

The Choice of Code Review Process: A Survey on the State of the Practice

Tobias Baum^(✉) , Hendrik Leßmann, and Kurt Schneider

FG Software Engineering, Leibniz Universität Hannover, Hannover, Germany
{tobias.baum,hendrik.lessmann,kurt.schneider}@inf.uni-hannover.de

Abstract. Code review has been known to be an effective quality assurance technique for decades. In the last years, industrial code review practices were observed to converge towards “change-based/modern code review”, but with a lot of variation in the details of the processes. Recent research also proposed hypotheses on factors that influence the choice of process. However, all current research in this area is based on small and largely non-random samples of cases. Therefore, we set out to assess the current state of the practice and to test some of these hypotheses with a survey among commercial software development teams. We received responses from 240 teams. They support many of the stated hypotheses, e.g., that change-based code review is the dominating style of code review in the industry, and that teams doing change-based code review have a lower risk that review use fades away. However, other hypotheses could not be confirmed, mainly that the balance of effects a team tries to reach with code reviews acts as a mediator in determining the details of the review process. Apart from these findings, we contribute the survey data set as a foundation for future research.

Keywords: Code reviews · Code inspections and walkthroughs · Change-based code review · Modern code review · Empirical software engineering

1 Introduction

Code review is a well-established method of software quality assurance. Several researchers noted that, in recent years, change-based review¹ has become the dominant style of code review in practice [8, 24]. The main characteristic of change-based review is the use of code changes performed in a unit of work, e.g., a user story, to determine the scope of the review. This is often combined with the replacement of management intervention through conventions or rules for many decisions [8], making a review planning phase [22] largely obsolete.

However, recent quantitative information on the use of different review practices in the industry is largely missing. Furthermore, it is important for

¹ Also called “modern code review”, “differential code review” or “patch review” in other publications.

researchers trying to improve development processes to know what mechanisms influence process choices in practice. Although previous research has put forward hypotheses on the benefits of change-based code review and reasons for the choice of a review process, these have not yet been verified on a larger sample. Consequently, the purpose of this article is to test hypotheses on industrial code review use and the use of change-based code review in particular and to provide further empirical data on current review practices.

Specifically, we selected three research questions based on hypotheses put forward in our and others' earlier research, all relating to how review processes are shaped in the industry and why they are shaped that way:

- RQ1** How prevalent is change-based review in the industry? (based on [8,24])
- RQ2** Does the chance that code review remains in use increase if code review is embedded into the process (and its supporting tools) so that it does not require a conscious decision to do a review? (based on [9])
- RQ3** Are the intended and acceptable levels of review effects a mediator in determining the code review process? (based on [9])

To answer these questions, we conducted an online survey among commercial software development teams. We will further concretize the questions and derive testable sub-hypotheses in Sects. 4.1, 4.2 and 4.3. In addition to answering the research questions, we give some descriptive statistics from the survey and we note some findings on review process characteristics and the use of reading techniques in Sects. 4.4 and 4.5. A secondary contribution of the study is the questionnaire instrument used to assess a team's review process, which we publish along with the full survey data for reuse in future research [7].

2 Related Work

As described in the previous section, the hypotheses we test in the current article stem from earlier research, mainly the summary of converging code review practices by Rigby and Bird [24] and the Grounded Theory study on code review use by parts of the current article's authors and others (Baum et al. [8,9]).

The most recent academic survey on the state of review practices we are aware of was published by Ciolkowski, Laitenberger, and Biffi in 2003 [13], with partial results also published in a technical report [21]. This survey targeted not only code review, but also reviews in other lifecycle phases. Its authors found a share of 28% of the 226 respondents using code reviews. We will discuss some similarities and differences between their and our survey in Sect. 6.

Bacchelli and Bird [2] surveyed expectations regarding code review at Microsoft and found a set of intended effects similar to the ones we are using. A recent survey on software testing practices by Winter et al. [30] briefly touches upon reviews and notes that 52% of the respondents often or always perform reviews for source code.

Looking beyond the academic literature, there are some more recent surveys that contain information on code review practices. A whitepaper written in 2010 by Forrester Consulting [15] for Klocwork, a company selling a code review tool, notes that 25% of the 159 survey respondents use a review process that we would describe as “regular, change-based code review” [8].

A survey performed in 2015 by Smartbear [26], another company selling code review software, contains information on code review practices and perceptions on code quality from about 600 respondents. Like the Forrester study, it contains very little information on the sampling method and possible biases. It states that 63% of their respondents are doing tool-based code review.

