Four presumed gaps in the software engineering research community's knowledge

Lutz Prechelt *Freie Universität Berlin*
Berlin, Germany
prechelt@inf.fu-berlin.de

Abstract—*Background:* The state of the art in software engineering consists of a myriad of contributions and the gaps between them; it is difficult to characterize.
Questions: In order to help understanding the state of the art, can we identify gaps in our knowledge that are at a very general, widely relevant level? Which research directions do these gaps suggest?
Method: 54 expert interviews with senior members of the ICSE community, evaluated qualitatively using elements of Grounded Theory Methodology.
Results: Our understanding of complexity, of good-enoughness, and of developers' strengths is underdeveloped. Some other relevant factors' relevance is apparently not clear. Software engineering is not yet an evidence-based discipline.
Conclusion: More software engineering research should concern itself with emergence phenomena, with how engineering tradeoffs are made, with the assumptions underlying research works, and with creating certain taxonomies. Such work would also allow software engineering to become more evidence-based.

1 Introduction

1.1 The state of knowledge about software engineering

What do we[1] know about software engineering (SE)? How does it work? What is the state of the art?

These questions are difficult to answer, because the things we know are so specialized that the overall landscape of our knowledge is highly fragmented.

For instance the *Guide to the Software Engineering Body of Knowledge* [1], despite its common abbreviation as SWEBOK, is only a guide to the knowledge (a catalog of sources), not the knowledge itself, and still comprises more than 300 pages. It captures well-established knowledge, not the most recent state of the art.

At the SE research front, Kitchenham and others in 2004 suggested to perform systematic literature reviews (SLRs) in order to synthesize and consolidate research results in order to enable evidence-based SE practice [2]–[4]. This was successful and many SLRs have appeared since[2], but they help only gradually in fighting the fragmentation, for several reasons: Most of them are mapping studies (that merely catalog results) rather than full SLRs (that synthesize combined results; see [6, Table 2] and [5, Table 2]), different SLRs are dominated by different study methods with different validity characteristics (e.g. experiments [10] vs. surveys [11]), and SLR quality varies widely (e.g. compare [12] to [13]). Most importantly, however, few SLRs are about widely applicable areas within SE (e.g. agile methods [14], motivation [11]) while most are rather more specialized in terms of their usage contexts; for instance: agile practices in distributed SE [15], the role of trust in agile methods [13], effectiveness of pair programming [10], search-based SE techniques involving user interaction [16], unit testing for BPEL routines [17].

This fragmentation is problematic because it makes it difficult to judge the relevance of a proposed research question: What importance does an answer have for SE overall? How much of an impediment is the gap that the answer might fill? Sure, we can argue why a given research question A appears more relevant than question B if A and B are related. But typically we cannot – and there is hardly any basis for systematically *deriving* relevant research questions.

1.2 What do we not know?

An alternative approach to making statements about the state of SE knowledge is to characterize things we do *not* know. And if we do so not at a specialized level (where the list would be nearly endless) but at a general one, this would overcome the fragmentation. In that case, each knowledge gap that is pointed out would have relevance for many, perhaps even most SE situations.

Finding such general-level gaps in the SE knowledge is the aim of the present work. The underlying study was largely emergent (rather than designed upfront) and came to be as follows:

1.3 A conundrum

Ever since I entered empirical software engineering research in 1995, I have been troubled by an apparent paradox: On

1. "we" here is the software engineering research community; the people whose job it is to build and explicate knowledge about software engineering.

2. There are even some SLRs of SLRs ("tertiary studies"), but most of these are concerned with research methods [5]–[8], not SE methods [9].

the one hand, even modest software is highly complex, few (if any) software systems ever work fully satisfactorily, and the people building those systems all have strong cognitive and other limitations. On the other hand, despite many deficiencies and local breakdowns, overall the software ecosystem appears to work OK. **How do we[3] manage to achieve that?**

When this question hit me once again the week before ICSE 2018 (International Conference on Software Engineering), I recognized that asking my colleagues about it might be a source of insight. I worked out a concise formulation and set out to run a number of interviews during the ICSE conference week. The rich answers I got prompted me to perform a thorough evaluation, which turned into the present article. I did *not* initially have an idea whether or how my interviews might turn into a research contribution; this emerged only during analysis. Note also that the outcome is about knowledge *gaps*, it does *not* answer the above question.

1.4 Research contributions

Formulating research *questions* for the present work would be misleading, as they were completely emergent; so I formulate contributions only.

Imagine, purely conceptually, a Software Engineering Topic Tree (SETT)[4] that arranges SE research articles according to their degree of specialization, with "all of SE" at the top. Then most SE articles would be at or near leaves, systematic literature reviews would be at medium levels, and the present article would be near the top. In this sense,

- the article derives six areas in which our knowledge is weakly developed and that each have relevance for a broad variety of SE contexts;
- the article suggests research directions to fill these knowledge gaps.

The interview respondents' views are opinions, not facts. But the gaps in understanding that show up in these views are arguably facts, not opinions; the article presents evidence for each of them. In contrast, the research directions are merely sketched; a fuller discussion is left to future work.

2 Method

I ran 54 semi-structured expert interviews based on a 4-item stimulus card and analyzed them using various elements of the Grounded Theory Methodology (GTM), in particular Open Coding, paying particular attention to elements that are *missing* in the responses. I report the outcomes in a topic-by-topic manner.

2.1 Epistemological stance

My stance is constructionist-interpretive [18], not positivist: We should not assume the existence of a single underlying truth; the data requires interpretation and the person of the researcher will unavoidably have influence on the outcome; alternative interpretations of these same data may exist that have similar validity. This does not threaten the *validity* of the present interpretation, but in terms of *appropriateness*, individual readers may prefer alternative interpretations in some places.

2.2 Design goals and approach

Although the study overall does not have an upfront design, the interviews have. I pursued the following ideas:

1) *Openness:* Due to the lack of a better idea, the study would use my own perception of the situation as a starting point (as sketched in the introduction), but beyond that I avoided to direct the attention of my respondents in any particular direction. The approach was to use basically just a single question. As far as I can see, this has worked well. For the same reason, I kept interjections and followup questions during the interviews to a minimum. This has worked well for many interviews, but presumably made some others poorer.
2) *Breadth over depth:* I expected responses to be highly diverse (which indeed they were) and thus preferred getting many short interviews over fewer long ones.
3) *Respondent diversity:* I strove to represent women disproportionately highly, because I suspected their views might have a different distribution. I strove to have respondents from as many countries as I could, because I suspected that national perspectives might differ. With one exception, a PhD student, all respondents are senior researchers. This may or may not have been a good idea.
4) *Provocativeness:* To reduce the risk for boilerplate-style responses, the interview stimulus used pointed formulations that I expected many respondents to disagree with. The formulations were also intentionally ambiguous to prompt differentiation (an important aspect, please keep it in mind). Both of these approaches worked in many if not all cases.

2.3 Interview structure: The stimulus card

The interview was based on the 142 mm by 104 mm laminated card shown in Figure 1.

Its rear side contained a minimal informed consent agreement appropriate for researchers:

> May I record this? (audio)
> Anonymous.
> Will quote only anonymously.

All respondents agreed to this. One requested to have the recording deleted after transcription, which I did.

The front side contained three statements (which I will call S1, S2, S3) meant to establish a context for the main interview question (which I will call QQ).

