2024 34th International Conference on Collaborative Advances in Software and COmputiNg (CASCON) | 979-8-3315-0483-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/CASCON62161.2024.10838181

An Empirical Study on Collaborative Uses of Communication Channels for Software Development and Management

Mariam El Mezouar
Royal Military College of Canada
Kingston, Canada
mariam.el-mezouar@rmc.ca

Daniel Alencar Da Costa
University of Otago
Otago, New Zealand
danielcalencar@otago.ac.nz

Ying Zou
Queen's University
Kingston, Canada
ying.zou@queensu.ca

***Abstract*—Communication channels refer to the mechanisms adopted to ensure the flow of information within an organization. Developers may use different communication channels (such as e-mail, chat, or simply face-to-face) to discuss issues related to new features, bugs, or documentation. With a wide spectrum of different channels to choose from, project maintainers need to select the best suited channels for the development activities. We survey 129 developers to reflect on their experiences of using different communication channels for development activities (e.g., user support or daily communication). We find that different development activities require distinct combinations of channels. For example, *user support* appears to be more efficient with a fewer number of channels, reducing confusion on how to contact developers. We observe that a mix of *socially-enabled* and *non-digital* channels seems to fit most of development activities.**

***Index Terms*—communication channels, developers, social software engineering**

I. Introduction

Over the past decade, software development has transitioned from a predominantly solo activity of developing standalone applications to a highly collaborative activity where boundaries between projects and teams are blurred [1]. This transition has been powered by the increasing popularity of socially-enabled communication and collaboration tools (e.g., social coding platforms such as GitHub), which enable remote and asynchronous collaboration through social networks. The reliance on these tools has recently accerelated following the rapid transition to remote and hybrid work environments, as a result of the global pandemic.

In the context of software development projects, communication is not limited to direct interactions among the developers (e.g., an exchange of e-mails), as communication can take other forms [1].For example, when a user files an issue report, developers are notified and become aware of the new issue. Although the issue report is not intended to be a direct communication between the user and a developer, the issue report itself *communicates* a problem to the development team. We refer to any tool that is used by developers/users to communicate any aspect of the software project as a *communication channel*. In general, a *communication channel* can refer both to *traditional* channels, which were adopted before the advance of social media (e.g., face-to-face interactions, phone), and channels infused with *social features*—such as user profiles (e.g., Twitter), *following* features (e.g., GitHub), reputation systems (e.g., StackOverflow), and activity feeds (e.g., GitHub).

Given the different requirements of software projects, different communication channels have been developed, e.g., co-located teams require different communication channels than internationally distributed teams [2]. Different types of communication channels support diverse capabilities, such as chat-based synchronous communication (e.g., Slack), long-format asynchronous communication (e.g., email, mailing lists), and voice-based communication (Skype, Zoom, or Google Meet).

Due to the large variety of communication channels, researchers have investigated the characteristics and benefits of communication channels. Storey *et al.* [3] showed the emergence of communication channels that support networking and the sharing of community knowledge. In a later study by Storey *et al.* [1], they surveyed developers to identify the communication channels that are essential for software development. Despite the advancements of existing research, developers report new challenges related to communication channels. For example, the use of too many channels result in information fragmentation while face-to-face communication is still not easily mimicked by the available alternative channels (e.g., chat or email). There also exists a lack of awareness of when the use of certain channels is most appropriate (e.g., when is best to use synchronous vs. asynchronous channels?) [4].

Overall, there is a limited amount of empirical studies on adapting appropriate communication channels to a given set of requirements. To advance this line of work, we survey 129 developers who are active in both open source and private repositories, aiming to understand the rationale behind the choices for certain communication channels. Next, we strive to derive recommendations of communication channels for certain development activities. Our study is guided by the three research questions below:

RQ1. ***What is the relationship between the number of communication channels used and the development activities?***

Our goal is to find wether there is an ideal number of communication channels to be used depending on the development activity. We find that software development activities that require traceability of information (e.g., code review) are best communicated with a fewer number of channels (i.e., 2).

RQ2. ***Social, digital, or non-digital channels: why is each one used and is the choice dependent on the development activity?*** Our goal is to uncover how much of the communication is social, digital, and non-digital to help the developers communicate effectively. We derive the reasons behind the uses of each type of channel. We also identify a dependence between the development activities and the selected channels.

