A Faceted Classification Scheme for Change-Based Industrial Code Review Processes

Tobias Baum, Olga Liskin, Kai Niklas and Kurt Schneider
Leibniz Universität Hannover
Hannover, Germany
Email: [firstname.lastname]@inf.uni-hannover.de

Abstract—**Code review in the industry today is different to code review twenty years ago. The process has become more lightweight, reviews are performed frequently and change-based and the use of specialized tools is increasing. An accurate view of the current state of the industrial practice is an indispensable foundation for improving it. Most recent descriptions of review practices come from a limited population of large high-tech companies. Therefore, we used interviews with software engineering professionals from a broad sample of 19 companies to gain insight into their code review practices. We augmented our findings with data for 11 companies found through a semi-systematic literature review. There are many commonalities in the code review processes of these companies, but also a lot of variation in the details. A simple process taxonomy cannot describe these variations adequately. Therefore, we present a faceted classification scheme that is grounded in our observations.**

Index Terms—**Code Reviews, Code Inspections and Walkthroughs, Software Engineering Process, Empirical Software Engineering**

I. Introduction

Code review, especially in the form of Inspection [1], is a well-researched pillar of software quality assurance. In recent years, the code review practices in industry have changed considerably. Coinciding with the rise of agile and open source development practices, they have become more lightweight, continuous and asynchronous [2]. There are several studies that describe the state of or improvements to the code review processes of large high-tech companies and open source projects. For small and medium software enterprises (SMEs), there is much less evidence.

Our long-term research objective is to improve the use of code reviews in industry, with a focus on SMEs. A thorough understanding of the current state of the practice and of the underlying reasons and problems will help to guide this research. Therefore, we performed a grounded theory study based on interviews with 24 software engineering professionals from 19 companies. Of these companies, 11 regularly perform code reviews. The current article presents some of the insights gained from these interviews. The broad research questions treated in this article are: *How is code review performed in industry today? Which commonalities and variations exist between code review processes of different teams and companies?* To triangulate our findings, we performed a semi-systematic literature review to find recent descriptions of code review processes in the research literature. We consolidated the findings from the interviews and from the literature review into a classification scheme for industrial code review processes. This classification scheme is focused on change-based code review processes, because we found them to be the dominant type of code review in our sample. Our study is one of the largest and most detailed studies of industrial code review practices to date. The presented classification scheme can help to guide future research. It can also be valuable for practitioners by stimulating their creativity when there are problems with their current code review process.

II. Related work

There is a large body of literature on code reviews and inspections, with the works from Fagan [1] often seen as the most important initial contribution. Much of the early knowledge on inspections has been condensed in books, like the ones by Gilb and Graham [3] and Wiegers [4], and literature reviews [5] [6]. An IEEE standard defines classical forms of reviews [7]. All of these books and articles contain some sort of taxonomy or classification scheme for review processes. The scheme presented by Laitenberger and DeBaud [6] is faceted, as is the scheme by Kim, Sauer and Jeffery [8]. Macdonald and Miller [9] develop a domain-specific language for the description of inspection processes based on a detailed comparison of different processes from the literature. All of these schemes focus on classic inspections and none of them is based on recent empirical data.

Most recent studies that describe code review processes only describe a single case or company. We identified a number of those studies in the semi-systematic literature review (see section III-B). They are shown in table II.

A study that consolidates several cases has been done by Kollanus and Koskinen [10]. It describes software inspection practices based on interviews with practitioners from five Finnish companies. In their sample, code review was quite rare and consequently not described in much detail. They conclude stating a need for further case studies on characteristics and problems of software inspection in practice.

In Rigby's and Bird's recent study [2], they compare peer review processes from several projects and note convergence towards a common process. This process is lightweight, flexible and based on the frequent review of small changes. Their analysis contains qualitative and quantitative parts, with a focus on the quantitative analysis of data sets from review tools. They surveyed a broad range of projects, but their

TABLE I
DEMOGRAPHICS AND REVIEW USE OF COMPANIES FROM THE INTERVIEW STUDY

ID	type	employees (IT, approx.)	country	dev. process	spatial	regular review use
IA	in-house IT, travel	450	DE	agile	co-located	no
IB	standard software, dev. tools	400	CZ	ad hoc	distributed	yes
IC	standard software, government	200	DE	classic / ad hoc	co-located	no
ID	standard software, CAD	100	DE	ad hoc	co-located	yes
IE	standard software, output mgmt.	70	DE	agile	co-located	yes
IF	standard software, agriculture	130	DE	agile	co-located	yes
IG	standard software, retail	50	DE	agile	co-located	yes
IH	contractor, automotive	70	DE	agile	co-located	yes
II	SaaS, dev. tools	5	US	ad hoc	distributed	no
IJ	in-house IT, finance	1100	DE	ad hoc	co-located	no
IK	in-house IT, finance	200	DE	classic	co-located	no
IL	in-house IT, finance	400	DE	classic	co-located	yes
IM	in-house IT, government	200	DE	classic	co-located	no
IN	in-house IT, marketing	50	DE	agile	co-located	no
IO	–	–	DE	–	co-located	yes
IP	in-house IT, finance	–	DE	–	distributed	yes
IQ	in-house IT, retail	120	DE	classic / agile	co-located	yes
IR	in-house IT, marketing	50	DE	agile	co-located	no
IS	in-house IT, automotive	4000	DE	agile	co-located	yes

study is limited to projects from large companies (Google, Microsoft, AMD) and large open source projects (Apache, Subversion, Linux, . . .). Our study differs from theirs by using a different methodology and extends it by studying a broader range of organization sizes and styles. Nevertheless, Rigby's and Bird's study is the one most closely related to ours and many of our findings confirm theirs.

Besides these qualitative studies, there are questionnaire-based surveys on review usage in practice: The largest scientific survey was done in 2002 by Ciolkowski, Laitenberger and Biffl [11]. Based on 226 responses, they found a share of 28% of the participants doing code reviews.

III. METHODOLOGY

The current article consolidates two independent data sets, one from an interview study that was done using Grounded Theory methodology [12], [13], [14], the other from a semi-systematic literature review. Section III-A provides details on the former, section III-B on the latter. Section III-A also contains a description of the method we used to derive the classification scheme.

A. Interview Study

To describe review practices without being biased by published suggestions for review processes, we chose an inductive, data-driven approach as our main method of analysis. We have chosen interviews with practitioners as our primary source of data because they are well-suited to elicit detailed descriptions and to avoid misunderstandings.

We performed "theoretical sampling" to select the interviewees: We used our emerging results to choose further participants, for example from a so far little-investigated context. We gained access to them either by direct or indirect personal connections or by approaching them on conferences or after they showed interest in code reviews on the Internet. In total, we performed 22 interviews with 24 participants. They described 22 different cases of code review use in 19 companies. The resulting sample of companies is heterogeneous, going in size from 5 to about 4000 IT employees, and using agile as well as classic and ad hoc processes. A majority of the companies are based in Germany and work co-located, but we also included contrasting cases. Detailed information on the companies can be seen in table I. The interviewees for companies *IN*, *IO* and *IP* were interviewed in a group interview. This turned out to be problematic because there was not enough time to convey all relevant information and we could not reach *IO* and *IP* to clarify further questions afterward. The interviewees were mostly software developers and team or project leaders, with an industrial software development experience ranging from 3 to 30 years. For all companies but *IE*, we only interviewed one participant each. The interviews were conducted between September 2014 and May 2015. Data collection was stopped when theoretical saturation was reached, i.e. when there was only marginal new information in the last interviews.