- **S1**: Software systems are complex.
- **S2**: Most software systems work only approximately well.
- **S3**: Most software engineers have only modest capabilities.
- **QQ**: What keeps the software systems world from breaking down?

3. Subsequently in this article, "we" can stand for the software engineering community (as in the present case), the software engineering research community, the ICSE community, or the union of authors and readers of this article.

4. This concept will be used again in Related Work.

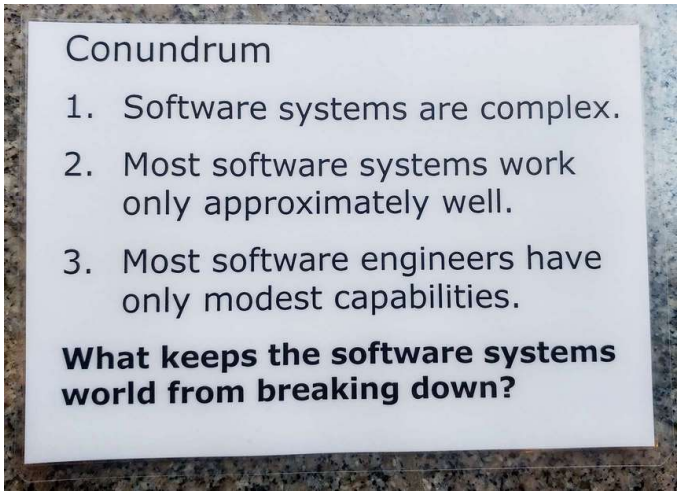

Fig. 1. Stimulus card (front)

2.4 Interviewer behavior

To acquire respondents, I would approach a candidate physically and either ask "Hi <firstname>! Would you have time for a 5-minute, 1-question interview?" or: ask "Hi <firstname>! Would you have time for a 5-minute interview on this question?" and hand them the stimulus card for review.

Most interviews started there and then. Others were postponed one or more times because the candidates were busy or hesitant. In some cases this did not converge and they are now missing.

At interview start, I would almost always hand over the stimulus card so that the statements and question became physically tangible and could be pointed to. My only prompt would be something like "OK, shoot.".

During the interview, my only common interventions would be these:

- When respondents spoke about individual software systems rather than the whole ecosystem, I would emphasize the question was about the "software systems *world*" and then often elaborate that this meant "widespread collapse".
- When respondents gave rather general answers, I would ask for specific "ingredients", "aspects", or "factors" that were most relevant for how we prevent breakdown.
- Once one or more key ingredients had been named, I would often ask the follow-up question "Are ICSE and the contributions it is seeing well aligned with what's important for bringing software engineering forward?". Its actual formulation often picked up statements the respondent had made before.[5]
- I sometimes asked short questions for clarification or repeated, in different words, what I had understood.

I generally avoided nudging respondents in any particular direction. The interview ended when the respondent had nothing more to say.

5. The answers to this question will not be discussed here, but rather in a separate arXiv article.

Later interviewees overheard parts of an interview in about 10% of the cases.

2.5 Equipment used

The only equipment used for the interviews were the laminated stimulus card and an Olympus VN-8700PC mono voice recorder, which produced clear recordings despite the often enormous background noise. Both fit easily in my jeans pockets so that I was ready for an interview at all times.

2.6 Respondent demographics

Of the 54 respondents, 18 (33%) were female. The respondents came from 18 different countries of affiliation: USA (24 interviewees), Germany (5), Canada (4), UK (4), Switzerland (2), India (2), Netherlands (2), Australia, Spain, Finland, Greece, Ireland, Israel, Italy, Luxembourg, South Africa, Sweden, Singapore.

All interviews but one were held in English; I noticed two respondents having some difficulty expressing their thoughts in English, all others appeared fluent.

16 respondents have previously been chairs of program committees of the ICSE Technical Research track, another 5 have been ICSE General Chairs, and yet another 8 have been PC chairs of other ICSE article tracks such as SEIP, NIER, SEIS, SEET. So at least 29 respondents (54%) would be considered *very* senior and many of the others were similarly accomplished.

2.7 Interview metadata

I ran 51 interviews during the ICSE week (from Sunday 2018-05-27 to Friday 2018-06-01) and 3 more within seventeen days after that, with people who could not make time for the interview during the ICSE week; 54 interviews overall. These range from 2 to 34 minutes in length, the middle half being 4 to 10 minutes. A few interviews consist of two parts, where a respondent approached me again shortly after the interview to add something.

Respondent quotes will be attributed to respondent pseudonyms chosen according to the names of the recording files: R394 to R488 (with gaps).

2.8 Data analysis

Analysis used elements of Grounded Theory Methodology GTM [19]–[21]. The approach is eclectic, taking the basic concepts of Open Coding [19, II.5], Constant Comparison [19, II.6], Selective Coding [19, II.8], applied multiple times, the "as much as needed" approach to transcription [19, p.30], and some Memoing [19, III.12], from Strauss and Corbin, a particular emphasis of Theoretical Sensitivity from Glaser [20] and Strauss/Corbin [19, I.3] alike, and the Constructionist epistemological stance (with some resulting aspects such as insistence on recordings as opposed to mere notes and heavy reliance on quotations in the present report) from Charmaz [18], [21].

The outcome does not claim to be a theory, so it is unproblematic that other GTM elements are missing: Axial Coding [19, II.7] is hardly applicable to this data, Theoretical

Sampling [19, II.11] and determining Theoretical Saturation [19, p.178] were impractical.

The analysis started by turning the recordings into text. This served two purposes: Speeding up subsequent steps and protecting respondent anonymity in the published data (see below). I transcribed concise pertinent parts verbatim (shown like this: "verbatim quotation"[R123], if it came from respondent [R123]), paraphrased verbose pertinent parts near-verbatim ("[paraphrased quotation]"[R123]), and paraphrased less-and-less relevant parts with higher-and-higher compression. In some cases I subsequently expanded paraphrased parts.

Right after textifying each interview, I annotated them with a preliminary set of concepts that grew in Open Coding manner over time, using the MaxQDA 2018[6] software.

In a second pass through the material I created a very short (1 to 6 lines) summary of each interview, all in a single file, to capture the gist of each interview in order to avoid distortion from over-emphasizing low-relevance aspects.

During this second pass, I recognized the three different frames of reference relevant in these data (Section 3) and discovered the core categories that led to the main insights (Sections 4 to 7). As a consequence, rather than going for a completely new second generation of concepts for my annotations as planned, I merely consolidated the first generation enough to support finding the pertinent material and validating the statements. I limited the annotation precision to the level needed to support the rough quantification provided here (see Section 2.9) and ensure the validity of the quotations.

All remaining analysis was then driven by the development and validation of the narrative you find in this article.

2.9 Quantification

Given the opinions presented below, it is tempting to ask for quantification of their frequency. We will have to resist that urge.

Not only is it unclear what such frequencies would mean, because we are not looking at a *representative* sample of our population. Worse: Although GTM tailors the categories to the data, the frequencies cannot be determined properly, because many responses are highly ambiguous or multilayered.[7]

To reflect at least large differences in frequency, I will use a coarse, 4-level, linguistic, roughly logarithmic ordinal scale as follows: "most" means at least 1-in-2; "many" means at least 1-in-4; "some" means at least 1-in-8; "few" means at least 2, but less than 1-in-8. The basis for these ratios is always smaller than 54, because there is no single aspect that all respondents commented on. Except where noted explicitly, I do not report concepts that occur only once in my data.