RQ3. ***What are the combinations of channels that work best?*** With different channels aimed at similar purposes, we examine whether certain combinations of specific channels (e.g., mailing lists + pull requests) have a positive association with the performance of the subject projects. We find that certain combinations of channels (e.g., pull requests and slack) are significantly and positively associated with the studied performance metrics (e.g., shorter bug fixing time).

For additional resources related to this study, including demographic details of participants, the invitation letter sent to participants, and the full set of survey questions, please visit our dedicated Zenodo repository.

II. Related Work

As we study the experience of developers regarding communication channels, we survey related research regarding: *a)* the importance of communication channels, and *b)* collaborative software development in the era of social media.

The importance of communication channels. There are several types of communication channels (e.g., instant messaging), each supporting a specific collaboration aspect (e.g., project awareness). For a while, mailing lists (coupled with issue tracking systems [5] and IDEs [6]) have been central to the software development process, particularly the open source projects (e.g., Linux, Apache). In terms of *project awareness*, an early study by Gutwin *et al.* [7] found that developers can maintain awareness of one another, and of the entire team's activities primarily through text-based communication (i.e., mailing lists and chat). Rigby *et al.* [8] examined the value of mailing lists to achieve *broadcast-based code review*. However, the study by Guzzi *et al.* [9] reported on the shift of interest to new social communication channels that support project development.

Another key type of communication channels adopted by developers is chat-based communication (e.g., IRC) [10]. On the use of IRC by the open source developers, Shihab *et al.* [11] reported that IRC meetings are popular among open source developers, as reflected by the volume of discussions and attendance levels, with content focused on project artifacts (e.g., test cases and source code). More recently, the modern chat systems, such as Slack and Gitter, brought further value to the developers through the offered social features (e.g., presence awareness and tagging users). In addition to discussing project artifacts, Lin *et al.* [12] reported that Slack is used for *information discovery*, and *the feeling of belonging to a community*.

Learning and *problem solving* are essential for software development. Developers' Q&A websites (e.g., StackOverflow) are used to support developers to solve problems and learn best practices. Mamykina *et al.* [13] found that the engagement with the community in StackOverflow is critical to successful knowledge sharing. Parnin *et al.* [14] reported on the steady growth of StackOverflow as an invaluable *documentation resource* (e.g., APIs). The activity rate of StackOverflow is found by Vasilescu *et al.* [15] to correlate with coding activities on GitHub. Twitter is another channel adopted by developers to *stay informed* on the latest trends and technologies. Singer *et al.* [16] reported that Twitter allows developers to keep up with the fast-paced development landscape, and even *build relationships*. El Mezouar *et al.* [17] found that monitoring the messages from end-users on Twitter can foster the *earlier discovery of bugs*.

Collaborative software development in the era social media. Existing research has investigated the communication channels used in collaborative software development. Notably, Storey *et al.* [3] delineated a timeline of how developers have adopted popular communication channels and social media channels. The timeline highlights the shift from a mostly solo software development to a social and transparent software development. The study examined the level of importance of different communication channels (e.g., a code hosting platform is found to be the most important) and their advantages (e.g., reducing collaboration barriers). A follow-up study by Storey *et al.* [1] stated that the studied communication channels support several development tasks, such as staying up to date, finding answers, and assessing others. For example, code hosting websites are found to be the most important to watch and publish activities. Their study also revealed challenges faced by developers in using communication channels (e.g., information overload).

Turner *et al.* [18] examined the use of communication channels in the workplace, over a one-year period in a small US corporation. The study highlighted the strengths and weaknesses of a set of communication channels, based on feedback from developers, in terms of function, immediacy, productiveness-efficiency, and social aspects. For instance, the strength of emails is keeping a persistent record, while, from a social aspect, it is the non-intrusiveness. The weakness of emails is the lack of immediacy, i.e., there is a long delay between emails with no tight looped discussion.

Other studies [19] [20] reported on the added value brought by socially-enabled channels to collaborative software development. Socially-enabled channels are platforms where participation of the users is transparent, in terms of identity, content, and interactions. A preliminary survey by Black *et al.* [19] revealed that socially-enabled channels are successfully used by the developers, mainly to discuss new ideas, design, and code-related issues. Giuffrida *et al.* [20] [21] investigated how social-enabled channels fit with the existing traditional communication channels. The socially-enabled channels are found

to act as a medium to negotiate coordination mechanisms, and to support specific communication tasks, such as knowledge sharing and team building.

Ultimately, the set of communication channels used by developers has had a strong adoption of new socially-enabled channels. Previous studies have examined the importance and challenges associated with the use of socially-enabled channels with an increasing presence of social features.