Compared to the small number of surveys, there is a lot more qualitative and case study research on code review practices and we can just name a few here. Baker [3] gave an early description of a change-based code review process in the industry and Bernhart et al. [11] describe its use (under the term “continuous differential code review”) in airport operations software. Other small-scale studies of code review and inspection practices in the industry have been performed by Harjumaa et al. [17] and by Kollanus and Koskinen [19].

A survey by Bosu and Carver studied the impact of code review on peer impression in open source projects [12]. Peer review practices in open source software development have been studied intensively in the last decade, with further contributions for example by Asundi and Jayant [1], Rigby and Storey [25], Wang et al. [29], Thongtanunam et al. [28] and Baysal et al. [10].

3 Methodology

Our goal was to reach out to a large number of commercial software development teams. We used (online) survey research as our main vehicle. In the following, we describe details of the planning, execution, and analysis of the survey.

3.1 Participant Selection

Our research questions deal with the code review practices and context of commercial software development teams. Consequently, our target population consists of all commercial software development teams.

As there is no repository of all software development teams, a controlled random sampling of participants was not possible. Instead, we relied on a number of communication channels to reach possible participants: We directly contacted 32 people belonging to the target population from our personal networks. We further asked 23 people outside the target population from our networks to advertise the survey. We posted the survey invitation to several online communities, on Twitter, Xing, and Facebook; and also advertised the survey at a German software engineering conference. Finally, we posted the invitation on some mailing lists. Probably the most important single channel was a post on the mailing list of the German software craftsmanship communities (“Softwerkskammer”), reaching out to roughly 1400 people. When selecting channels, we took care

to avoid introducing bias on the type of review process used. Specifically, we decided against sampling GitHub users, and we turned down an offer to spread our invitation to a mailing list of former participants of a series of review courses. In Sect. 5, we discuss the remaining risk of sampling bias.

Since we were not able to exactly control who was answering the survey, we included a filter question at the start of the survey and excluded participants not working in a commercial software development team. Our intended granularity of sampling was teams. As our survey was conducted anonymously, we could not tell whether two respondents come from the same or different teams. We told survey participants that we only want one response per team and asked them to only forward the invitation to people in other teams or companies. When inviting participants directly, we took care to only invite one person per company. Nevertheless, there is a risk that the sample includes several respondents from the same team.

3.2 Questionnaire Creation and Pilot Tests

Most parts of the survey were created based on existing qualitative empirical research, mainly the classification scheme for change-based code review processes by Baum et al. [8] and the collection of contextual factors influencing review process choices by the same authors [9].

The process of survey creation followed established guidelines [18, 27]. To ease answering and analyzing the survey, we mainly used multiple choice and numerical questions. Many questions contained an “Other” option to allow participants to specify missing options in free-text. The instrument was self-contained and it included all relevant information, for example by giving a definition of code review (from [8]) when asking for the use of code review in the team.

Based on our sampling strategy, we expected the main share of participants to come from Germany and a mix of other countries for the remaining share. Therefore, we decided to create a German as well as an English version of the survey, following the rationale that it is better to have a consistent, pre-tested translation instead of demanding translation effort from each participant.

Following guidelines for survey research [27], we tried to reuse questions from existing surveys, but only a limited number of questions from the first version of the HELENA survey [20] could be reused after some adjustments.

To ensure that the survey questions were comprehensible and valid with respect to the study constructs, we iteratively tested and refined the questionnaire. Initial testing of the research and survey questions was performed among the authors of this article. We used a checklist distilled from existing guidelines to check each survey question. This was followed by 6 rounds of pre-tests, 4 of these with members of the target population and 2 with members of our lab. During each pre-test, a participant completed the survey, followed by a discussion about possible problems and misunderstandings. For two of these pre-tests, we had detailed knowledge about the process used by the team and could, therefore, compare answers to our expectations. The final survey also allowed the participants to enter feedback on the survey, which we checked for possible problems.

The review process classification scheme [8] we used as a foundation consists of 20 process facets, and we identified 16 potentially relevant contextual factors in our previous research [9]. It became evident early during questionnaire creation that even if we restricted the survey to these two groups of questions, it would become too long for the intended audience. Therefore, we limited the questionnaire to a subset of the contextual factors and split the remaining questions into a main part and an optional extension part. Answering the main part took around 15 min and answering the extension part additional 8 min in our pre-tests. The exact number of questions differed depending on the answers of a respondent, e.g., it was shorter for teams that have never used code reviews.

3.3 Data Collection and Instrument

We started data collection on February 22nd, 2017, and closed the survey on March 20th, 2017. Invitations were sent out gradually during the first weeks. The questionnaire was implemented using LimeSurvey, hosted by our university.