2.10 Literature micro-studies

To roughly triangulate [23] the interview-based evidence with a very different data source, I performed a number of micro-studies.[8] Most of these consider the 153 articles from the ICSE 2018 Technical Research track (including the 47 Journal First entries) in order to estimate, roughly, how often SE research addresses a certain issue. The classification is made based on article title, session title, and (where needed) the abstract. The outcomes serve as modest corroboration of the interview interpretations, not as research results in their own right. Details for the micro-studies are included in the above-mentioned raw data package as well.

2.11 Member check

I sent a complete draft of the article to the respondents for a member check, asking for possible objections to each person's quotations, the insights, and the conclusions. Ten respondents reacted. As a result, I corrected the handling of two quotations and inserted minor clarifications in a few places. There were some comments on, but no objections against any of the insights or conclusions.

2.12 Chain of evidence

The chain of evidence for the present study consists of (1) the raw interview data, (2) a set of first-generation and (3) second concepts from open coding, (4) concepts from selective coding, and (5) the micro-studies. In this article, (1) is represented by loads of verbatim quotations, (4) and (5) as argumentative text. (3) is mostly and (2) is completely suppressed for sake of readability; see Section 8 for a discussion.

A raw data package (interview transcripts, annotations, code memos, etc.) for this study is available to fill in details.[9] Quotations reported in the article were chosen for their brevity first and to show the breadth of opinions and formulations second.

3 Deduction Method and Nature of the Results

When you think about the results below, be aware there are three different frames of reference involved:

- Software engineering space (SES) is the space of real-world SE phenomena.
- SE knowledge space (KS) is the space of what we, the SE research community, know about SES.
- Interview space (IS) is the space of what my respondents say about SES and KS.

When interpreted as statements of fact at the SES level, the respondents' IS statements can be dismissed as mere opinions, but this is *not* the point of this article and not its level of discussion. Rather, the article mainly uses the IS statements to draw conclusions about KS — and mostly does so by analyzing what is *not* being said with respect to S1, S2, S3, or QQ.

The reasoning works like this:
1. QQ asks for a rough theory of how SE works and S1, S2, S3 are relevant only to help stating this theory.

6. https://www.maxqda.com/

7. It is possible to nail down categories completely despite such effects, but this is *extremely* laborious and would not pay off in the present case. For instance [22] required about 7 person-years of work to understand 60 concepts precisely.

8. Another one is for supporting a statement in the Related Work section.

9. https://www.dropbox.com/s/woabq7kuxipojz8/knowgaps-materials.zip?dl=0

2. If none or nearly none of the expert respondents use some concept X although (i) X would be useful for their discussion of S1, S2, S3, QQ and (ii) the respondents are researchers and therefore normally justify their views, I conclude that knowledge of X is not or not readily available to those respondents. Let us call this conclusion CX.
3. Since the respondents are many, broadly selected, senior representatives of the ICSE community, I conclude that CX holds for the ICSE community as a whole.
4. Since ICSE represents top-quality SE research broadly, I further conclude that CX holds for the SE research community as a whole.

4 S1: Software Systems are Complex

If respondents comment on S1, S2, or S3 at all (many did not), I expect their comments to provide support for their answer to QQ and analyze them from that perspective.

The result sections on S1, S2, S3, and QQ follow a common pattern: They first describe general observations about the responses, then formulate an insight, and then present evidence supporting that insight.

Observations: S1 was meant as a common starting point, a self-evident consensus statement present only to set the stage for the controversial statements S2 and S3 following it. And indeed almost everybody of the 27 respondents who commented on S1 agreed with it:
"I definitely agree that's the case."[R471] or
"And yeah, of course: Software is complex"[R462] or
"I'd agree with all three of those; that seems fairly uncontroversial to me."[R452] and so on. A few remarked it wasn't unconditionally true, e.g.
"I'm willing to accept that as a statement [sic!]; not all software systems are complex"[R445].

To my surprise, two respondents rejected S1. They both referred to the execution state space and argued that only a tiny part of it was actually relevant:
"A small number of variables matter"[R400] or
"actually [developers] don't have to reason about all of it, they just have to reason about a little bit of it"[R414].

Reviewing the evidence offered by the large majority that had agreed to S1 led to

Insight 1: Our notion of complexity is undercomplex.

I will first explain this by evidence taken from the interviews (indicated by the 👤 symbol), then by further evidence from a micro-study (symbol 📄).

👤 Evidence 1a: The respondents who agreed with S1 offered no evidence in support of their claim nor even a definition of complexity. Where an implicit concept of complexity became visible at all, it was either size ("you're asking about software systems that are complex, not small ones; [...] a few million lines of code"[R452]) or it referred to some effect of complexity ("[...] not just that it's too complex to understand [...]"[R467]).

Few respondents differentiated complexity into different kinds (technical vs. socio-technical; [R416,R446]) or complexity from different sources (components, interaction, algorithm, distribution, scheduling; [R396]). No respondent offered a reference that would spell out the discriminations. □

📄 Evidence 1b: There is a trans-disciplinary literature on Systems Science and Complex Systems [24], [25] which offers a number of attributes associated with complexity, such as nonlinearity, emergence, self-organization, adaptation, or feedback loops. Of these, the one most broadly applicable to software is emergence, but current SE literature has little discussion of that. Searching for the word stem "emerg" in the PDFs of the ICSE 2018 technical research track finds 144 matches in 35 articles, but zero of those relate to software complexity. The closest miss, a work on "emerging issues", talks about a merely temporal aspect [26]. □

5 S2: Most Software Systems Work Only Approximately Well

Observations: The purposefully ambiguous and provocative S2 question resulted in a lot more discussion. Many of the 33 respondents who commented on S2 criticized the formulation, e.g. by pointing out that SE shoots at a moving target ("Requirements evolve"[R439]). Two relevant criticisms recurred. First, some respondents found the word 'only' to be overly negative, e.g.
"[...] that's a celebration!"[R400] or
"'Approximately' is what you want in engineering"[R419].

Second, some pointed out that software is part of a socio-technical system. In some cases this was apparently crucial for giving a positive answer:
"Software plus society: yes."[R414] or
"they are working sufficiently well so that they are useful."[R430] or
"most systems work sufficiently well. They support their community, support society."[R448]

In one case it was reason for a strongly negative one: "I think of what we are doing as racking up social debt. Analogous to technical debt. Big time."[R446].

Overall, half the respondents who commented on S2 agreed with it; some disagreed, nearly all of them in the optimistic direction ("Not true [...]. I think they work very well."[R440] or "Most software systems that are used work well for the purposes their users expect of them and therefore they work well."[R445]) and many took no clear position.

But lurking behind all this differentiation is a lot of vagueness in a relevant area:

Insight 2: We lack understanding of what makes software "good enough".

I will again first explain this by evidence taken from the interviews, then by further evidence from a micro-study.