Complementing prior research, our work is the first to empirically study the motivation behind choices for certain communication channels. Our study is important to help developers make more informed decisions when choosing communication channels given their unique project settings. We derive recommendations regarding which communication channels to use for certain activities (e.g., bug fixing or release announcements).

III. Methodology

To capture the experience of the developers with communication channels, we distribute a survey to the developers (i.e., our respondents) contacted through GitHub. We use the replies to our survey to answer research questions 1 and 2. Afterwards, we study associations between the most frequent sets of communication channels and quantitative performance metrics (e.g., the time it takes to fix bugs) calculated from the projects on which developers based their answers.

A. Developers' Survey

1) Survey Design. We design our survey in 8 main parts. Due to space restrictions, we make our survey available online. In the first part, we ask about the respondents' *demographics* (e.g., gender, age, and overall development experience) and their roles in their software projects. Studying the role of the respondents is important to analyze the communication channels from different perspectives (e.g., user, contributor or maintainer). In the remaining seven parts, we ask about the software development activities that require communication and collaboration among developers.

Each part contains questions that are designed to capture: *a)* the different communication channels used for each activity, *b)* the satisfaction of developers regarding communication channels, and *c)* 2 open-ended questions for the elaboration and justification of the experiences and satisfaction ratings. The 2 open-ended questions included in the survey are intentionally generic not to steer the respondents in any direction.

2) Survey Distribution. Prior to the survey distribution, the survey is reviewed and granted clearance from the ethics board of our university. We target participants of all levels of expertise and development role to gain insights from different perspectives. For example, end users might be more sensitive to communication problems in the documentation compared to project maintainers, who are more familiar with the intricacies of the project. Therefore, our criterion to select the respondents is simply the participation in a GitHub repository through code commits.

We first identify developers with commit activities on GitHub, while considering all the public repositories hosted on the platform. This initial selection of the respondents results in over 4 million GitHub developers. We then select a statistically significant random sample (confidence interval = 2 and confidence level = 95%), resulting in 2,400 developers. Finally, we send the survey request to the selected developers, resulting in 129 responses, i.e., a 5.37% response rate (a comparable response rate to studies such as [22], [23]). In terms of the completeness of the responses, we have received $44.1\%^{57/129}$ fully completed forms, $27.1\%^{35/129}$ mostly completed forms (i.e., all multiple-choice questions and some open-ended questions), and $28.7\%^{37/129}$ partially completed forms (i.e., only the multiple-choice questions). The instances with missing responses to the open-ended questions are discarded.

The majority of the respondents have over 10 years of software development experience (i.e., 50.3%), and act as project maintainers (i.e., 51.2%). $67.4\%^{87/129}$ of the survey responses are associated to public repositories (e.g., Apache Spark), $17.8\%\ ^{23/129}$ to private repositories (e.g., Thumbtack), and $14.7\%\ ^{19/129}$ are unknown.

3) Survey Analysis. The survey contains two types of questions: a) multiple-choice questions, and b) open-ended questions. The analysis of the multiple choice questions (e.g., *What are the primary communication channels used to communicate about release planning?*) is performed using a mix of techniques, such as frequency and correlation analysis. We detail, under each research question, the approaches used to analyze the multiple choice questions.

Regarding the analysis of the open-ended questions, the first two authors (*the evaluators*) collaboratively performed a thematic analysis [24] for every survey response, until consensus was reached. The evaluators are assistant and associate professors in software engineering. Specifically, thematic analysis consists of extracting *themes* that capture the meaning within qualitative data. In the first phase of the thematic analysis (i.e., coding), a set of initial *codes* is generated by collapsing the data into a set of labels that capture relevant meaning to the research questions. In our study, a survey response can be collapsed into one or more codes, depending on the complexity and richness of the response. For example, the response "*Issue tracking and pull requests are a good way to plan future additions of features, but the additional communication overhead up to a pull request happens through Skype and E-Mail*" is assigned the codes 'feature planning' and 'communication overhead'. In the second phase of the thematic analysis, the codes are combined into overarching themes that accurately depict the data. For instance, the codes '*low friction*' and '*casual exchange*' can be categorized under the theme '*informal communication*'. To attach further meaning and context to the emerging themes, the evaluators record, for each theme/code, additional information. Using the above quote as an example, the evaluators take note of the specific communication channels (e.g., Pull requests) associated to the generated theme/code (e.g., *feature planning*). This information is used to present the themes from different perspectives.