The interviews were semi-structured, using open-ended questions. The corresponding interview guide was initially created based on the research questions and checked by another researcher who has experience with interview studies and Grounded Theory. It was later continually adjusted according to earlier interview experiences and the emerging theory[1]. All interviews were recorded and later transcribed. Most interviews were conducted by the first author. Some of the interviewees were colleagues of the first author. To reduce the resulting bias as a potential threat to credibility, these particular interviews were performed by the second author.

After performing and transcribing the first few interviews,

[1]The first and last used versions of the interview guide are available online: http://tobias-baum.de/rp/interviewGuideFirst.pdf, http://tobias-baum.de/rp/interviewGuideLast.pdf

we started data analysis: We used open coding to identify common themes in the data and analyzed the resulting codes for dimensions in which they vary as well as similarities. Coding was done paper-based at first and later using Atlas.TI [15]. Coding was done incrementally and iteratively, including new interviews as they were taken and revisiting most interviews several times. The results were compared and discussed afterward to check for possible bias or different viewpoints. The resulting analysis was reported back to all participants, asking for review regarding misunderstandings and relevance. This "member checking" resulted in minor extensions and changes to our results and increased our confidence that they are a suitable abstraction of the data.

We included a variation point as a facet of the classification scheme when it is an identifiable, fixed part of the process for at least one case and at least two different variations were observed. Some of the interviewees reported several distinct "cases" of review use for a single company. We consider a review variant a distinct case if it differs from other review process variants in the same company and if this choice only depends on external factors (team, product, . . .). We will later use the IDs from table I as subscripts to refer to the companies. When the ID is followed by a number, this refers to a specific case for that company.

B. Literature Review

We performed a semi-systematic literature review to find recent descriptions of industrial code review processes. We used these descriptions to triangulate and extend our findings from the interviews. The literature review is systematic in the way that we used the rigorous procedure for snowballing-based systematic literature reviews described by Wohlin [16]. It is semi-systematic because the decisions regarding inclusion or exclusion of studies were done only by a single researcher.

Our inclusion criteria for studies were as follows: (1) the study has been published since 2006 (inclusive), (2) it has been peer-reviewed, (3) it is published in English and (4) it describes code review practices in industry in some detail. There has to be some clue that the process is really used and not only brought into the company for a case study by the researchers. The description of the code review process does not have to be the article's main topic, as long as it is described in enough detail. We included open source projects that are largely driven by a company (e.g. Android, Qt) as "industrial" and excluded other open source projects.

The start set consists of four papers: [2], [19], [23], [28]. They have been chosen because they span a number of different years and publication venues. In addition, we consider the article by Rigby and Bird [2] to be a key publication that combines several previous studies and that is cited relatively often. We reached saturation after four iterations. Table II lists the found sources, grouped by the companies whose code review process they describe. The information gained from the publications was much more shallow compared to the rich descriptions from the interviews. Its main use is as additional evidence for the identified variants.

TABLE II
COMPANIES WITH REVIEW PROCESS INFORMATION EXTRACTED FROM THE LITERATURE REVIEW

ID	**Company name**	**Sources**
LA	AMD	[2], [17]
LX	*name unknown*	[18]
LE	Eiffel Software	[19]
LC	Critical Software S.A.	[20]
LV	VMWare	[21]
LF	Frequentis	[22], [23]
LM	Microsoft	[2], [24], [25], [26]
LG	Google/Android	[2], [27]
LS	Sony Mobile	[28]
LQ	Digia/Qt	[29]
LL	Salesforce.com	[30]

IV. RESULTS

The comparison of the code review processes described by the interviewees as well as in the literature revealed commonalities and differences of these processes. The basic review process and its embedding into the development context were similar in the studied cases. It is described in section IV-A. To systematize the observed differences, we developed a classification scheme. Section IV-B provides an overview of this scheme and some of the rationale behind it. The following sections IV-C to IV-G describe the single facets in detail.

A. Commonalities of Code Review Processes in Industry

In our interviews, all interviewees had a rather broad but common idea of code reviews. It is summarized in the following definition. Together with definition 2, it defines the scope of our classification scheme:

Definition 1: Code Review is a software quality assurance activity with the following properties:

- The main checking is done by one or several humans.
- At least one of these humans is not the code's author.
- The checking is performed mainly by viewing and reading source code.
- It is performed after implementation or as an interruption of implementation.

The humans performing the checking, excluding the author, are called "reviewers".

Each part of the definition delimits code review towards other quality assurance techniques, namely static analysis, self checks, testing and pair programming. All these delimitations are blurred: Human reviewers can be assisted by static code analysis, they sometimes execute tests or click through the GUI, in some cases the author joins the reviewers in checking his own code, and when author and reviewer jointly correct issues on-the-fly, they are basically doing pair programming. Definition 1 specifies a least common denominator of what practitioners consider a code review, in contrast to the definitions given in the IEEE Standard for Software Reviews and Audits [7] that describe specific processes in detail.

In our interviews, we observed cases where code review was performed irregularly and driven by individual needs as well as cases in which there was a defined and regularly used code review process. In this article, we focus on the latter type. In all of the observed cases, this regular code review process was change-based [31]:

Definition 2: Regular, change-based code review is a type of code review that is codified in the development process of the team or organization in the following way: Every time a "unit of work" is seen as "ready for review", all changes that were performed in the course of its implementation are considered a review candidate. This candidate is then assessed: For which parts of the change is a review needed, and is it needed at all? If a review is needed, the review candidate then waits for the reviewers to start reviewing.

The term "unit of work" is similar to the "patch set" identified in other studies. We did not use "patch set" because it seemed more narrowly focused on specific technologies to us. We consider "differential code review" [32] to be a synonym to "change-based code review". The terms "modern code review" and "contemporary code review" [2] are sometimes used as synonyms and sometimes refer to specific sub-variants of change-based code reviews.

Of the companies from the interviews, eleven (*IB*, *ID* - *IH*, *IL*, *IO*, *IP*, *IQ* and *IS*) have a regular, change-based code review process, whereas the remaining eight (*IA*, *IC*, *II*, *IJ*, *IK*, *IM*, *IN*, *IR*) only do irregular code reviews. Systematic review that was not change-based was mentioned in the interviews, but always in the form of "we did this once, but it was discontinued". In the literature, *LE* and *LC* describe cases that do not use change-based code review. All other literature sources describe change-based code review processes.

We identified the following commonalities of the observed change-based code review processes:

- No management action is required to trigger single code reviews, they are triggered solely based on pre-agreed rules. This replacement of the planning phase with conventions and rules is the difference most consistently separating change-based code reviews from classical inspection variants.
- When code reviews are performed in addition to unit testing or other developer-centric tests, testing is performed before code review. The same applies to static code analysis and to (other) checks that are performed automatically on a continuous integration server.
- The number of reviewers is two or less for the majority of reviews.
- All teams try to prevent situations in which code review happens after the changes are released to the customer.