👤 Evidence 2a: The idea of software being *good enough* and working *sufficiently well* came up in many responses and in many guises, e.g.
as a self-evident observation:
"Approximately well may be well enough, that's the point."[R397];
as typical-case satisficing:
"they work approximately well. That's often good enough; it's usually good enough."[R397];
as a sufficient-for-praise level of quality:
"But we have many many computer systems doing amazing things all the time; and working very well, working well enough."[R418];
as an investment decision and engineering tradeoff:

"It's always how much do you want to invest to get your software right and when is it good enough"[R396];
as defined by socio-technical criteria:
"Many systems are working and they are working sufficiently well so that they are useful."[R430];
as a morally justifiable standard:
"In most cases we're building systems that are good enough for their purpose and there's nothing wrong with that."[R448];
as a matter of life and death:
"I want my airbags to go off; and those work pretty well"[R471];

Like the latter, many comments on S2 and (more often) QQ suggested that high-stakes software ("airplanes"[Several]) usually worked well while low-stakes software ("apps"[Several]) often did not but that this was acceptable. But what about the huge region in between: medium-stakes software? This was hardly ever mentioned and nobody ever offered a definition of "good enough" for this realm. Indeed, given the multitude of perspectives offered above, a definition is not obvious. □

Given that this notion appears to sit at the very center of software quality, SE research without such a definition does not appear good enough.

EVIDENCE 2B: In a literature micro-study, I looked for ICSE 2018 articles that appeared to have any kind of conscious quality tradeoff as a main concern. This is considerably wider than good-enoughness and so provides a conservative estimate how big the problem formulated by Insight 2 is. Considering all 153 articles, the search found only five such articles[10] [27]–[31], or 3%. □

6 S3: Most Software Engineers Have Only Modest Capabilities

OBSERVATIONS: Of the 36 respondents who commented on S3, most agreed with it, if in very different ways:
from "Not all programmers are super-programmers."[R471]
over "Sure, I don't know what more to say."[R416]
and "we know software engineers are not very good at their jobs. I work at one of the best places in the world[11]and still...it scares me."[R415]
to "I've seen software developers; they do mostly suck."[R467].
These statements resonate with topics we are all familiar with from the SE literature when developers are discussed: they often commit mistakes, often produce deficient designs, are often lazy, often lack discipline.

Some respondents, however, *dis*agree with S3 – also in very different ways:
from "To be a software engineer, you need to have good capabilities."[R419]
over "I think most people who develop software and have only modest capabilities are not software engineers."[R459]
to "we're talking about the most intelligent people on the planet here"[R421].

Other comments state that (1) capabilities vary widely, (2) needed capabilities vary widely as well ("People find software systems where the complexity matches their capabilities."[R399]) and (3) a few brilliant people can achieve a lot as enablers for the others ("We get the good people to design the essence, so that everybody else can build the peripheral stuff that is not as critical. Which means that the people that are average can contribute as much as the people that are great."[R395]).

Which sounds a lot more optimistic, but also curiously unfamiliar from the research literature. Which leads to

Insight 3: We don't know much about what software engineers are good at.
This time, we rely on a micro-study only.

EVIDENCE 3A: We determine how many works at ICSE have one or more developer strengths as a main concern or one or more developer weaknesses or both or neither – and expect the latter group to be largest. For instance all works that focus on product rather than process will automatically land in the "neither" group. Considering all 153 articles, we find 14 studies (9%) that are developer-behavior-centric (about actual behavior, not tasks, roles, practices, or expectations), but only six of them (a mere 4%) are outside the "neither" category: Four concern weaknesses [27], [32]–[34], one concerns strengths [30], and one concerns both [35].[12]

Apparently, most developer-centric work is merely descriptive, not evaluative. Within evaluative works, there indeed appear to be more regarding weaknesses than strengths, but both types are rare. □

7 QQ: What Keeps the Software Systems World From Breaking Down?

OBSERVATIONS: While nobody claimed or assumed *everything* was fine and many respondents talked about various cases or degrees of local breakdowns, only few respondents rejected the assumption behind QQ and stated they expected the software systems world to likely break down ("There's very little from keeping the software systems world from breaking down."[R407]) or saw it as breaking down already ("I do not think that the software systems world is not breaking down. It is."[R436]).

At the other end of the spectrum, only few respondents expressed that they firmly expected the software systems world to *not* break down ("I'm totally impressed how software is being constructed."[R439])

The big majority, however, took the assumption of non-breakdown for granted and focused on what they considered key factors preventing breakdown. A majority would initially state only one key factor and add others only after prodding. The ensuing overall list of presumed key factors is long and diverse. I consolidated the list by grouping related concepts (which had originally been formed according to an intuition of sufficient differentness) until, with two exceptions that appeared too relevant, they all had "some" mentions (that is, in at least 7 interviews) or "many" mentions.

There is only one factor with many mentions: The *developers*.
"Of course, people are always the most important."[R431] or
"It's because people are creative."[R460] or
"It isn't our research community that keeps it from breaking

10. Three of these are journal-first publications.
11. practitioner industrial respondent
12. Two of these six are journal-first publications.

down, that's for sure. I think it's the people in industry that figure out what's happening and adapt."[R488].

After that, there are 11 factors mentioned in more than a few interviews (in decreasing order of frequency):

- An *appropriate development process*[13]:
  "Disciplined practices"[R426] or
  "a reasonable process"[R403] or
  "we have learned to think about our processes"[R462].
- The *flexibility of users*:
  "Systems actually do break down all the time, but as humans we work around these systems."[R418] or
  "Humans are resilient; the human world is resilient. Human processes step in"[R445] or
  "expectation management. [We have learned to tell people this is the best they will get – with some bugs in it.]"[R433] or
  "People are incredibly adaptive."[R434].
- *Abstraction*:
  "It all boils down to abstraction."[R485] or
  "'Module' would probably be my biggest."[R438].
- *Software adaptation/evolution*:
  "We organically grow the systems to be more and more complex"[R418] or
  "we continue to invent"[R483].
- *Quality assurance*:
  "inspections"[R474] or
  "a lot of testing being absolutely indispensable."[R428].
- *Repairing flaws*:
  "[Few of our errors] manifest in visible faults. And when they do, we fix them."[R403] or
  "we continue to fix some of the things that weren't working as well in the past."[R483].
- *Good-enoughness* (as a required level of quality far below perfection):
  "Beta-testers find stuff and everybody assumes that's just fine."[R436] or
  "Most of the time, no really terrible things happen."[R462].
- *System resilience* against imperfections:
  "Fault-tolerant software"[R462] or
  "Code is less sensitive to input changes than we think."[R486].
- Development *tool support*:
  "Tooling helps. Much more so today than in the past."[R395] or
  "a lot of advances in computer-supported cooperative work."[R420].
- Developer *collaboration*:
  "Teamwork."[R453] or
  "People together are intelligent enough for fixing stuff that doesn't properly work."[R424].
- *Reuse* of software and approaches:
  "many systems today are built out of components that have proven their worth."[R395] or
  "Solution by analogy."[R460].

I left two factors as separate concepts although they were mentioned in only a few interviews. Both apply predominantly to critical systems:

- *effort/investment*:
  "in many software organizations things get built by brute force. Built and maintained by brute force. That's not [efficient], a lot of effort is wasted, but especially critical systems like flight software, Airbus, things are verified again and again."[R430] or
  "A lot of manpower is invested"[R425] or
  "an enormous amount of redundancy and recheck and carefulness which is built into all of the processes to make sure that that stuff works. And so they check and recheck and recheck and recheck and recheck."[R436].
- *formal methods*:
  "adopting formal methods that have matured enough to get some incremental value out of it."[R414].