Saturation is reached after analyzing approximately 50

survey responses. We refer to the individual respondents using a code of the form R# (where # depicts the unique identifier of a respondent). Whenever discussing our qualitative data, we support our observations by presenting quotes from the respondents and the resulting themes.

B. Software development activities

1) Definitions. In RQ3, we study the relationship between communication channels and a set of development activities. We specifically study 7 development activities that we believe involve most of the communications happening during the software development cycle, including the planning, design, implementation, testing, delivery, documentation, and maintenance of a software system [25]. We list below the development activities and the types of communication associated with each activity:

- **Release planning** involves communicating about aspects such as the product features and the project deadlines. It also serves as a base to monitor progress within the project [26].
- **Bug fixing** involves discussions about issues, such as new bugs, prioritization of bugs, bug assignment, and estimation of bug fixing time.
- **Code review** involves developers reviewing each others' source code to ensure code quality.
- **User support** involves communicating with the project's users to assist, troubleshoot, and collect issues reported by the users.
- **Recruiting developers** involves getting in touch with newly qualified developers to contribute to the source code of a project. A developer could voluntarily contribute or be invited to contribute by the maintainers.
- **Project promotion** involves increasing awareness about the project to attract more users.
- **Documentation** involves all the communication related to improving the project documentation.

We find evidence of the 7 development activities using the features offered in GitHub. For instance, the commits submitted to a project on GitHub can be either bug fixes (i.e., involving *bug fixing* and/or *code review* activities), new features (i.e., involving *release planning* and/or *code review* activities), or changes to the documentation (i.e., involving *documentation* discussions).

2) Performance metrics. We study the *most suitable* sets of communication channels with respect to the investigated activities, such as release planning, bug fixing, and code review. Out of the 87 public repositories associated with the survey responses, we can access the issue tracking systems, pull requests, and releases data for 58 repositories. The remaining 29 repositories do not use GitHub to track issues, manage pull requests, or create releases. We collect the following metrics:

- **Bug fixing time** captures the duration between the submission of a bug report and when the bug is resolved. Shorter bug fixing time intervals indicate that a development team is more *efficient* when fixing bugs. It is defined as:

$$T_{resolved} - T_{reported}.$$

where $T_{resolved}$ is the timestamp a given bug is fixed; and $T_{reported}$ is the timestamp the bug is reported.

- **Bug fixing rate** captures the proportion of fixed bugs from the set of reported and closed bugs. Higher bug fixing rates suggest that a development team is more *effective* in fixing bugs. It is computed as follows:

$$\frac{\#fixed\_bugs}{\#reported\_closed\_bugs}$$

- **Merge time** is the interval between the submission of a code change and its merging to the main branch of a project. Shorter merge time values can indicate that a development team is *efficient* when reviewing code changes. We compute this metric in the following way:

$$T_{merged} - T_{submitted}.$$

where T_{merged} is the timestamp a given code contribution is merged to the main branch of a project; and $T_{reported}$ is the timestamp the code contribution is first submitted.

- **Merge rate** captures the proportion of merged code changes in the submitted code changes. A higher merge rate is a possible indicator that a development team is more *effective* when reviewing code changes. It is computed as follows:

$$\frac{\#merged\_changes}{\#submitted\_changes}$$

- **Release frequency** reflects the frequency by which a project rolls out new releases. Higher release frequency possibly suggests the *efficiency* of a development team in preparing and packaging new releases. It is measured as follows:

$$average(\forall j : \Delta(T_{release_{j+1}} - T_{release_j}))$$

C. Categories of communication channels

TABLE I: Categorization of the communication channels.

Category	Communication channels
Socially-enabled and digital	Pull requests, Slack, Gitter, Google Chats, Twitter, Facebook, StackOverflow
Digital	Email, Mailing lists, Internet Relay Chat, Skype
Non-digital	Face-to-Face, Telephone

The communication channels used in software development can be loosely categorized in three categories [3]: **1) Non-digital media** refers to the types of communications occurring through traditional channels, such as voice and paper; **2) Digital media** is an all encompassing term covering all online communications, including mobile and web; and **3) Digital and socially-enabled media** is a subset of digital media that is based on platforms where participation is transparent, in terms of identity, content, and interactions. The digital (e.g., mailing list) and non-digital (e.g., phone) channels can be easily distinguished. However, it is not obvious to label channels as socially-enabled. A communication channel is considered socially-enabled if it promotes participation with a transparent identity (e.g., user profiles, time of last activity, number of bugs filed, and number of comments made) [27]. We use a similar categorization as Storey *et al.* [3] to assign our studied communication channels to the three categories,

as shown in Table I. We exclude the issue tracking systems from this categorization (unless the respondents specify which specific issue tracking system they use) because it is harder to categorize an issue track system without knowing the specificities of the used tool.