B. A Faceted Classification Scheme for Change-Based Code Review Process Variation Points

Common classification schemes for review processes use simple labels like "Inspection", "walkthrough" or "pass-around" to classify types of review processes as a whole. Such a one-dimensional taxonomy simplifies discussions, but we found it to be inadequate for describing the variations we found without losing a lot of information. Therefore, we opted for a *faceted* classification scheme: A review process is described by the combination of values for a number of facets/dimensions. Because we identified a large number of facets, we grouped them into thematic categories. The resulting scheme is summarized in Fig. 1, which shows the groups, the contained facets and the possible values for the facets. The number in parentheses is the section in which the group or facet is described in more detail.

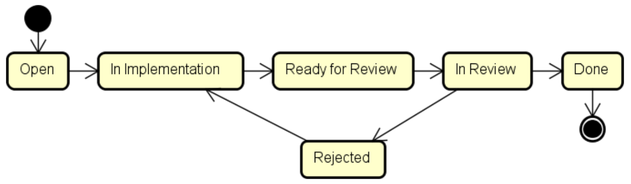

Fig. 2. Example of code review embedded in a task ticket's states. $_{IE}$ [33]

C. Process embedding

The first group of facets contains aspects that varied with regard to how the code review process is embedded into the rest of the development process, i.e. when and in which way a review is triggered and which influences it has on other process activities.

1) Unit of work: Definition 2 states that the changes performed in a "unit of work" define the scope of the review. The smaller the unit of work, the smaller the effort spent on every single review, but the higher the number of reviews. In the studied cases, one of these types of "unit of work" was chosen as a trigger for reviews:

RELEASE Review is triggered for changes that are put into production or are ready for "production approval" $_{IQ}$

STORY/REQUIREMENT Review is triggered for a user story/requirement that is considered done $_{IF1,IH,IL}$

TASK Many teams divide user stories into separate implementation tasks. If the chosen value for "unit of work" is "task", a review spans the changes done in such an implementation task. $_{IE,IF2,IG,LX,LF}$

PUSH/PULL/COMBINED COMMIT Review is done for each source code management (SCM) "pull request" or some other types of combined commit. A pull request often corresponds to a task when a team uses both. $_{IB1,IO,IS,LA,LS,LQ}$

SINGULAR COMMIT Review is triggered for every small-grained SCM "commit". This variant is mainly used when changes are rare and strictly controlled, such as in release branches. $_{ID,IB2,IP1,IP2}$

2) Tool support/enforcement for triggering: The triggering of reviews can be supported by tools, or it is done completely manually:

TOOL A tool ensures that a review candidate is created for each unit of work. This can be accomplished

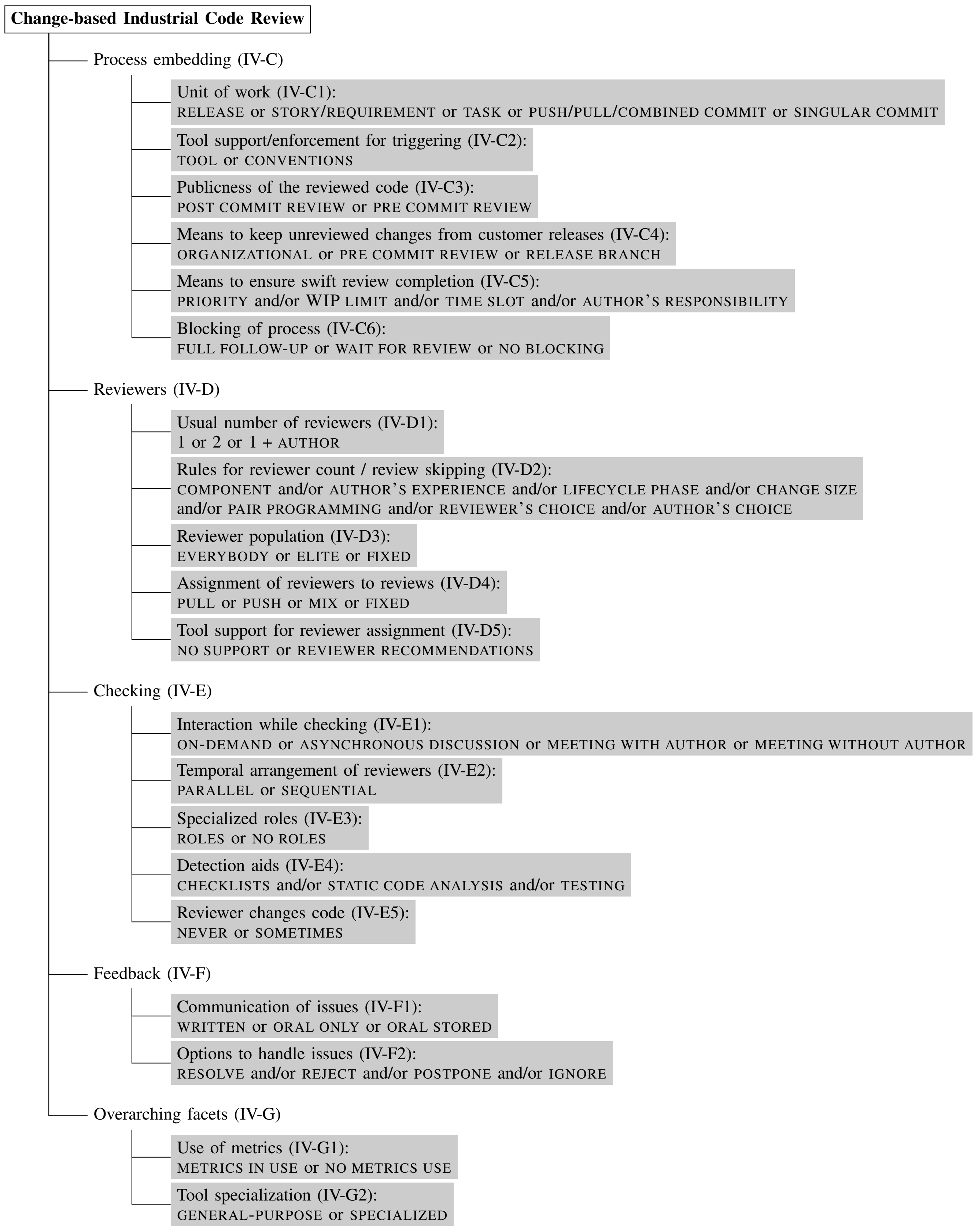
Change-based Industrial Code Review
Process embedding (IV-C)
Unit of work (IV-C1): RELEASE or STORY/REQUIREMENT or TASK or PUSH/PULL/COMBINED COMMIT or SINGULAR COMMIT
Tool support/enforcement for triggering (IV-C2): TOOL or CONVENTIONS
Publicness of the reviewed code (IV-C3): POST COMMIT REVIEW or PRE COMMIT REVIEW
Means to keep unreviewed changes from customer releases (IV-C4): ORGANIZATIONAL or PRE COMMIT REVIEW or RELEASE BRANCH
Means to ensure swift review completion (IV-C5): PRIORITY and/or WIP LIMIT and/or TIME SLOT and/or AUTHOR'S RESPONSIBILITY
Blocking of process (IV-C6): FULL FOLLOW-UP or WAIT FOR REVIEW or NO BLOCKING
Reviewers (IV-D)
Usual number of reviewers (IV-D1): 1 or 2 or 1 + AUTHOR
Rules for reviewer count / review skipping (IV-D2): COMPONENT and/or AUTHOR'S EXPERIENCE and/or LIFECYCLE PHASE and/or CHANGE SIZE and/or PAIR PROGRAMMING and/or REVIEWER'S CHOICE and/or AUTHOR'S CHOICE
Reviewer population (IV-D3): EVERYBODY or ELITE or FIXED
Assignment of reviewers to reviews (IV-D4): PULL or PUSH or MIX or FIXED
Tool support for reviewer assignment (IV-D5): NO SUPPORT or REVIEWER RECOMMENDATIONS
Checking (IV-E)
Interaction while checking (IV-E1): ON-DEMAND or ASYNCHRONOUS DISCUSSION or MEETING WITH AUTHOR or MEETING WITHOUT AUTHOR
Temporal arrangement of reviewers (IV-E2): PARALLEL or SEQUENTIAL
Specialized roles (IV-E3): ROLES or NO ROLES
Detection aids (IV-E4): CHECKLISTS and/or STATIC CODE ANALYSIS and/or TESTING
Reviewer changes code (IV-E5): NEVER or SOMETIMES
Feedback (IV-F)
Communication of issues (IV-F1): WRITTEN or ORAL ONLY or ORAL STORED
Options to handle issues (IV-F2): RESOLVE and/or REJECT and/or POSTPONE and/or IGNORE
Overarching facets (IV-G)
Use of metrics (IV-G1): METRICS IN USE or NO METRICS USE
Tool specialization (IV-G2): GENERAL-PURPOSE or SPECIALIZED