All of these 14 were explained in some way by at least a few respondents and appeared sensible and sufficiently important to me to consider them all valid factors. As a list of "key" factors they are already many, but looking at them more closely led to

Insight 4: There is no consensus on a small set of neat key software engineering success factors. [14]

I will first explain how the set is not small, then how the concepts suggested are not neat.

EVIDENCE 4A: The above factors list is misleading in the sense that it may suggest each item is a single, neat concept. And indeed that may be sort of true for some, for instance *abstraction* and *collaboration*. But most are in fact collections of related, but different concepts. For instance the biggest one, *developers*, upon closer inspection falls apart into the remainder represented in the examples given above and two clusters. Cluster 1 talks about developers' attitude:
"people with a particular sense of quality."[R409] or
"Maybe I'm a paranoid software engineer[15]and that's the appropriate thing to be."[R415].
Cluster 2 talks about high developer capabilities:
"exceptional people"[R394] or
"We get the good people to design the essence, so that everybody else can build the peripheral stuff that is not as critical. Which means that the people that are average can contribute as much as the people that are great."[R395].
Cluster 2 *further* falls apart into comments regarding the crucial role of the top-talented developers (as in the quotations above) and other comments assuming that talent is *generally* high (as in the [R488] quotation at the top). Likewise for most other concepts. □

EVIDENCE 4B: Futhermore, it is nearly impossible to make these concepts orthogonal and independent of each other. For instance the concept behind the [R395] *developer* quotation above is intertwined with the *abstraction* concept; *good-enoughness* can only be understood via *users' flexibility*; the *effort/investment* talks about a style of *quality assurance*; and so on. □

So now we know that perceived SE success factors are diverse. But what about the validity of the specific factors offered by the respondents? Investigating the evidence offered by the respondents led to

13. "appropriate" does not imply well-defined, fully orderly, etc.

14. Note that "success" here is just an abbreviation for non-breakdown.

15. practitioner industrial respondent

Insight 5: Software engineering is so far not an evidence-based discipline.

I will make three arguments based on the kinds of evidence offered in the interviews[16] and a fourth based on a comparison with medicine.

EVIDENCE 5A: Many respondents did not offer any evidence at all. This occurred in two forms: In form 1, the factor would simply be stated, period. Form 2 was less obvious: Mentioned factors were often accompanied by examples of all kinds, from vague hints to phenomena through to specific titles of research articles. There were 65 such examples overall throughout the study, many of those not single examples but a short salvo of two to four related ones. The roles of these examples where split about half-and-half into corroboration of a statement on the one hand (see Evidence 5b and 5c for a discussion of these) versus mere clarification on the other[17] – the latter not constituting even an attempt at providing evidence. □

EVIDENCE 5B: In some cases, the example offered was highly underspecific, coming from a non-software domain, which suggests SE evidence is less readily available: "And when those edge cases happen is when we find things collapsing. The Shuttle breaking apart"[R416]. Here, "edge cases" relates to software logic but the example given does not. □

EVIDENCE 5C: Most evidence offered was informal, most often referring to certain categories of software, less often to specific software systems, some non-software domain, personal experience, or 2018 ICSE keynote talks. Respondents made only 11 references to specific research works and only about half of those were used to corroborate a claimed success factor. □

EVIDENCE 5D: Contrast this with the situation in a more strongly evidence-oriented discipline: In medicine, there is the Cochrane Collaboration, an open consortium of researchers from over 130 countries who prepare systematic reviews and meta-analyses of the evidence provided in the medical literature. On their website [36], one can find 391 such reviews for the search term "diabetes" alone; works that meta-evaluate treatments for diabetes or diabetes as a complicating factor in the treatment of other diseases. A medical researcher would presumably refer to some of this evidence in a context where it is relevant. □

So there is plenty of evidence that we do not make much use of evidence.

Reviewing together the examples offered per respondent finds another disturbing pattern which leads to

Insight 6: We lack a shared taxonomy of software engineering contexts.

I will explain this based on how respondents discriminated contexts when they explained their view of success factors.

EVIDENCE 6A: Software engineering settings or contexts are diverse in many respects, such as team size, development durations, product and release models, technology used, degrees of reuse, development process models, quality assurance methods and intensities, software architectures, and many more. Most key success factors should not be expected to apply to all of these contexts alike.

And yet, some respondents made no context discriminations at all. Most did, but their discriminations are mostly only binary and mostly along the same dimension: low-criticality situations ("apps"[Several]) versus high-criticality situations ("airplanes"[Several]). Cases in between, although relevant, are rarely talked about explicitly. The context characterization tends to be stereotypical and those stereotypes are not accurate.

In particular, most respondents appeared to consider the techniques applied in high-criticality settings to be of high-tech type, whereas those few respondents who have actually seen such contexts characterize them as mostly low-tech, as we already saw above:
"brute force"[R430] or
"they check and recheck and recheck and recheck and recheck."[R436] or
"And there was a sufficient amount of paranoia. Which you had to have, because you knew that if your software was broken, things were going to blow up, people were going to die. I don't know how much that actually helped us[18], though. I know we had bugs in that software."[R415]

Besides criticality levels, a number of other dimensions were used to discriminate contexts in a manner relevant for the applicability of key success factors, but all of those only by one or a few respondents. The two that appear most relevant are when clear expectations about the software's behavior disappear ("I'm very worried as the world embraces machine learning"[R472]) or when humans can no longer compensate software misbehavior ("Where the transition, the problems, will happen is as automation increases. There won't be that human backstop that's there to do that last-minute adapation. And I do worry a little bit about that."[R434]) □

Summing up, the success factor statements made by the respondents tended to be overly general, presumably because we have no standard vocabulary in our community for thinking and talking about reasonably sized subsectors of our field.

8 LIMITATIONS

Tracy [37] suggests a set of eight method-independent quality criteria for qualitative studies: (a) worthy topic, (b) rich rigor, (c) sincerity, (d) credibility, (e) resonance, (f) significant contribution, (g) ethics, and (h) meaningful coherence, each with several facets [37, Table 1]. They are useful for study design by researchers and for quality assessment by readers.

A credible author self-assessment with these criteria would require multiple pages, but in short I see the biggests strengths of the present work in *(a) worthy topic* (facets: relevant, significant, interesting) by the generality of the statements made and the research directions suggested in Section 10 and in *(h) meaningful coherence* (facets: achieves stated goal, uses methods that fit the goal, meaningfully interconnects its pieces) by a gapless train of thought and careful argumentative triangulation that considers multiple possible views.

16. Please keep in mind Section 3.

17. I tried to find quotations to illustrate the difference, but they all would require so much context from the interview that they are not practical to present.