The survey questions can be roughly classified into four groups: (1) Demographics or filter questions (e.g., on the country, role of the participant or the use of reviews), (2) questions on the context of the review process (e.g., product, development process, team characteristics, . . .) (3) questions on the used review process (based on [8]) and (4) ranking questions to assess the relative importance of intended and unintended review effects. The full instrument can be found in our online material [7]. Most parts of the survey were confirmatory or descriptive, but it also contained some exploratory parts, mainly on the non-use of reviews. In the current article, we focus on the former.

We offered respondents the chance to leave an email address if they were interested in the results of the survey. All participants that did this have been informed about preliminary results some weeks after the survey closed.

3.4 Data Analysis

Our data analysis constitutes a mix of descriptive and inferential statistics. We will describe the detailed analyses for the research questions in the respective subsections of Sect. 4.

Multiple-choice questions that contained an “other” option with free-text answers were coded for analysis. The free-text answers were either converted to new categories or classified as belonging to one of the existing categories.

All but the filter questions were optional, to avoid forcing participants to answer. We handled the resulting missing data by “pairwise deletion” (also called “available case analysis”), i.e., we excluded participants only from those analyses where data was missing for at least one of the needed questions. Consequently, the total number of respondents taken into account differs between analyses.

Most statistical tests performed during analysis checked for a dependence between two dichotomous variables. Unless otherwise noted, these 2×2 contingency tables were checked using Fisher’s exact test and statistical significance

was tested at the 5% level. We perform Bonferroni correction when there are multiple tests for a research question (i.e., for *RQ3*), but not between research questions. When we give confidence intervals for proportions, they will be 95% confidence intervals calculated using the Clopper-Pearson method. All percentages will be presented rounded to the nearest integer.

The raw data of the survey, descriptive statistics for all questions, and the source code used for data analysis is available in the study’s online material [7].

4 Results

In total, 240 respondents from the target population answered the survey.² 130 participants went on to answer the extension part after finishing the main part. Due to our sampling method, we are not able to give a response rate, but we will describe some characteristics of the sample in the following.

The respondents are working in 19 different countries. The majority of respondents, 170 (76%), is from Germany. 33 respondents (15%) work in other European countries, 11 (5%) in Asia (including the Middle East) and 11 (5%) in Northern America.³ We distributed the survey invitation through various channels and asked the respondents how they heard about the survey. 19 respondents (10%) were invited directly by one of the researchers, 30 (16%) were indirectly invited by other people, 104 (55%) heard about the survey on a mailing list, 24 (13%) in an online forum and 13 (7%) named some other channel. When asked about their role, 154 respondents (67%) said they mainly work as a developer, 50 (22%) work as architects, 14 (6%) as managers and 11 (5%) gave other roles.

The target population of this survey is teams in commercial software development. Quite unsurprisingly, the large majority (94%, 215 teams) of the responding teams works on closed source software. The remaining share (14 teams) said their team mainly works on an open source project. The teams work in companies of vastly differing sizes, from less than 10 to more than 10,000 employees; Fig. 1 shows the detailed distribution of company sizes. 68% (148 of 217) of the participants work in collocated (as opposed to distributed) teams.

We asked teams whether they are currently using code reviews. Teams not using code reviews were subdivided further: Have they never used reviews before, and if so have they never thought about it or did they explicitly decide against review use? Or did they stop using reviews in the past, and if so was this an explicit decision or did the review use “fade away” (i.e., end without an explicit decision, just becoming less and less frequent over time)? Figure 2 shows the results: With a share of 78% (186 teams), the majority of teams is currently

² More precisely, 240 respondents answered at least the questions about being part of the target population and about their team’s review use, which were the only obligatory questions in the survey.

³ The remaining 15 did not answer this question. Unless otherwise noted, we only include respondents that answered the respective questions in our analyses; consequently, the total sum of respondents will differ between analyses.

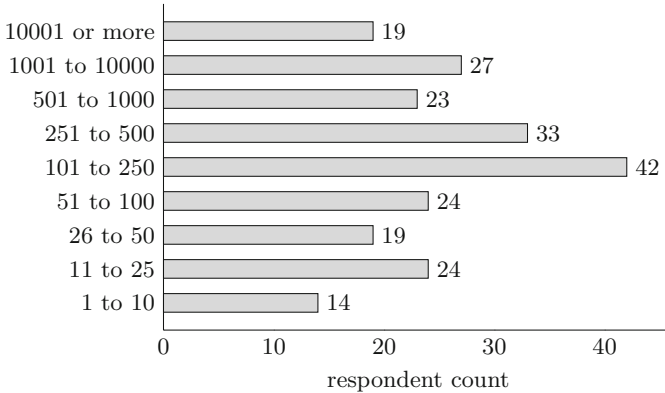


Fig. 1. Company sizes (number of employees)

using code reviews.⁴ 38 teams (16%) have never used code reviews so far, 8 of them because there was an explicit decision against their use. In 16 teams (7%), the use of code reviews ended, but in only one of those teams this was an explicit decision. For teams that currently use code reviews, we asked how much time ago they started using them. The results are shown in Fig. 3.