Fig. 1. Overview of the classification scheme. Each gray box is a facet. Possible values are written in small caps. Values separated by "or" are alternatives, values separated by "and/or" can be combined. A tuple with values for all facets describes a review process.

for example with a separate state in a bug tracker's ticket workflow or with specialized tools that enforce this process, like pull requests on GitHub[2] or Gerrit[3]. An example of a separate task state is shown in Fig. 2. *ID,IE,IF1,IF2,IG,IO,IS,LG,LS,LQ*

CONVENTIONS If no tools are used, conventions and group pressure are used to reach process compliance: *"... and this is more like peer pressure. If something goes live and there was no review, the other developers will ask 'Why not? What's up?' "* *IQ IB1,IB1,IH,IL,IQ,LM*

3) Publicness of the reviewed code: The changes under review can be made public to other developers before the review, or they are still private during the review. The two corresponding values for this facet are:

POST COMMIT REVIEW A review is performed after the changes are visible for other developers[4]. *IB1,IE,IF1,IF2, IG,IH,IL,IP2,IQ*

PRE COMMIT REVIEW A review is performed before the changes reach the main development trunk[5]. Review using pull requests is a special case of pre commit review. *IB2,ID,IO,IP1,IS,LA,LV,LM,LG,LQ*

4) Means to keep unreviewed changes from customer releases: All studied teams that perform regular reviews to detect defects try to avoid performing reviews after the changes have been released to the customer(s). They choose different means to reach this goal:

ORGANIZATIONAL They manually check for open reviews when a release approaches, in combination with organizational means to ensure swift completion of reviews (see section IV-C5). *IE,IF1,IF2,IH,IL,IQ*

PRE COMMIT REVIEW They use pre commit reviews (see section IV-C3), either generally or for a certain time before releases. *IB2,ID,IO,IP1,IS,LA,LV,LS*

RELEASE BRANCH There is a permanent technical separation between development branch/stream and release branch/stream. *IG,LS*

5) Means to ensure swift review completion: To ensure that a review does not stay open for too long and reviews don't accumulate too much, many teams employ at least one of the following organizational means:

PRIORITY Reviews have higher priority than other tasks (*"In one team we defined a rule in which order tasks have to be worked on. That production problems are the first priority, but that when the implementation of a user story is done and it is ready for review on the taskboard, that this has a higher priority then starting a new story. To keep the cycle times in a sprint short."* *IF*). *IE,IF1,IF2*

WIP LIMIT The team has a "work in progress (WIP) limit"[6] that restricts the number of tasks that can be "ready for review". *IH*

[2]https://github.com
[3]https://www.gerritcodereview.com
[4]also called "Commit Then Review" in other publications
[5]also called "Review Then Commit" in other publications
[6]a term from "Kanban" and similar methodologies [34]

TIME SLOT The reviewers reserve specific times of the day or week for code review. *IG*

AUTHOR'S RESPONSIBILITY The author actively seeks out a reviewer, perhaps they even review together. When pre commit reviews are used, the author has an interest to get the review done, in order to get his changes included into the common code base. *IB1,ID,IL,IO,IP1,IQ,IS*

6) Blocking of process: There are steps following code review in a unit of work's life cycle. Commonly this is declaring the feature ready for use and delivering it to the customer. The observed cases differ in whether these following steps are blocked while the review is underway:

FULL FOLLOW-UP The steps following code review in a unit of work's life cycle, e.g. declaring it as "ready for delivery", are not begun until all issues found in the review have been resolved. *IB2,ID,IE,IF1, IF2,IG,IL,IO,IP1,IP2,IS,LA,LF,LM,LG,LS,LQ*

WAIT FOR REVIEW The unit of work is blocked until the reviewers finished checking. Fixing is done based on trust only and not explicitly waited for. *IH,IQ*

NO BLOCKING The unit of work can be further processed, e.g. delivered to the customer, without checking for missing reviews. *IB1*

D. Reviewers

The facets belonging to this group describe differences we found regarding the selection of reviewers.

1) Usual number of reviewers: It was already noted as a commonality of the observed processes that there are usually at most two reviewers (excluding the author). Some teams report rare cases with more reviewers: *"When it's database migration code [...], five to six selected reviewers have to give their OK to it"* *IH*. In cases *ID* and *IL*, the author usually takes part in reviewing (see section IV-E1).

2) Rules for reviewer count / review skipping: The teams have different rules regarding the minimal number of reviewers, and whether this number can be zero, i.e. review can be skipped. In most cases, at least one of the following factors can influence the minimal number of reviewers:

COMPONENT Review is only obligatory for certain components, or certain components have to be checked by more reviewers. These components are commonly more complex or the estimated consequences of defects are more severe. *IB1,IE,IO,LS*

AUTHOR'S EXPERIENCE Only changes by inexperienced developers have to be reviewed. *IP2*

LIFE CYCLE PHASE In some teams, review is only obligatory near releases. *IB2,ID,IP1,LS*

CHANGE SIZE Changes smaller than a certain threshold do not have to be reviewed. *IL*

PAIR PROGRAMMING Changes done using pair programming do not have to be reviewed or have to be reviewed by one reviewer less. *IE,IF2,IS*

REVIEWER'S CHOICE The reviewer can decide that a review is not needed or that a second reviewer would be advisable, for example based on an assessment of the change's complexity. *IF1,IF2,IG,IH*

AUTHOR'S CHOICE The author can choose to have additional reviewers, e.g. because she considers the change risky. *IB1,IB2*

3) Reviewer population: There are large differences in the studied cases regarding the set of potential reviewers. The significant factor is how, and to what extent, the experience of the potential reviewer is taken into account. Among the studied teams, the variants were:

EVERYBODY Every team member shall be available as a reviewer for every change, mostly with some exceptions allowed. *IB1,IB2,IE,IF1,IF2,IH,IL,LF*

ELITE Only a certain subset of experienced developers shall do reviews. *IG,IS,LF,LG,LL*

FIXED All reviews are done by the same reviewer(s). In one of the observed cases, the team elects two experienced reviewers to perform every review. Another variant is the team leader reviewing everything. *ID,IP2,IQ,LX*

The authors themselves are not taken into account as the only reviewer in any of the cases, but they are sometimes asked to review jointly with another developer.