18. practitioner industrial respondent

The biggest weakness is in *(b) rich rigor* (facets: appropriate theoretical constructs, time in the field, samples, contexts, analysis processes) for primarily the following reasons:

- Analysis processes: The deduction rules from Section 3 in step 3 rely on inference from my respondents' collective interview behavior to the community's knowledge: If some knowledge X was not used by any of the respondents although it would have been useful for their argumentation, I conclude that the community overall is missing that knowledge. This conclusion may be invalid if the format of the interviews provided insufficient trigger for the respondents to show their knowledge of X. It is also invalid if, due to the spontaneous character of the interview, the knowledge of X was *momentarily* unavailable to them – but if they had had a few days to dig through the literature or simply think about what they know, they could have come up with it.
- Samples: More junior researchers might have added views or evidence not currently present in the interview data; a few senior respondents I would have liked to have are also not present in the data. Their presence might have shifted a few of the emphases with respect to concepts that had only some or few mentions.
- Analysis processes: The reduced presentation of the chain of evidence as described in Section 2.12 limits the reader's possibilities for checking the analysis underway. This decision was made to make the article readable, as a more rigorous presentation would have meant to (1) litter the article with many more distracting concept names and lengthy concept definitions, (2) do so even for some low-relevance concepts, (3) and even for some first-generation open codes that do not fit into the article's train-of-thought at all – which is why they were replaced by the second generation.
  Instead, I hope there is sufficient credibility of the presentation for most of my readers and refer the others to the raw data package for further detail.

Of these, the second and third are likely minor, but the first is a serious threat to the validity of the findings.

9 Related Work

9.1 Relating to knowledge gaps

It appears that newer SE articles talk (in their final section) about knowledge gaps less often than it was formerly the case:

Evidence 7a: Out of a random sample of 10 articles from ICSE 1997, **5** of them describe knowledge gaps of sorts [38]–[42], **3** at least speak of work remaining to be done [43]–[45], and only **2** do neither [46], [47].

For ICSE 2018, the corresponding numbers are **1** time yes [48], **4** times to-do [49]–[52], and **5** times nothing [32], [53]–[56]. The difference from 1997 to 2018 is statistically significant (Fisher's exact test, $p < 0.01$; one-sided[19] asymptotic with-ties Mann-Whitney test, $p < 0.032$). □

I interpret this as an increased reluctance to talk about what we do not know. This is at or near the leaves of the SETT; what about higher up?

Systematic Literature Reviews sometimes conclude there are large gaps in the knowledge about the respective topic area. For instance in their SLR on software measurement and process improvement, Unterkalmsteiner et al. [57] conclude *"Considering that confounding factors are rarely discussed (19 out of 148 studies [...]), the accuracy of the evaluation results can be questioned"*, yet *"no good conceptual model or framework for such a discussion is currently available."* And Hannay et al. [10], after their quantitative meta-analysis of pair programming efficiency, conclude that knowledge about more complex, qualitative factors is low: *"Only by understanding what makes pairs work, and what makes pairs less efficient, can steps be taken to provide beneficial conditions for work and to avoid detrimental conditions"*. So much for medium levels of the SETT; what about still higher up?

This brings us back to the discussion from Section 1.1. More pertinent work, if it exists, is difficult to find, because all obvious search terms for it are impractically broad. Several of the sources in the following subsection *could* be pertinent (and a few are, to a small degree), but they all focus on knowledge, not knowledge gaps.

9.2 Relating to the interview content

Is there literature that answers QQ? The literature most related to QQ is that on critical success factors. There are such works for domains only partly related to SE, such as general project management [58] or information systems [59]. Within SE, there are works for subdomains, such as agile methods [60], or using specific perspectives, such as CMMI [61] or project management failures [62].

If we count out textbooks, there appear to be only few works with a global perspective. Hoare [63] asks *"How did software get so reliable without proof?"*. This looks pertinent, but turns out to be a position paper with no evidence and not even references. The Handbook of Endres and Rombach [64] is focused on references to evidence, but is unopinionated with respect to discriminating key factors from less central ones. The latter is also true for the SWEBOK Guide [1], which furthermore does not emphasize evidence.

The best match appears to be Johnson et al. [65], which asked a pre-structured form of QQ in 2013: They derive 28 SE success factors and ask each of 60 ICSE participants to arrange his or her personal top-10 of those factors into a boxes-and-arrows diagram of how they impact "software engineering success". The authors cluster these diagrams into three recurring types and point out that finding multiple types implies a lack of consensus in our field. SE context discriminations are actively *suppressed* by the study design [65, Section 2.1]. Also, the factors mentioned are at least as vague as in the present study, but this is not discussed at all.

With respect to SE community knowledge gaps, all four sources mention that gaps exist, but none of them work out

19. one-sided because I was surprised by the low values in 2018 and only then decided to look at 1997, firmly expecting to find higher ones.

which gaps appear to have key roles for the state of our knowledge.

A diverse set of literature was suggested by my respondents (some of it after the interview when I asked by email), e.g. about S1-not-as-high-as-one-might-think complexity [66]–[68], about the possibility of catastrophes due to our limited understanding [69, regarding S1, S2, S3, QQ], about Lehman's astonishingly fresh and comprehensive laws of software evolution [70, regarding S1, S2, S3, QQ], about good-enoughness [71, regarding S2, QQ], about the role of developers as people and as knowledge carriers beside and beyond the program code [72, regarding S1, QQ], about research with industrial relevance [73, regarding IA], or about the astonishing amount of knowledge and collaboration required for producing something as simple as a pencil [74, regarding QQ].

Also, Dijkstra's Turing Lecture [75], when read from an S1/S3/QQ/IA perspective, is not only as keen as we would expect, but also surprisingly fresh. What's missing is mostly the social dimension, on both the development and the usage side.

10 Conclusions and Further Work

10.1 Insights

We (the SE research community) are concerned a lot with complexity, but hardly understand how it comes to be (see Section 4). We are aware that perfection is not typically to be achieved, but barely understand how quality tradeoffs are made (see Section 5). We research or invent techniques without giving much consideration to the conditions under which they will or will not be helpful (see Section 7).

10.2 ETAT: Suggested research directions

The above points suggest a number of research directions that should presumably be emphasized more in SE research.

1. Focus on emergence (see Insight 1): Assuming the indirect complexity effects are harder to understand and handle than direct ones, we should do more research to understand where and how complexity phenomena emerge from constituent parts and events.

2. Focus on tradeoffs (see Insight 2): Assuming that good-enoughness and developer limitations have near-ubiquitous relevance for impactful SE research, we should do more research to understand where, how, and how well engineers make tradeoffs in practical SE situations.

3. Evaluate assumptions (see Insight 6): Assuming that differences between software engineering contexts are relevant, we should collect and tabulate all the assumptions, tacit or explicit, that are used for justifying and scoping SE research in order to determine which different contexts might be relevant. These assumptions should then be evaluated: When are they true? When are they not-so-true? To which degree?

4. Create taxonomies (see Insight 5): Based on these results we could then define community-wide terminology that allows to state assumptions and claims more precisely than we do today, in order to become a properly evidence-based discipline. Such terminology would define taxonomies of the different sources and kinds of complexity (from emergence analysis), the different natures of engineering decisions (from tradeoffs analysis), and the different contexts in which we attempt to produce software (from assumptions analysis and field work).

For refering to these goals together, I suggest we call them **ETAT** goals (acronym of emergence, tradeoffs, assumptions, taxonomies; also French for *state* or *constitution* or various other things).

10.3 Next step

TSE reviewers of this work found the problem with the deduction rule (as discussed in Section 8) so big that they insisted the insights relying on the rule need to be independently validated.

So a survey asking about those statements and about possible counter-examples of work providing the respective knowledge is the next step in this research.