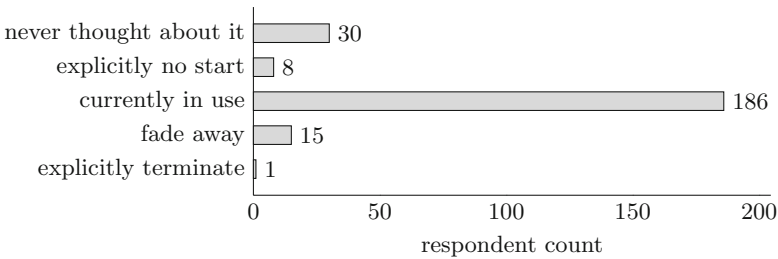


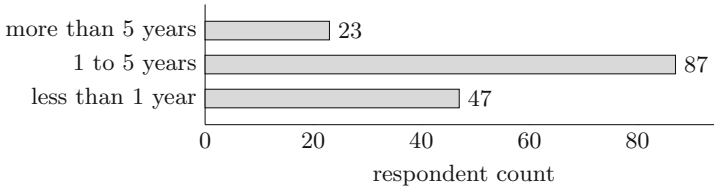
Fig. 2. Use of reviews

4.1 The Dominance of Change-Based Code Review (RQ1)

A number of recent articles postulate that code review based on code changes/patches is dominating in industrial as well as open-source practice [8, 24]. In this section, we provide quantitative empirical support for this claim (RQ1) and study the prevalence of several more specific review styles.

To answer RQ1, we asked our participants how the review scope is determined: Based on changes, based on architectural properties of the software (whole module/package/class) or in some other way (with free text for further details). With a share of 90% (146/163; confidence interval 84–94%) of the teams doing code reviews, a change-based review scope is indeed dominating.

⁴ This number is likely biased, see Sect. 5.

**Fig. 3.** Time that code review is in use**Table 1.** Frequency of use of different styles of code review

Style	Used approximation of definition using survey constructs	Frequency of use
Review based on code-changes	scope = changes	90% (146/163)
Regular, change-based code review [8]	scope = changes and trigger = rules	60% (96/160)
Contemporary code review [24]	scope = changes and publicness = pre-commit and unit-of-work \leq user-story	46% (61/133)
Pull-based software development [16]	scope = changes and trigger = rules and publicness = pre-commit and interaction = no-meeting	22% (29/134)
Approximating Inspection [14]	interaction = meeting and communication = oral + stored and temporal-arrangement = parallel and trigger = explicit	2% (3/141)

In the recent literature on code review and related work practices, there are slightly differing definitions and descriptions of sub-styles of change-based code review. Table 1 shows the frequency of use for “modern/contemporary code review” [24], “regular, change-based review” [8] and “pull-based software development” [16]. As not all of these sub-styles are concisely defined in the respective publications, the table also shows how the definitions/descriptions were approximated in terms of constructs used in the survey. We will not discuss every detail of the table, but want to note that most of the teams that do not fall under Rigby and Bird’s description of contemporary code review do so because they do not use pre-commit reviews (pre-commit: 46%, 61 teams; post-commit: 54%, 72 teams). There is only one respondent whose team uses a review scope that is larger than a user story/requirement.

We did not focus on Fagan-style Inspection [14] in our survey and therefore cannot completely tell whether a team uses a fully-fledged Inspection process

to review code. To estimate an upper bound on the number of teams doing Inspection, we combined a number of necessary conditions that we would expect to hold for those teams (see Table 1). Only 2% (3/141; confidence interval 0–6%) of the teams have a process that approximates Inspection in that way.

Because much existing research on modern/change-based code review is based on open-source development or agile teams, we also checked whether there is a difference in the use of change-based review between open-source and closed-source products and between agile and classic development processes. We did not find a statistically significant difference in either case.

4.2 Change-Based Code Review and the Fading of Review Use (RQ2)

This section mainly deals with Baum et al.’s hypothesis “*Code review is most likely to remain in use if it is embedded into the process (and its supporting tools) so that it does not require a conscious decision to do a review.*” [9]. More specifically, we tested a subset of this hypothesis:

H2 The risk that code review use fades away depends on the mechanism that is used to determine that a review shall take place: This risk is lower when rules or conventions are used instead of ad-hoc decisions.