4) Assignment of reviewers to reviews: The interviewees described several possibilities how the connection between reviewers and reviews can be determined:

PULL Using this style of reviewer assignment, it is the reviewer who chooses among the outstanding reviews. *IE,IF1,IF2,LF*

PUSH This is the opposite of the "pull" style: The author chooses who should perform the review. *IB1,IB2, IL,LA,LV,LM,LQ,LL*

MIX In a mix of the "push" and "pull" style, the author invites a preferred subset of the reviewer population and the potential reviewers then decide if they want to participate. *IG,IH,IS,LA,LM,LQ*

FIXED In some cases, the same reviewers perform all reviews for a certain team or module, so that there is no choice left. *ID,IP2,IQ,LA,LV,LQ*

5) Tool support for reviewer assignment: When the number of potential reviewers is large and reviewer assignment is performed in the "push" style, tool support can help the authors find suitable reviewers.

NO SUPPORT There is no support for finding a suitable assignment of reviewers to reviews. *ID,IE,IF1,IF2, IG,IH,IL,IP2,IG,IS*

REVIEWER RECOMMENDATIONS There is computer support for finding suitable reviewers for a given change. *IB1,IB2,LV,LF,LM*

Tool support for "pull" assignment, i.e. help for the reviewers to find open reviews that are suitable for them, comes to mind as an obvious additional possibility. We could not observe this kind of support in the studied cases.

E. Checking

The facets in this group describe variations that we observed regarding the central activity of a code review: Checking the code. Like in the rest of this paper, we only describe variations in the codified processes of the teams, i.e. personal differences in the checking habits of the individual reviewers are out of the scope of our classification scheme.

1) Interaction while checking: Differences in reviewer interaction while checking have been described in some of the earliest publications on code reviews. We could also observe similar differences in the studied cases:

ON-DEMAND The review participants only interact on-demand, e.g. when there are questions regarding the code. *IE,IG,IP2,LV,LF*

ASYNCHRONOUS DISCUSSION In addition to on-demand interaction, review remarks are instantly communicated and are discussed asynchronously by the review participants. *IB1,IB2,IH,IO,IS,LA,LM*

MEETING WITH AUTHOR All review participants, potentially including the author, meet for checking. Often, the author actively guides the reviewer and explains the code. This has been called "pair review" by some of the interviewees. *ID,IF1,IF2,IL,LS*

MEETING WITHOUT AUTHOR The reviewers meet to discuss the code. This variant has been called "pair review", too. *IQ*

The main use of direct interaction according to our interviewees is to help to understand the code and to gain background information on implementation decisions (e.g. to decide if something is there for a legitimate reason or if it is an issue). It is also used to foster discussion on better solutions.

2) Temporal arrangement of reviewers: When multiple reviewers perform a review, we observed two variants of temporal arrangement:

PARALLEL All reviewers work in parallel and rework starts when all are finished. Parallel review is a prerequisite for direct inter-reviewer interaction. *IB1,IB2,ID, IF1,IF2,IH,IO,IQ,LF,LM*

SEQUENTIAL Only one reviewer reviews at a time and rework is done after each reviewer. *IE,IG,LV*

3) Specialized roles: When there are multiple reviewers, they can take on different roles, specializing on certain quality aspects. We only observed a limited number of cases using different roles, so that we divided this facet's possibilities coarsely:

ROLES The different reviewers take on different roles, e.g. with one reviewer specializing in code quality and another looking at GUI aspects. *IG,LF*

NO ROLES Reviewers do not take on different roles. *IB1,IB2, ID,IE,IF1,IF2,IH,IL,IP2,IQ,IS,LF,LQ*

4) Detection aids: The checking in a code review is performed mainly by viewing and reading source code (definition 1). To increase the chance of finding defects, some teams use one or several auxiliary techniques:

CHECKLISTS A checklist, i.e. a list of questions or topics that have to be checked in a review, is used by a small subset of the observed cases. *IF1,IF2,IL,IQ,IS,LA*

STATIC CODE ANALYSIS Static code analysis *during* checking can be used to guide and focus the reviewer on portions of the code that might need further attention. This differs from the use of static analysis before reviewing that is commonly done to find simple issues early (see section IV-A). $_{IE}$

TESTING In some of the cases, a limited amount of manual exploratory testing is common when performing a review (sometimes even adding additional unit tests during reviewing). $_{IB1,IB2,IE,IF1,IF2,IG,IL,LS}$

5) Reviewer changes code: When performing a code review, it is often tempting for the reviewer to directly change some code. The observed teams use one of two options with regard to code changes by the reviewer:

NEVER The reviewer is not allowed or technically not able to change code during checking. $_{IB1,IB2,ID,IH,IQ}$

SOMETIMES The reviewers may change code during checking. This option is most often used for changes that are small and which are regarded as riskless and not worth the effort to write a remark. $_{IE,IF1,IF2,IG,IL,IS}$

F. Feedback

Feedback in the form of review remarks is the main output of the checking. The facets in this group summarize differences in the handling of feedback between the team's review processes.

1) Communication of issues: The bulk of issues that is not fixed by the reviewer (and sometimes also issues that have been fixed) has to be communicated to the author. This communication can generally be divided into oral and written communication.

WRITTEN The found issues are mainly communicated in written form. Depending on the used tools, this can happen for example in the form of email, as ticket comments, using comments in the source code or using a specialized review tool. $_{IB1,IB2,IE,IF1,IF2,IG,IH,IO,IQ,IS,LA,LV,LF,LM,LG,LS,LQ}$

ORAL ONLY The issues are discussed orally with the author, mostly face-to-face. There is no storage of the issues, except for short-term note taking. $_{ID,LX}$

ORAL STORED The issues are mainly communicated orally, but also stored in written form, e.g. for traceability purposes. $_{IL}$

2) Options to handle issues: When the review remarks finally reach the author, he or she decides how to cope with them. We found the following possibilities:

RESOLVE Change the code according to the remark.

REJECT Ask for clarification or justification of the remark (when it is seen as some sort of "false positive" by the author).

POSTPONE Create a task to perform the needed changes. This task is then prioritized according to the team's development process. $_{IB1,IH,IL,IQ}$

IGNORE Do nothing. $_{IB1,IH,IQ}$

The actually found differences apply to the options POSTPONE and IGNORE. Some teams dismiss postponing with an attitude of "either fix it now or forget it". And some teams demand a reaction on every raised issue ("*... when the reviewer says something, that either has to be done this way or it has to be discussed together and a consensus reached.*" $_{IS}$) while for some it is an accepted practice to ignore remarks.

G. Overarching facets

The facets in this group pertain to aspects that span the whole code review process.