Acknowledgments

I dearly thank my interviewees for their time and explanations. I thank Franz Zieris and Kelvin Glaß for sharing their views as pilot respondents. I am indebted to Franz Zieris for his excellent contribution as advocatus diaboli.

References

[1] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge (SWEBoK V3.0)*. IEEE Computer Society, 2014.

[2] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," in *Proc. of the 26th Int'l Conf. on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 273–281.

[3] B. A. Kitchenham, "Procedures for undertaking systematic reviews," Computer Science Department, Keele University, ISSN:1353-7776, Tech. Rep. SE-0401, 2004. [Online]. Available: http://www.academia.edu/download/35088954/kitchenham.pdf

[4] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Software Engineering Group , School of Computer Science and Mathematics, Keele University, Tech. Rep. EBSE-2007-01, 2007. [Online]. Available: http://www.robertfeldt.net/advice/kitchenham_2007_systematic_reviews_report_updated.pdf

[5] B. A. Kitchenham, R. Pretorius, D. Budgen, P. Brereton, M. Turner, M. Niazi, and S. G. Linkman, "Systematic literature reviews in software engineering - A tertiary study," *Information & Software Technology*, vol. 52, no. 8, pp. 792–805, 2010.

[6] F. Q. B. da Silva, A. L. de Medeiros Santos, S. Soares, A. C. C. França, C. V. F. Monteiro, and F. F. Maciel, "Six years of systematic literature reviews in software engineering: An updated tertiary study," *Information & Software Technology*, vol. 53, no. 9, pp. 899–913, 2011.

[7] D. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study," *Information & Software Technology*, vol. 53, no. 5, pp. 440–455, 2011.

[8] S. Imtiaz, M. Bano, N. Ikram, and M. Niazi, "A tertiary study: experiences of conducting systematic literature reviews in software engineering," in *17th International Conference on Evaluation and Assessment in Software Engineering, EASE '13, Porto de Galinhas, Brazil, April 14-16, 2013*, F. Q. B. da Silva, N. J. Juzgado, and G. H. Travassos, Eds. ACM, 2013, pp. 177–182.

[9] G. K. Hanssen, D. Smite, and N. B. Moe, "Signs of agile trends in global software engineering research: A tertiary study," in *6th IEEE International Conference on Global Software Engineering, ICGSE 2011, Workshop Proceedings, Helsinki, Finland, August 15-18, 2011*. IEEE Computer Society, 2011, pp. 17–23.

[10] J. E. Hannay, T. Dybå, E. Arisholm, and D. I. Sjøberg, "The effectiveness of pair programming: A meta-analysis," *Information and Software Technology*, vol. 51, no. 7, pp. 1110–1122, 2009.

[11] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Information & Software Technology*, vol. 50, no. 9-10, pp. 860–878, 2008.

[12] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg, "A systematic review of effect size in software engineering experiments," *Information & Software Technology*, vol. 49, no. 11-12, pp. 1073–1086, 2007.

[13] E. Hasnain and T. Hall, "Investigating the role of trust in agile methods using a light weight systematic literature review," in *Agile Processes in Software Engineering and Extreme Programming, 9th International Conference, XP 2008, Limerick, Ireland, June 10-14, 2008. Proceedings*, ser. Lecture Notes in Business Information Processing, P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan, and X. Wang, Eds., vol. 9. Springer, 2008, pp. 204–207. [Online]. Available: https://doi.org/10.1007/978-3-540-68255-4_22

[14] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information & Software Technology*, vol. 50, no. 9-10, pp. 833–859, 2008.

[15] S. Jalali and C. Wohlin, "Agile practices in global software engineering - A systematic map," in *5th IEEE International Conference on Global Software Engineering, ICGSE 2010, Princeton, NJ, USA, 23-26 August, 2010*. IEEE Computer Society, 2010, pp. 45–54.

[16] A. Ramirez, J. R. Romero, and C. Simons, "A systematic review of interaction in search-based software engineering," *IEEE Trans. on Software Engineering*, 2019, to appear.

[17] Z. Zakaria, R. B. Atan, A. A. A. Ghani, and N. F. M. Sani, "Unit testing approaches for BPEL: A systematic review," in *16th Asia-Pacific Software Engineering Conference, APSEC 2009, 1-3 December 2009, Batu Ferringhi, Penang, Malaysia*, S. Sulaiman and N. M. M. Noor, Eds. IEEE Computer Society, 2009, pp. 316–322.

[18] K. Charmaz, "Constructionism and the Grounded Theory method," in *Handbook of Constructionist Research*, J. Holstein and J. Gubrium, Eds. The Guilford Press, 2008, pp. 397–412.

[19] A. L. Strauss and J. M. Corbin, *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE, 1990.

[20] B. G. Glaser, *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Mill Valley, CA: Sociology Press, 1978.

[21] K. Charmaz, *Constructing grounded theory: A practical guide through qualitative analysis*. London: Sage, 2006.

[22] S. Salinger and L. Prechelt, *Understanding Pair Programming: The Base Layer*. BoD, Norderstedt, Germany, 2013, 978-3-7322-8193-0.

[23] R. K. Yin, *Case Study Research: Design and Methods*. Sage, 2003.

[24] G. E. Mobus and M. C. Kalton, *Principles of Systems Science*, ser. Understanding complex systems. Springer, 2015, iSBN 978-1493919192.

[25] Wikipedia, "Complex system," https://en.wikipedia.org/wiki/Complex_system, July 2018.

[26] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," in *Proc. 40th Int'l. Conf. on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 48–58. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180218

[27] I. Hadar, T. Hasson, O. Ayalon, E. Toch, M. Birnhack, S. Sherman, and A. Balissa, "Privacy by designers: software developers' privacy mindset," *Empirical Software Engineering*, vol. 23, no. 1, pp. 259–289, 2017.

[28] A. Head, C. Sadowski, E. R. Murphy-Hill, and A. Knight, "When not to comment: questions and tradeoffs with API documentation for C++ projects," in *Proc. 40th Int'l. Conf. on Software Engineeering*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018.

[29] N. Ramasubbu and C. F. Kemerer, "Integrating technical debt management and software quality management processes: a framework and field tests," *IEEE Transactions on Software Engineering*, 2019, to appear. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2017.2774832

[30] I. Rehman, M. Mirakhorli, M. Nagappan, A. A. Uulu, and M. Thornton, "Roles and impacts of hands-on software architects in five industrial case studies," in *Proc. 40th Int'l. Conf. on Software Engineering*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018.

[31] M. Scavuzzo, E. Di Nitto, and D. Ardagna, "Experiences and challenges in building a data intensive system for data migration." *Empirical Software Engineering*, vol. 23, no. 1, pp. 52–86, 2017.

[32] M. Beller, N. Spruit, D. Spinellis, and A. Zaidman, "On the dichotomy of debugging behavior among programmers," in *Proc. 40th Int'l. Conf. on Software Engineering*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 572–583.

[33] D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, ""Was my contribution fairly reviewed?": a framework to study the perception of fairness in modern code reviews," in *Proc. 40th Int'l. Conf. on Software Engineering*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 523–534.

[34] J. Krüger, J. Wiemann, W. Fenske, G. Saake, and T. Leich, "Do you remember this source code?" in *Proc. 40th Int'l. Conf. on Software Engineering*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 764–775.