To test this hypothesis, we compare two sub-samples: Teams currently doing code reviews, and teams where review use faded away. We also asked how it was decided whether a review should take place: By fixed rules or conventions, or ad-hoc on a case-by-case basis. For the ad-hoc triggers, we further distinguished triggering by the reviewer, the author or a manager. There was the possibility for respondents to select “other” and enter a free-text description.

Of 12 teams where review use faded away, 3 used rules or conventions and the remaining 9 used ad-hoc decisions. For the 162 teams currently doing reviews, the relation was 103 with rules/conventions compared to 59 without. Put differently, the risk to be in the “fade away” subsample increases from 2.8% with rule triggers to 13.2% with ad-hoc triggers, a risk ratio of 4.7. The exact Fisher test of the corresponding 2×2 contingency table results in a p-value of 0.01237, therefore the difference is statistically significant at the 5% level. Table 2 shows the detailed numbers for the different review triggers. An interesting side-note is that having managers trigger reviews seems to be especially prone to discontinuation.

Another possible explanation for the higher share of teams with ad-hoc triggers in the “fade away” subsample is a generation effect: Teams that introduced reviews more recently could use rule triggers more often. Therefore, we compared teams that have used reviews for less than a year with those that used them for two years or more. Of 45 teams with brief review use, 25 use rules. For teams with long review use, the share is 49 of 75. This higher share of rule-use for longer review use supports *H2* and opposes the stated generation effect.

Table 2. Review triggers vs. review continuation

Trigger	Reviews in use	Review use faded away
Manager	7	3 (30%)
Reviewer	14	2 (13%)
Author	38	4 (10%)
Rules/conventions	103	3 (3%)

4.3 Rankings of Review Effects as a Mediator in Determining the Review Process (RQ3)

In this section, we deal with another of Baum et al.’s [9] hypotheses: “*The intended and acceptable levels of review effects are a mediator in determining the code review process.*” (H3) More specifically, Baum et al. state that “Many process variants are expected to promote certain effects, and often also to impair others. . . . Consequently, the chosen review process is heavily influenced by the combination of intended effects. Some effects are seen as more important than others, while others are seen as secondary or not pursued at all. This is used to perform trade-offs while designing the review process.” [9] Intended review effects are for example “better code quality”, “finding defects” and “learning of the reviewer”. Unintended effects are “[increased] staff effort”, “increased cycle time” and “offending the author/social problems”. Following *H3*, we would expect to find that the relative ranking of review effects influences the chosen variant for some of the review process facets, that the team’s context influences the relative ranking of review effects, and that this indirect effect is in most cases stronger than the direct influence of context on review process facets.

Based on the addendum to Baum et al.’s article [6], we systematically checked each of the listed combinations of review effect and process facet.⁵ For intended review effects, none of the checked interactions were statistically significant, even at the 10% level and without Bonferroni correction. For the relative ranking of undesired effects, some of the predicted effects had p-values smaller than 0.05:

- When “increased staff effort” is most unintended this makes a “very small review scope” (i.e., more overhead due to a higher number of small reviews) less likely: Risk ratio = 2.2; $p = 0.034$. The detailed contingency table can be found in Table 3.
- When “increased staff effort” is most unintended this makes “pull or mixed reviewer to review assignment” more likely: Risk ratio = 1.6; $p = 0.037$. The detailed contingency table can be found in Table 4.
- When “increased cycle time” is most unintended this makes “review meetings” less likely: Risk ratio = 2.8; $p = 0.006$. The detailed contingency table can be found in Table 5.

⁵ I.e. we did not check every possible combination, but only those where the prior research gave reason to expect an influence.

Table 3. Contingency table: staff effort most undesired vs. small review scope

	Small review scope	Medium to large review scope (\geq Task)	Total
Increased staff effort most undesired	5	19	24
Something else most undesired	36	43	103
Total	41	62	103

Table 4. Contingency table: staff effort most undesired vs. review assignment

	Push assignment	Pull or mixed assignment	Total
Increased staff effort most undesired	12	20	32
Something else most undesired	52	34	86
Total	64	54	118

Table 5. Contingency table: increased cycle time most undesired vs. review meetings

	Meetings	No meetings	Total
Increased cycle time most undesired	6	40	46
Something else most undesired	27	46	73
Total	33	86	119

Those three interactions are also those with the highest risk ratio (i.e., effect size). Even though they have p-values smaller than 0.05, none of them is statistically significant after Bonferroni correction. A complete list of all tested interactions can be found in the study’s online material [7].

Summing up, only 3 of 30 cases give some support for the expected relationship. Therefore, there is little evidence that the intended and acceptable levels of review effects influence the code review process, except in some narrow areas. Consequently, they cannot be mediators, and we cannot support hypothesis $H3$.