1) Use of metrics: The collection of review metrics and the systematic use of review results to improve the process is mentioned in the literature as a key feature that separates "inspection" from "Inspection" [3]:

METRICS IN USE Metrics on the code reviews are gathered and used systematically, for example for report generation and process improvement. $_{LA,LF,LM}$

NO METRICS USE Metrics on the code reviews are either not available or not used systematically. $_{IB1,IB2,ID,IE,IF1,IF2,IG,IH,IL,IO,IP1,IP2,IQ,IS}$

We could not observe the value METRICS IN USE in the empirical data from the interviews, but only in the cases described in the literature.

2) Tool specialization: In the preceding sections, several facets referred to tool support for specific tasks. More generally, we observed a difference between the use of a specialized tool and the combination of general purpose tools:

GENERAL-PURPOSE No specialized review software is in use. Instead, the teams use a combination of IDE, source code management system (SCM) and ticket system/bug tracker. Combined, these systems support all features seen as the core of a review tool: Viewing, examining and possibly editing the code, determining which parts of the code belong to a unit of work, documenting issues and integration of reviews into the development process. $_{ID,IE,IF1,IF2,IG,IL,IQ}$

SPECIALIZED The teams use a specialized code review tool that combines all relevant features. $_{IB1,IB2,IH,IO,IP2,IS,LA,LV,LF,LM,LG,LS,LL}$

Specifically, the following tools were used or mentioned in at least one of the observed cases: Gerrit[7], Crucible[8], Stash[9], GitHub pull requests[10], Upsource[11], Collaborator[12], ReviewClipse[13] and CodeFlow [26].

To conclude the presentation of the classification scheme, the tables III and IV show the values taken for each facet for the analyzed cases for which enough data for a thorough classification was available.

[7]https://www.gerritcodereview.com

[8]https://www.atlassian.com/software/crucible/

[9]https://www.atlassian.com/software/stash

[10]https://github.com

[11]https://www.jetbrains.com/upsource/

[12]http://smartbear.com/product/collaborator/

[13]https://www.inso.tuwien.ac.at/projects/reviewclipse/

TABLE III
PROCESS CHARACTERISTICS FOR THE CASES WITH REGULAR CODE REVIEWS (1/2). ONLY CASES WHERE INFORMATION FOR NEARLY ALL FACETS WAS AVAILABLE ARE LISTED.

Company/case	**IB1**	**IB2**	**ID**	**IE**	**IF1**	**IF2**
Unit of work (sec. IV-C1)	PUSH/PULL/COMBINED COMMIT	PUSH/PULL/COMBINED COMMIT	SINGULAR COMMIT	TASK	STORY/REQUIREMENT	TASK
Trigger support (sec. IV-C2)	CONVENTIONS	CONVENTIONS	TOOL	TOOL	TOOL	TOOL
Publicness (sec. IV-C3)	POST COMMIT REVIEW	PRE COMMIT REVIEW	PRE COMMIT REVIEW	POST COMMIT REVIEW	POST COMMIT REVIEW	POST COMMIT REVIEW
Keep from customer (sec. IV-C4)	none	PRE COMMIT REVIEW	PRE COMMIT REVIEW	ORGANIZATIONAL	ORGANIZATIONAL	ORGANIZATIONAL
Swift completion (sec. IV-C5)	none	AUTHOR'S RESPONSIBILITY	AUTHOR'S RESPONSIBILITY	PRIORITY	PRIORITY	PRIORITY
Blocking (sec. IV-C6)	NO BLOCKING	FULL FOLLOW-UP	FULL FOLLOW-UP	FULL FOLLOW-UP	FULL FOLLOW-UP	FULL FOLLOW-UP
Reviewer count (sec. IV-D1)	1	1	1+AUTHOR	1	1	1
Reviewer rules (sec. IV-D2)	COMPONENT + AUTHOR'S CHOICE	LIFECYCLE PHASE + AUTHOR'S CHOICE	LIFECYCLE PHASE	COMPONENT + PAIR PROGRAMMING	REVIEWER'S CHOICE	PAIR PROGRAMMING + REVIEWER'S CHOICE
Population (sec. IV-D3)	EVERYBODY	EVERYBODY	FIXED	EVERYBODY	EVERYBODY	EVERYBODY
Assignment (sec. IV-D4)	PUSH	PUSH	FIXED	PULL	PULL	PULL
Assignment support (sec. IV-D5)	REVIEWER RECOMMENDATIONS	REVIEWER RECOMMENDATIONS	NO SUPPORT	NO SUPPORT	NO SUPPORT	NO SUPPORT
Interaction (sec. IV-E1)	ASYNCHRONOUS DISCUSSION	ASYNCHRONOUS DISCUSSION	MEETING WITH AUTHOR	ON-DEMAND	MEETING WITH AUTHOR	MEETING WITH AUTHOR
Arrangement (sec. IV-E2)	PARALLEL	PARALLEL	PARALLEL	SEQUENTIAL	PARALLEL	PARALLEL
Roles (sec. IV-E3)	NO ROLES	NO ROLES	NO ROLES	NO ROLES	NO ROLES	NO ROLES
Detection aids (sec. IV-E4)	TESTING	TESTING	none	STATIC CODE ANALYSIS + TESTING	CHECKLISTS + TESTING	CHECKLISTS + TESTING
Code changes (sec. IV-E5)	NEVER	NEVER	NEVER	SOMETIMES	SOMETIMES	SOMETIMES
Issue communication (sec. IV-F1)	WRITTEN	WRITTEN	ORAL ONLY	WRITTEN	WRITTEN	WRITTEN
Issue handling options (sec. IV-F2)	RESOLVE + REJECT + POSTPONE + IGNORE	unknown	RESOLVE + REJECT	RESOLVE + REJECT	unknown	unknown
Metrics (sec. IV-G1)	NO METRICS USE	NO METRICS USE	NO METRICS USE	NO METRICS USE	NO METRICS USE	NO METRICS USE
Tool specialization (sec. IV-G2)	SPECIALIZED	SPECIALIZED	GENERAL-PURPOSE	GENERAL-PURPOSE	GENERAL-PURPOSE	GENERAL-PURPOSE

V. LIMITATIONS

In this section, we will first describe limitations of our approach to the collection and analysis of empirical data from the interviews. After that, we will describe some limitations of the literature review and weaknesses of our classification scheme.

A characteristic that our study shares with all Grounded Theory studies is that it relies on the human researcher as the instrument for data collection and analysis and is, therefore, prone to researcher bias. We tried to mitigate this threat by following Grounded Theory best practices and additional measures described below.