[35] S. Frey, A. Rashid, P. Anthonysamy, M. Pinto-Albuquerque, and S. A. Naqvi, "The good, the bad and the ugly: a study of security decisions in a cyber-physical systems game," *IEEE Transactions on Software Engineering*, 2019, to appear.

[36] T. Cochrane Collaboration, "Cochrane. trusted evidence. informed decisions. better health." https://www.cochrane.org, accessed 2018-07-23. [Online]. Available: https://www.cochrane.org/

[37] S. J. Tracy, "Qualitative quality: Eight "big-tent" criteria for excellent qualitative research," *Qualitative Inquiry*, vol. 16, no. 10, pp. 837–851, 2010.

[38] G. Bernot, L. Bouaziz, and P. L. Gall, "A theory of probabilistic functional testing," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 216–226.

[39] J. Jacquot and D. Quesnot, "Early specification of user-interfaces: Toward a formal approach," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 150–160.

[40] S. Kusumoto, O. Mizuno, T. Kikuno, Y. Hirayama, Y. Takagi, and K. Sakamoto, "A new software project simulator based on generalized stochastic petri-net," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 293–302.

[41] C. Lindig and G. Snelting, "Assessing modular structure of legacy code based on mathematical concept analysis," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 349–359.

[42] C. B. Seaman and V. R. Basili, "An empirical study of communication in code inspections," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 96–106.

[43] L. C. Briand, P. T. Devanbu, and W. L. Melo, "An investigation into coupling measures for C++," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 412–421.

[44] J. D. Reese and N. G. Leveson, "Software deviation analysis," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 250–260.

[45] E. Y. Wang, H. A. Richter, and B. H. C. Cheng, "Formalizing and integrating the dynamic model within OMT," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 45–55.

[46] G. Froehlich, H. J. Hoover, L. Liu, and P. G. Sorenson, "Hooking into object-oriented application frameworks," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 491–501.

[47] J. M. Verner and N. Cerpa, "The effect of department size on developer attitudes to prototyping," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston,*

Massachusetts, USA, May 17-23, 1997., W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 445–455.
[48] X. Huang, H. Zhang, X. Zhou, M. A. Babar, and S. Yang, "Synthesizing qualitative research in software engineering: a critical review," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1207–1218.
[49] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 665–676.
[50] X. Wang, J. Sun, Z. Chen, P. Zhang, J. Wang, and Y. Lin, "Towards optimal concolic testing," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 291–302.
[51] P. Wang, J. Svajlenko, Y. Wu, Y. Xu, and C. K. Roy, "Ccaligner: a token based large-gap clone detector," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1066–1077.
[52] J. Yang, P. Subramaniam, S. Lu, C. Yan, and A. Cheung, "How *not* to structure your database-backed web applications: a study of performance bugs in the wild," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 800–810.
[53] J. Hua, M. Zhang, K. Wang, and S. Khurshid, "Towards practical program repair with on-demand candidate generation," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 12–23.
[54] L. Lambers, D. Strüber, G. Taentzer, K. Born, and J. Huebert, "Multi-granular conflict and dependency analysis in software engineering based on graph transformation," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 716–727.
[55] M. Wen, J. Chen, R. Wu, D. Hao, and S. Cheung, "Context-aware patch generation for better automated program repair," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1–11.
[56] Y. Zhao, M. S. Laser, Y. Lyu, and N. Medvidovic, "Leveraging program analysis to reduce user-perceived latency in mobile applications," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 176–186.
[57] M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement - A systematic literature review," *IEEE Trans. Software Eng.*, vol. 38, no. 2, pp. 398–424, 2012.
[58] T. Cooke-Davies, "The 'real' success factors on projects," *International Journal of Project Management*, vol. 20, no. 3, pp. 185–190, 2002.
[59] E. W. T. Ngai, C. C. H. Law, and F. K. T. Wat, "Examining the critical success factors in the adoption of enterprise resource planning," *Computers in Industry*, vol. 59, no. 6, pp. 548–564, 2008. [Online]. Available: https://doi.org/10.1016/j.compind.2007.12.001
[60] T. Chow and D.-B. Cao, "A survey study of critical success factors in agile software projects," *Journal of Systems and Software*, vol. 81, no. 6, pp. 961–971, 2008. [Online]. Available: https://doi.org/10.1016/j.jss.2007.08.020
[61] D. R. Goldenson and D. L. Gibson, "Demonstrating the impact and benefits of CMMI: An update and preliminary results," Carnegie Mellon Software Engineering Institute, Special Report CMU/SEI-2003-SR-009, 2003.
[62] K. Ewusi-Mensah, *Software Development Failures: Anatomy of Abandoned Projects*. MIT Press, 2003.
[63] C. A. R. Hoare, "How did software get so reliable without proof?" in *International Symposium of Formal Methods Europe*. Springer, 1996, pp. 1–17.
[64] A. Endres and D. Rombach, *A handbook of software and systems engineering: Empirical observations, laws, and theories*. Pearson Education, 2003.
[65] P. Johnson, P. Ralph, M. Ekstedt, and Iaaktudy, "Consensus in software engineering: A cognitive mapping study," *CoRR*, vol. abs/1802.06319, 2018. [Online]. Available: http://arxiv.org/abs/1802.06319
[66] M. Allamanis, E. T. Barr, P. T. Devanbu, and C. A. Sutton, "A survey of machine learning for big code and naturalness," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 81:1–81:37, 2018.
[67] T. Menzies, D. Owen, and J. Richardson, "The strangest thing about software," *IEEE Computer*, vol. 40, no. 1, pp. 54–60, 2007.
[68] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic improvement of software: A comprehensive survey," *IEEE Trans. Evolutionary Computation*, vol. 22, no. 3, pp. 415–432, 2018.
[69] N. G. Leveson, "High-pressure steam engines and computer software," in *Proc. 14th Int'l. Conf. on Software Engineering, Melbourne.*, 1992, pp. 2–14.
[70] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *Journal of Systems and Software*, vol. 1, pp. 213–221, 1980.
[71] M. Shaw, "Everyday dependability for everyday needs," in *Supplemental Proc. 13th Int'l Symp. on Software Reliability Engineering*. IEEE Computer Society, 2002, pp. 7–11.
[72] P. Naur, "Programming as theory building (1985)," in *Computing: a human activity*, P. Naur, Ed. ACM, 1992.
[73] V. R. Basili, L. C. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh, "Software engineering research and industry: A symbiotic relationship to foster impact," *IEEE Software*, vol. 35, no. 5, pp. 44–49, 2018.
[74] L. E. Read, "I, pencil," *The Freeman*, 1958, reprinted by the Foundation for Economic Education. [Online]. Available: https://fee.org/resources/i-pencil/
[75] E. W. Dijkstra, "The humble programmer," *Communications of the ACM*, vol. 15, no. 10, pp. 859–866, 1972. [Online]. Available: https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

Lutz Prechelt Lutz Prechelt received a PhD from the University of Karlsruhe for work that combined machine learning and compiler construction for parallel machines. He then moved to empirical software engineering and performed a number of controlled experiments before spending three years in industry as a manager. He is now full professor for software engineering at Freie Universität Berlin. His research interests concern the human factor in the software development process, asking mostly exploratory research questions and addressing them with qualitative methods. Additional research interests concern research methods and the health of the research system. He is the founder of the Forum for Negative Results and of Review Quality Collector.