4.4 Further Convergence in Review Practices?

Apart from their description of a change-based review process that we referred to in Sect. 4.1, Rigby and Bird consolidated three further convergent review practices, cited in the following. In this section, we analyze to what degree these practices can also be observed in the survey’s sample.

“Contemporary review usually involves two reviewers. However, the number of reviewers is not fixed and can vary to accommodate other factors, such as

the complexity of a change.” [24]: Our results support the finding that the usual number of reviewers is low, indeed our numbers are even lower than Rigby and Bird’s.⁶ The average usual number of reviewers in our sample is 1.57, the median is 1 reviewer. With regard to the accommodation of other factors when determining the number of reviewers, 51% of the teams (47 of 92) named at least one rule that they use to adjust the number of reviewers in certain situations. The most commonly used rule is to decrease the number of reviewers or to skip code review completely when the code change was implemented using pair programming: Such a rule is used in 36% of the teams.

“Contemporary reviewers prefers [sic] discussion and fixing code over reporting defects.” [24]: Fig. 4 shows how the surveyed teams usually interact during a review. Depending on how many of the teams discuss code during review meetings, between 55% and 81% of the teams have a review process that includes discussion of the code. Regarding fixing the code, 54% (84 of 157) of the respondents indicate that reviewers sometimes or often fix code during a review. This pragmatic attitude towards the classic boundaries of code review also shows up when 76% (69 of 91) of the respondents state that the reviewer executes the code for testing during review at least occasionally.

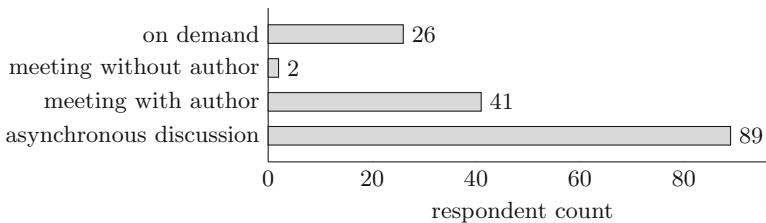


Fig. 4. Interaction during reviews

“Tool supported review provides the benefits of traceability, when compared to email based review, and can record implicit measures, when compared to traditional inspection. The rise in adoption of review tools provides an indicator of success.” [24]: In our sample, 59% of the teams (96 of 163) use at least one specialized review tool. 33% (33 of 163) use only general software development tools, like ticket system and version control system, for review. 13 respondents indicated no tool use.⁷ The ability of specialized review tools to record implicit measures might be one of their benefits, but it is seldom used in practice. Only 5% (4 of 88) of the teams systematically analyze review metrics.

⁶ The numbers are not fully comparable: Rigby and Bird looked at the actual number of reviewers in a large sample of reviews, whereas we asked our participants for the usual number of reviewers in a review.

⁷ A weakness in the used questionnaire is that there was no explicit “We do not use any tool” choice available. Therefore, the distinction between non-response and non-use of tools cannot be reliably made.

4.5 Some Notes on the Use of Reading Techniques

Research on Inspection has resulted in a number of different “reading techniques” to guide the reviewer during review [4]. We looked at the spread of some reading techniques in the survey, albeit not in much detail, and will describe the corresponding results in the following.

It is sometimes claimed that “checklist-based reading” is the prevalent reading technique in practice. Our results do not support this claim: Only 23% (22 of 94) of the respondents state that they use a checklist during reviews.

Another family of reading techniques uses different roles/perspectives to focus the reviewers (e.g., perspective-based reading [5]). 7% (6 of 90) of the respondents state that they explicitly assign distinct roles to the different reviewers. 72% (63 of 88) use neither checklists nor roles.

5 Limitations

This section discusses the addressed and unaddressed threats to validity.

The primary threat to internal validity in this study is sampling bias, given that we distributed the survey over various channels and could not control who answered. Consequently, our participants likely differ systematically from the population of all developers, and they do so not only in their geographical distribution: They are probably more interested in code reviews and/or in process improvement or software quality in general. People who introduced code reviews just recently are probably more interested in learning more about them, which could explain the high share of participants that introduced reviews less than a year ago. The tendency to have participants with a higher than average quality orientation was probably further amplified by using the software craftsmanship mailing list to advertise the survey. Due to this bias, the share of teams using code review that we observed in the survey should be regarded as an upper limit rather than as an estimate of the real proportion. Apart from this bias, we actively tried to avoid favoring certain types of code review processes in the sample.