Some threats to validity result from our choice of interviews as the main method of data collection. There is a risk that the interviewees left out or changed some aspects of their processes when describing them. The interviews did not touch upon sensitive personal data. Nevertheless, we tried to reduce this risk by ensuring anonymity to all participants. We checked for consistency using triangulation of several descriptions of the same process at company *IE* and authors' observations at companies *IE* and *IJ*. This led us to conclude that this risk is

TABLE IV
PROCESS CHARACTERISTICS FOR THE CASES WITH REGULAR CODE REVIEWS (2/2). ONLY CASES WHERE INFORMATION FOR NEARLY ALL FACETS WAS AVAILABLE ARE LISTED.

company/case	**IG**	**IH**	**IL**	**IQ**	**IS**
Unit of work (sec. IV-C1)	TASK	STORY/REQUIREMENT	STORY/REQUIREMENT	RELEASE	PUSH/PULL/COMBINED COMMIT
Trigger support (sec. IV-C2)	TOOL	CONVENTIONS	CONVENTIONS	CONVENTIONS	TOOL
Publicness (sec. IV-C3)	POST COMMIT REVIEW	POST COMMIT REVIEW	POST COMMIT REVIEW	POST COMMIT REVIEW	PRE COMMIT REVIEW
Keep from customer (sec. IV-C4)	RELEASE BRANCH	ORGANIZATIONAL	ORGANIZATIONAL	ORGANIZATIONAL	PRE COMMIT REVIEW
Swift completion (sec. IV-C5)	TIME SLOT	WIP LIMIT	AUTHOR'S RESPONSIBILITY	AUTHOR'S RESPONSIBILITY	AUTHOR'S RESPONSIBILITY
Blocking (sec. IV-C6)	FULL FOLLOW-UP	WAIT FOR REVIEW	FULL FOLLOW-UP	WAIT FOR REVIEW	FULL FOLLOW-UP
Reviewer count (sec. IV-D1)	2	2	1+AUTHOR	2	1
Reviewer rules (sec. IV-D2)	REVIEWER'S CHOICE	REVIEWER'S CHOICE	CHANGE SIZE	none	PAIR PROGRAMMING
Population (sec. IV-D3)	ELITE	EVERYBODY	EVERYBODY	FIXED	ELITE
Assignment (sec. IV-D4)	MIX	MIX	PUSH	FIXED	MIX
Assignment support (sec. IV-D5)	NO SUPPORT	NO SUPPORT	NO SUPPORT	NO SUPPORT	NO SUPPORT
Interaction (sec. IV-E1)	ON-DEMAND	ASYNCHRONOUS DISCUSSION	MEETING WITH AUTHOR	MEETING WITHOUT AUTHOR	ASYNCHRONOUS DISCUSSION
Arrangement (sec. IV-E2)	SEQUENTIAL	PARALLEL	PARALLEL	PARALLEL	PARALLEL
Roles (sec. IV-E3)	ROLES	NO ROLES	NO ROLES	NO ROLES	NO ROLES
Detection aids (sec. IV-E4)	TESTING	none	CHECKLISTS + TESTING	CHECKLISTS	CHECKLISTS
Code changes (sec. IV-E5)	SOMETIMES	NEVER	SOMETIMES	NEVER	SOMETIMES
Issue communication (sec. IV-F1)	WRITTEN	WRITTEN	ORAL STORED	WRITTEN	WRITTEN
Issue handling options (sec. IV-F2)	unknown	RESOLVE + REJECT + POSTPONE + IGNORE	RESOLVE + REJECT + POSTPONE	RESOLVE + REJECT + POSTPONE + IGNORE	RESOLVE + REJECT
Metrics (sec. IV-G1)	NO METRICS USE	NO METRICS USE	NO METRICS USE	NO METRICS USE	NO METRICS USE
Tool specialization (sec. IV-G2)	GENERAL-PURPOSE	SPECIALIZED	GENERAL-PURPOSE	GENERAL-PURPOSE	SPECIALIZED

low enough. Nevertheless, the reader should keep in mind that there is only one data point for most of the companies and that a larger study using several data points from each company as well as longitudinal observation could provide more reliable results.

The choice of the interviewer can also result in bias. To mitigate this risk, we used an interview guide and asked another experienced researcher to check this guide before using it. As some of the interviewees were colleagues of some of the authors, we made sure that these interviews were conducted by another one of the authors so that passive researcher bias was reduced.

During data analysis, there is the risk of introducing observational bias, for example, when important data is not taken into account or open questions are not followed up. This risk is reduced through the used Grounded Theory practices: Careful and thorough coding, constant comparison, theoretical sampling and memoing. To avoid premature closing of the interviews, we explicitly asked for points missed in the discussion at the end of each interview. By recording and transcribing all interviews, we ensured that no information was lost unintentionally and that we could come back to the data for coding and checking several times.

To mitigate researcher bias during data analysis and interpretation, coding was performed by several authors and the results were discussed by all authors. As another important mitigation for observational as well as researcher bias, we performed "member checking": We provided our results to the study participants and asked for feedback, which was then incorporated into the study.

Regarding the literature review, we mitigated many potential threats to validity by following the rigorous procedure outlined by Wohlin [16]. A snowballing-based literature review is sensitive to the choice of the start set. We chose a start set that was heterogeneous and quite large, which minimized this threat. The choice to only include peer-reviewed publications ensured a minimum of reliability of the data. It comes at the cost of excluding many review process descriptions from other sources, e.g. from blogs on the Internet. The most significant threat to the validity of the literature review is that the decision on inclusion and exclusion was done only by a single author. We consider it acceptable because the results of the literature review are only used as an additional data source.

As Grounded Theory studies use theoretical sampling and rather small sample sizes, it is hard to assess their generalizability. We used a broad range of different interviewees from a heterogeneous sample of companies, and our sample size is quite large compared to most qualitative studies. By triangulating the empirical findings with data from the literature review, we gained more confidence in the generalizability of our findings. Nevertheless, the set of potential code review processes is essentially unbounded, so it would obviously be presumptuous to claim that our model captures *all* variation there is in change-based industrial code review processes.

By choosing a faceted description, we were able to cleanly describe the variations we found in practice. A weakness of this approach is that it is more complicated to use, compared to a small number of placative process names. In addition, interdependencies between certain values of the facets are not immediately obvious. Another drawback of the proposed model is the large number of facets. We intentionally did not exclude facets based on some measure of relevance, because relevance depends on the context in which the model is used. Researchers using our model should consider excluding facets that they know to be irrelevant for their study.

VI. Discussion

Our study confirms many of the findings of Rigby and Bird [2]. We could also note a convergence towards lightweight, change-based code review processes in industry. Additionally, many of the variations between the processes described by Rigby and Bird could be observed in our data, too. However, we couldn't observe as much convergence as they do regarding the subtleties of the processes. In part, this is probably due to our more fine-grained comparison of the processes, but there are also some contradictions: They state that code review converges towards a process that happens "before a change is committed", while we observed a quite balanced distribution of pre commit vs post commit reviews. Furthermore, we could observe several cases with face-to-face interaction or change sets larger than the ones described in their study. Lastly, they state that review has changed to a group problem solving activity, which is also not supported as convergent practice by our data. We believe that most of these differences are founded in different contextual characteristics of the respective study samples. Our sample contains many companies that are small and work co-located, in contrast to the large companies and open source projects sampled by Rigby and Bird. In a more abstract sense, we consider this an indicator that results from studies of open source software development can be valuable for commercial development, but it is also a warning that results from large companies and open source development are not fully generalizable to smaller companies.

An analysis and systematization of the current state of industrial code review processes is interesting in itself, but we believe that the main benefit of the current study is as a foundation for further research. The most obvious direction of further research is to complement and extend our results using quantitative methods. We are currently considering to perform a survey study that will allow us to assess which of the found variants are in wide-spread use and which are of marginal importance only. Such a study could also substantiate some hypotheses on the reasons for the observed variations that we gained from the interviews.

