and underlying factors, and negative impression as proxies for potential failure outcomes and underlying factors. The emphasis of the paper is on failure-related factors, with subsequent corrective actions associated with negative impressions taken as evidence for those factors' potential impact. The authors highlight a small subset of these factors—estimation accuracy, bug fixing processes, resource availability, schedule constraints, and clarity of specifications/instructions—as most important, but caution that personal biases and interests of team members as well as the controllability and complexity of the issues and solutions likely influenced the results. They identify recurrence of themes over time as an indication of high complexity and lack of controllability for those themes. Estimation accuracy was identified as one of those themes. When recurrence happens despite repeated corrective action, this is interpreted as a potential waste of resources. The most prominent example of such waste was the effort spent on discussing and addressing bug fixing processes despite any notable improvements.

The selections illustrate the diversity of perspectives with respect to software engineering success and failure. The second paper by Bano, Zowghi, and Da Rimini and the third paper by Bajwa, Wang, Nguyen Duc, and Abrahamsson both depart from pre-determined, external notions and antecedents of success or failure that relate to user, customer, or market needs, yet they take distinctly different approaches in methods and treatment. The first paper by Mendez Fernandez focuses both on internal and external notions and antecedents of success or failure and the last paper by Lehtinen, Itkonen, and Lassenius focuses mostly on internal ones. Both works are exploratory, in that the notions and ancedents emerge from the date rather than being pre-supposed. Like the second and third papers, the similarities stop there since their methods, treatments, and conclusions are, again, distinct, reflecting the idiosyncrasies of the particular contexts in which they were conducted.

4 Conclusion

This special issue on software engineering success and failure presents four sound empirical studies. These studies provide novel and worthwhile contributions on requirements engineering, user satisfaction, startup pivots, retrospective discussions and their relationships to the success or failure of software projects.

To help situate these studies, we conducted a brief systematic literature review. The main takeaways from our analysis are:

- research on software project success and failure is diverse with many practices and context factors potentially influencing outcomes;
- this research is published in at least three academic communities of practice: software engineering, information systems, and project management;
- most of the research in this area has been published in the past decade; and
- enduring methodological challenges in this area include developing sound instruments for measuring success, representative sampling without population lists, and creating both empirically sound and practically actionable taxonomies of success antecedents.

Clearly, how to understand, measure, and improve the chances of success remain fundamental questions for software engineering research, with many continuing challenges for future research. We hope you find the discussions thereof in this special issue stimulating and useful. Enjoy!

Appendix: Coding Scheme

The qualitative analysis of the literature review was based on the following questions:

1. How does the paper operationalize success (e.g., as stakeholder satisfaction) or measure success (e.g., using subjective assessment of participants)?
2. How does the paper operationalize software engineering / does the paper use a surrogate or subtype of SE (e.g., investigating ERP implementation projects rather than software projecs in general)?
3. What is the paper saying about success or failure?
4. Does the paper propose/test/discuss antecedents (i.e., causes, factors, correlates) of success or failure? What are the antecedents?
5. What are the paper's core concepts or topics (e.g., risk, open source, agile)?
6. What are the paper's research questions, purposes, theses or key positions?
7. Does the paper apply any core foundation, theory, taxonomy or viewpoint (e.g., the Project Triangle)?
8. Is the paper empirical? If so:
    a. what is its sampling strategy / who or what is being studied?
    b. what data collection methods are used (e.g., questionnaire, experiment, case study, SLR)
    c. what analytical methods are used (e.g., correlation-based stats, inductive coding, discourse analysis, deconstruction)?
9. Where was the paper published (i.e., is there anything unusual about the publication outlets)?

References

Agarwal N, Rathod U (2006) Defining 'success' for software projects: an exploratory revelation. Int J Proj Manag 24(4):358–370

Agrawal, Amritanshu, Wei Fu, and Tim Menzies. (2016). "What Is Wrong with Topic Modeling? (and How to Fix It Using Search-Based Se)." *arXiv Preprint arXiv:1608.08176*. https://arxiv.org/abs/1608.08176

Atkinson R (1999) Project management: cost, time and quality, two best guesses and a phenomenon—it's time to accept other success criteria. Int J Proj Manag 17(6):337–342

Bano M, Zowghi D (2015) A systematic review on the relationship between user involvement and system success. Inf Softw Technol 58:148–169

Chen CC, Law CCH, Yang SC (2009) Managing ERP implementation failure: a project management perspective. IEEE Trans Eng Manag 56(1):157–170