A general problem of online surveys is that there is little control over the quality of responses. We included filter questions to check whether participants belong to the target population. We also screened free-text answers for obviously nonsensical responses. These checks, and the fact that there was no financial reward and therefore little incentive to participate without giving honest answers make us believe that this threat is under control. Another threat with long online surveys is survey fatigue. As 209 of 240 participants reached the end of the main part, there is no indication of major fatigue effects.

The survey was anonymous, and most of the questions did not touch upon sensitive topics. However, the results of some questions might be influenced by social desirability bias, e.g., by stating that the team is using reviews just because it is desirable to do. Again, this might have influenced the descriptive parts but we do not see a major influence on the confirmatory parts of the current article.

An important input for the testing of $H3$ was the ranking of intended and unintended review effects. We used LimeSurvey’s ranking widget for these questions, and we observed some usability problems with it that might have increased noise in the results or lowered the response rate.

To reduce the threat of participants misunderstanding a question, we spent approximately four months carefully designing the survey during which we performed several rounds of quality control and pre-testing. Furthermore, our questions were based on a qualitative study, which increases our confidence in their ecological validity. To further reduce threats to construct validity, we used randomization where appropriate, and we provided definitions for key terms.

Although the total sample contains 240 respondents, some of our conclusions might be affected by threats due to a small sample size. The test of $H3$ demanded the comparison of many imbalanced and therefore small sub-samples. Consequently, the power of these statistical tests was largely low.

A weakness of our method of data collection, i.e., of cross-sectional observational studies, is that they cannot be used to distinguish between correlation and causation. Therefore we cannot reliably exclude other explanations for $H2$.

The coding of free-text answers might be affected by subjectivity. For most questions, the proportion of free-text “other” answers was low. Exceptions were generally easy to code, e.g., country or review tool. An unexpectedly high number of free-text answers was given for the review trigger question: We regard “a review has to be performed for every task/story” as a special kind of rule, but many participants selected “other” instead of “rule” in this case. Details on how the free-text answers were coded can be found in the study’s online material [7].

6 Conclusion

To conclude, we relate our findings to other studies and outline future work.

Comparing our results to those of Ciolkowski et al. [13], the raw numbers indicate a large increase in the use of code reviews in the last 15 years. We already noted that the proportion observed in our survey is probably biased, but even when taking the much smaller number from Winter et al. [30] as a comparison, there was a significant increase. The systematic use of review metrics, on the other hand, seems to have decreased, as has the use of review meetings. We cannot reliably decide whether this is really due to a change in practices or due to differences between the studies.

By answering our first research question, we provided quantitative evidence that change-based review is indeed dominating in practice and that there is still a lot of variation in the details. Many researchers have begun to study and improve change-based code review, and our results should encourage them further.

By strengthening the evidence that using rules or conventions to trigger code reviews helps to keep code review use from fading away, we provide a partial explanation for the dominance observed in RQ1. As a more abstract consequence for future software engineering research, we believe this finding strengthens the case for software engineering techniques that not only work in isolation but are

also able to “survive” in the environment of a software development team. The low number of teams using perspective-based reading or a similar technique for code review could be an example for such a mismatch: There is little use in perspectives when there is only one reviewer.

Due to the low statistical power and multiple threats, the analysis of RQ3 is problematic. Assuming that our non-finding is not caused by flaws in the data collection and analysis, we see two explanations: (1) There is an effect, but we checked the wrong sub-hypotheses, or (2) the intended effects determine a team’s review process only to a small degree. The second explanation is in line with Ciolkowski, Laitenberger, and Biffi’s conclusion that many companies use reviews unsystematically [13]. It would also mean that satisficing [23] and orientation along experiences from peers and processes used by review tools are even more important than noted by Baum et al. [9]. There remains a lot of research to be done, both to find out which process variants are best in a given situation, and to find ways to bring these results into practical use.

Finally, we hope that by making the questionnaire and the survey data available [7], independent researchers can profit and build upon our efforts.

Acknowledgments. The authors would like to thank all pre-testers and all participants of the survey for the time and effort they donated. We would further like to thank Philipp Diebold and Paul Clarke for providing questions from their surveys for reuse.