The current study is purely descriptive. It does not study which of the identified process variants perform best given a certain context and goal. Consequently, assessing the benefits and problems of the identified variants is another research avenue. For some facets such studies already exist, for example the studies by Votta [35], Johnson and Tjahjono [36] and others on "Interaction while checking" and "Communication of issues", the large body of literature on reading techniques with distinct roles (e.g. the studies by Laitenberger et al. [37]) and some results on differences regarding "Publicness of the reviewed code" by Rigby, German and Storey [38] and Baum et al. [33]. Nevertheless, many open questions remain.

A final direction of further research is to develop better processes and tools for code review. The current study allows researchers to identify possible incompatibilities between their suggestions and current industrial practices. Minimizing these incompatibilities helps to foster the adoption of improvements.

VII. Conclusion

The current article describes a classification scheme for change-based code review processes in industry. This scheme is based on descriptions of the code review processes of eleven companies, obtained from interviews with software engineering professionals that were performed during a Grounded Theory study. It has been further checked and extended with data from published code review process descriptions for eleven other companies found using a semi-systematic literature review.

Despite the differences in the details, there are many similarities in the observed code review processes. Code review is performed change-based, with rules and conventions taking the place of explicit planning and management actions. It is performed after unit testing and static code analysis verified the changes. The number of reviewers is generally low, mostly only one or two. Review processes that are not change-based are very uncommon and consequently not analyzed in detail in the study.

The classification scheme systematizing the processes' differences consists of 20 facets that were grouped into the following categories: "Process embedding", "Reviewers", "Checking", "Feedback" and "Overarching facets". The full scheme is shown in Fig. 1. Some exemplary facets are

- "Publicness of the reviewed code": In some cases reviews are performed before committing to the central source code repository, in other cases afterward,
- "Reviewer population": In some cases, all team members should or even have to be available as reviewers, while in others only an elite can act as reviewers, and
- "Temporal arrangement of reviewers": In some cases, multiple reviewers work in parallel, while in others the second reviewer starts only after the remarks of the first have been fixed.

The results presented in this article can be used when choosing or adapting a code review process in the industry, but we see their main use as a foundation for further research on code reviews.

Acknowledgments

The authors would like to thank all participants for the time and effort they donated for the interviews and for member checking. Further thanks go to Raphael Pham for his methodological advice and valuable feedback.

References

[1] M. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[2] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 202–212.

[3] T. Gilb and D. Graham, *Software Inspection*. Addison-Wesley, 1993.

[4] K. Wiegers, *Peer reviews in software: a practical guide*, ser. Addison-Wesley information technology series. Addison-Wesley, 2002.

[5] A. Porter, H. Siy, and L. Votta, "A review of software inspections," *Advances in Computers*, vol. 42, pp. 39–76, 1996.

[6] O. Laitenberger and J.-M. DeBaud, "An encompassing life cycle centric survey of software inspection," *Journal of Systems and Software*, vol. 50, no. 1, pp. 5–31, 2000.

[7] *IEEE Standard for Software Reviews and Audits*, IEEE Std. 1028-2008.

[8] L. P. W. Kim, C. Sauer, and R. Jeffery, "A framework for software development technical reviews," in *Software Quality and Productivity*. Springer, 1995, pp. 294–299.

[9] F. Macdonald and J. Miller, "Modelling software inspection methods for the application of tool support," University of Strathclyde, Tech. Rep. EFoCS-16-95, 1995.

[10] S. Kollanus and J. Koskinen, "Software inspections in practice: Six case studies," in *Product-Focused Software Process Improvement*. Springer, 2006, pp. 377–382.

[11] M. Ciolkowski, O. Laitenberger, and S. Biffl, "Software reviews: The state of the practice," *IEEE software*, vol. 20, no. 6, pp. 46–51, 2003.

[12] J. Corbin and A. Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 3rd ed. SAGE Publications, 2007.

[13] B. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, 1967.

[14] S. Adolph, W. Hall, and P. Kruchten, "Using grounded theory to study the experience of software development," *Empirical Software Engineering*, vol. 16, no. 4, pp. 487–513, 2011.

[15] S. Friese, *Qualitative Data Analysis with ATLAS.ti*. SAGE Publications, 2012.

[16] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *EASE '14*, 2014.

[17] J. Ratcliffe, "Moving software quality upstream: The positive impact of lightweight peer code review," in *Pacific NW Software Quality Conference*, 2009.

[18] M. V. Mantyla and C. Lassenius, "What types of defects are really discovered in code reviews?" *Software Engineering, IEEE Transactions on*, vol. 35, no. 3, pp. 430–448, 2009.

[19] B. Meyer, "Design and code reviews in the age of the internet," in *Software Engineering Approaches for Offshore and Outsourced Development*. Springer, 2009, pp. 126–133.

[20] A. L. Ferreira, R. J. Machado, L. Costa, J. G. Silva, R. F. Batista, and M. C. Paulk, "An approach to improving software inspections performance," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–8.

[21] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 931–940.

[22] M. Bernhart, S. Reiterer, K. Matt, A. Mauczka, and T. Grechenig, "A task-based code review process and tool to comply with the do-278/ed-109 standard for air traffic managment software development: An industrial case study," in *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*, 2011.

[23] M. Bernhart, S. Strobl, A. Mauczka, and T. Grechenig, "Applying continuous code reviews in airport operations software," in *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, 2012, pp. 214–219.

[24] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 712–721.

[25] C. Bird, T. Carnahan, and M. Greiler, "Lessons learned from building and deploying a code review analytics platform," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, 2015.

[26] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *MSR '15 Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 146–156.

[27] M. Mukadam, C. Bird, and P. C. Rigby, "Gerrit software code review data from android," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 45–48.

[28] J. Shimagaki, J. Shimagaki, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi, "A study of the quality-impacting practices of modern code review at sony mobile," in *ICSE 16 Companion*, 2016.

[29] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, pp. 1–44, 2015.

[30] T. Zhang, M. Song, J. Pinedo, and M. Kim, "Interactive code review for systematic changes," in *Proceedings of 37th IEEE/ACM International Conference on Software Engineering. IEEE*, 2015.

[31] R. A. Baker Jr, "Code reviews enhance software quality," in *Proceedings of the 19th International Conference on Software Engineering*. ACM, 1997, pp. 570–571.

[32] M. Bernhart and T. Grechenig, "On the understanding of programs with continuous code reviews," in *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*. IEEE, 2013, pp. 192–198.

[33] T. Baum, F. Kortum, K. Schneider, A. Brack, and J. Schauder, "Comparing pre commit reviews and post commit reviews using process simulation," in *Software and System Process (ICSSP), 2016 International Conference on*, 2016.

[34] C. Ladas, *Scrumban-essays on kanban systems for lean software development*. Lulu.com, 2009.

[35] L. G. Votta, "Does every inspection need a meeting?" *ACM SIGSOFT Software Engineering Notes*, vol. 18, no. 5, pp. 107–114, 1993.

[36] P. M. Johnson and D. Tjahjono, "Does every inspection really need a meeting?" *Empirical Software Engineering*, vol. 3, no. 1, pp. 9–35, 1998.

[37] O. Laitenberger, "Cost-effective detection of software defects through perspective-based inspections," Ph.D. dissertation, 2000.

[38] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *Proceedings of the 30th International Conference on Software Engineering*. ACM, 2008, pp. 541–550.