Fellows I (2012) Wordcloud: word clouds. R Package Version 2:109

Garousi V, Mäntylä MV (2016) Citations, research topics and active countries in software engineering: a bibliometrics study. Comput Sci Rev 19:56–77

Grewal R, Lilien GL, Mallapragada G (2006) Location, location, location: How network embeddedness affects project success in open source systems. Manag Sci 52(7):1043–1056

Griffiths TL, Steyvers M (2004) Finding scientific topics. Proceedings of the National Academy of Sciences 101(suppl. 1):5228-5235

Kaplan B, Harris-Salamone KD (2009) Health IT success and failure: recommendations from literature and an AMIA workshop. J Am Med Inform Assoc 16(3):291–299

Kappelman LA, McKeeman R, Zhang L (2006) Early warning signs of IT project failure: the dominant dozen. Inf Syst Manag 23(4):31–36

Khan SU, Niazi M, Ahmad R (2012) Empirical investigation of success factors for offshore software development outsourcing vendors. IET Softw 6(1):1–15

Lehtinen TOA, Mäntylä MV, Vanhanen J, Itkonen J, Lassenius C (2014) Perceived causes of software project failures–an analysis of their relationships. Inf Softw Technol 56(6):623–43

Linberg KR (1999) Software developer perceptions about software project failure: a case study. J Syst Softw 49(2):177–192

Martins Serra CE, Martin K (2015) Benefits realisation management and its influence on project success and on the execution of business strategies. Int J Proj Manag 33(1):53–66

Ralph, Paul, and Paul Kelly (2014) "The Dimensions of Software Engineering Success." In *Proceedings of the 36th International Conference on Software Engineering*, 24–35. ACM. http://dl.acm.org/citation.cfm?id=2568261

The Standish Group International Inc (1994) "The Chaos Report (1994)." The Standish Group International Inc

Trochim WMK (2001) Research methods knowledge base. Atomic Dog Publishing, Ohio, http://www.anatomyfacts.com/research/researchmethodsknowledgebase.pdf. Accessed 6 Feb 2017

Mika V. Mäntylä is professor of Software Engineering at the University of Oulu, Finland. He received a D. Sc. degree in 2009 in software engineering from the Helsinki University of Technology, Finland. His research

interests include empirical software engineering, software testing, mining software repositories, and behavioral software engineering. He has previously worked as a post-doc at the Lund University, Sweden and as an assistant professor at the Aalto University, Finland. His previous studies have appeared in journals such as IEEE Transaction on Software Engineering, Empirical Software Engineering, and Information and Software Technology. For more information, visit: http://mikamantyla.eu/.

Magne Jørgensen received the Diplom Ingeneur degree in Wirtschaftswissenschaften from the University of Karlsruhe, Germany, in 1988 and the Dr. Scient. degree in informatics from the University of Oslo, Norway, in 1994. He is a professor of software engineering at the University of Oslo and a chief research scientist at the Simula Research Laboratory. He has several years industry experience, and combines academic research with advisory work for software companies. His research interests include project management, human judgment and software economics.

Paul Ralph is an award-winning scientist, author and consultant, a senior lecturer in computer science at The University of Auckland and a visiting assistant professor of management at the University of British Columbia. His research centers on the empirical study of software and game development, including projects, processes, practices, tools and developer cognition, socialization, productivity, creativity, wellbeing and effectiveness. Dr. Ralph is the founding director of the Auckland Game Lab, co-founder of the AIS Special Interest Group for Game Design and Research (SIGGAME) and a member of the IEEE Technical Council on Software Engineering and ACM Special Interest Group on Software Engineering. He holds a PhD in Management from the University of British Columbia.

Hakan Erdogmus is an Associate Teaching Professor of Electrical and Computer Engineering at Carnegie Mellon University's Silicon Valley campus. Prior to joining CMU, he was an independent consultant providing training in software process and project finance and a senior research officer at the Canadian National Research Council. His research focuses on software process, modern development techniques, software quality, software engineering economics, and empirical software engineering. He holds a Ph.D. in Telecommunications from INRS-Université du Québec and an M.Sc. in Computer Science from McGill University, Montreal. Dr. Erdogmus is a former Editor in Chief of IEEE Software, a Senior Member of IEEE, a Golden Core member of IEEE Computer Society, and a member of ACM. He served in IEEE Computer Society's Board of Governors from 2012 to 2014.