References

1. Asundi, J., Jayant, R.: Patch review processes in open source software development communities: a comparative case study. In: 40th Annual Hawaii International Conference on System Sciences, HICSS 2007, p. 166c. IEEE (2007)
2. Bacchelli, A., Bird, C.: Expectations, outcomes, and challenges of modern code review. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 712–721. IEEE (2013)
3. Baker Jr., R.A.: Code reviews enhance software quality. In: Proceedings of the 19th International Conference on Software Engineering, pp. 570–571. ACM (1997)
4. Basili, V., Caldiera, G., Lanubile, F., Shull, F.: Studies on reading techniques. In: Proceedings of the Twenty-First Annual Software Engineering Workshop, vol. 96, p. 002 (1996)
5. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M.V.: The empirical investigation of perspective-based reading. *Empirical Softw. Eng.* **1**(2), 133–164 (1996)
6. Baum, T.: Detailed table with review effects (team level) and their connections to contextual factors and process variants for “factors influencing code review processes in industry” (2016). <http://dx.doi.org/10.6084/m9.figshare.5104111>
7. Baum, T., Leßmann, H., Schneider, K.: Online material for survey on code review use. <http://dx.doi.org/10.6084/m9.figshare.5104249>
8. Baum, T., Liskin, O., Niklas, K., Schneider, K.: A faceted classification scheme for change-based industrial code review processes. In: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, Vienna (2016)

9. Baum, T., Liskin, O., Niklas, K., Schneider, K.: Factors influencing code review processes in industry. In: Proceedings of the ACM SIGSOFT 24th International Symposium on the Foundations of Software Engineering. ACM, Seattle (2016)
10. Baysal, O., Kononenko, O., Holmes, R., Godfrey, M.W.: Investigating technical and non-technical factors influencing modern code review. *Empirical Softw. Eng.* **21**, 932–959 (2016)
11. Bernhart, M., Strobl, S., Mauczka, A., Grechenig, T.: Applying continuous code reviews in airport operations software. In: 2012 12th International Conference on Quality Software (QSIC), pp. 214–219. IEEE (2012)
12. Bosu, A., Carver, J.C.: Impact of peer code review on peer impression formation: a survey. In: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 133–142. IEEE (2013)
13. Ciolkowski, M., Laitenberger, O., Biffl, S.: Software reviews: the state of the practice. *IEEE Softw.* **20**(6), 46–51 (2003)
14. Fagan, M.E.: Design and code inspections to reduce errors in program development. *IBM Syst. J.* **15**(3), 182–211 (1976)
15. Forrester Research Inc.: The value and importance of code reviews, March 2010. <http://embedded-computing.com/white-papers/white-paper-value-importance-code-reviews/>. Accessed 13 June 2017
16. Gousios, G., Pinzger, M., Deursen, A.V.: An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering, pp. 345–355. ACM, Hyderabad (2014)
17. Harjumaa, L., Tervonen, I., Huttunen, A.: Peer reviews in real life-motivators and demotivators. In: Fifth International Conference on Quality Software (QSIC 2005). IEEE (2005)
18. Jacob, R., Heinz, A., Décieux, J.P.: Umfrage: Einführung in die Methoden der Umfrageforschung. Walter de Gruyter (2013)
19. Kollanus, S., Koskinen, J.: Software inspections in practice: six case studies. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 377–382. Springer, Heidelberg (2006). doi:10.1007/11767718_31
20. Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., Linssen, O., Hanser, E., Prause, C.R.: Hybrid software and system development in practice: waterfall, scrum, and beyond. In: ICSSP 2017 (2017)
21. Laitenberger, O., Vegas, S., Ciolkowski, M.: The state of the practice of review and review technologies in Germany. Tech. report 011.02, Virtual Softw. Eng. Competence Center (VISEK) (2002)
22. Laitenberger, O., DeBaud, J.M.: An encompassing life cycle centric survey of software inspection. *J. Syst. Softw.* **50**(1), 5–31 (2000)
23. March, J.G., Simon, H.A.: Organizations. Wiley, London (1958)
24. Rigby, P.C., Bird, C.: Convergent contemporary software peer review practices. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 202–212. ACM, Saint Petersburg (2013)
25. Rigby, P.C., Storey, M.A.: Understanding broadcast based peer review on open source software projects. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 541–550. ACM (2011)
26. SmartBear: The state of code quality 2016. <https://smarterbear.com/resources/ebooks/state-of-code-quality-2016/>. Accessed 13 June 2017
27. Sudman, S., Bradburn, N.M.: Asking Questions: A Practical Guide to Questionnaire Design. Jossey-Bass Publishers, San Francisco (1982)

28. Thongtanunam, P., McIntosh, S., Hassan, A.E., Iida, H.: Investigating code review practices in defective files: an empirical study of the QT system. In: Proceedings of the 12th Working Conference on Mining Software Repositories, MSR 2015 (2015)
29. Wang, J., Shih, P.C., Wu, Y., Carroll, J.M.: Comparative case studies of open source software peer review practices. *Inf. Softw. Technol.* **67**, 1–12 (2015)
30. Winter, M., Vosseberg, K., Spillner, A.: Umfrage 2016 “Softwaretest in Praxis und Forschung”. dpunkt.verlag (2016)