IV. RESULTS

In this section, we present the approach and findings for each research question.

RQ1. What is the relationship between the number of communication channels used and the development activities?

Approach. We first explore the reasons that may impact the choice of the number of communication channels to be used. For this purpose, we analyze the survey responses associated to the open-ended questions using thematic analysis (as described in Section III-A), and we present the resulting themes. We only include in RQ1 the themes that justify the *number* of communciation channels. We present the rest of the emerging themes in the subsequent research questions, as appropriate.

Afterwards, we examine the number of channels used per development activity across different projects. To statistically compare the number of communication channels used in each development activity, we use the Scott-Knott test [28], with p-value < 0.05. Scott-Knott is a statistical multi-comparison procedure based on cluster analysis. The procedure separates the data into statistically different clusters. In our case, if two different development activities use a substantially different number of communication channels, the activities are placed within different groups, otherwise, they are placed within the same group.

Results. We identify five different factors that can impact the choice of the number of channels used in the projects. The thematic analysis performed on the developers' responses generates the following overarching themes regarding the number of channels used: 1) reach, 2) standards, 3) traceability, 4) organization, and 5) not needed. We show in Table II the list of themes, along with the definition of each theme, examples of codes, and the number of times a theme appeared in the survey responses.

Development activities require a number of communication channels ranging from 1.5 to 2.9, in average. The maximum reported number of channels used is 9, appearing in release planning, documentation, and recruiting developers. The minimum number of channels used reported is 0, appearing in user support when no user support is provided. The 7 development activities can be divided into 4 statistically significantly different groups, as indicated by the Scott-Knott test. Table III shows the significantly different groups of development activities, along with descriptive statistics of the number of communication channels used for each activity. Based on the developers' feedback about 129 projects, it appears that different development activities require a slightly different number of channels for information propagation and collaboration.

The majority of developers report that they find a certain number of communication channels to be *very effective* or *extremely effective*, as shown in Table III. For release planning, 48% of the respondents report the number of channels to be effective, while 27.2% find it extremely effective. As a justisfication to this rating, *R75* recommends "*to announce releases in multiple places to hit large chunks of users, who have different preferences for how to receive communications)*." On the other hand, activities, such as user support and documentation, average the use of $\sim$2 communication channels. In regard to user support specifically, a high number of communication channels may reduce satisfaction. The respondent *R10* elaborates on the usage of 4 channels for the user support activity: "*customer feedback needs to percolate through several communication walls before it reaches developers, which is not very effective.*" As seen in Table III, recruiting developers appears to have a more balanced distribution across the five satisfaction ratings. With an average of 1.8 channels used to recruit developers, we conjecture that the recruitment process requires more aggressive information propagation to reach as many candidates as possible, especially in open source projects where the contributors are self-selecting. Indeed, *R11* confirms that "*Finding people who really care about open source (and are available) is hard, so we need wide reach to gather a valid pool.*" Finally, the code review activity uses the least number of channels (1.5 on average), a number that over 80% of respondents find *very* to *extremely effective*. With a lower number of channels in code review, *R7* explains that "*all the discussion can be captured in one place and comments/approvals are added easily*". *R22* further reports that a lower number of channels "*reduces (the) overhead of communication and allows clear traceability of the information*"

RQ2. Social, digital, or non-digital channels: why is each one used and is the choice dependent on the development activity?

Approach. Similarly to RQ1, we apply thematic analysis to the open-ended survey questions to identify the themes that justify the use of each type of communication channels (e.g., socially-enabled). We report in Table IV the resulting themes, as well as their respective frequencies in the survey reponses.

Next, we assign to the studied projects (129 in total from the responses) the counts of social-enabled, digital, and non-digital communication channels for each of the 7 investigated development activities. To capture the dominant category of communication channels used in a development activity, we further assign a final category to each activity in every project (e.g., mostly social, or equally social and digital). For example, a project that uses, in the context of release planning, one (1) socially-enabled channel, two (2) digital channels, and zero (0) non-digital channels will be assigned the category *mostly digital*. We then examine the frequency of the dominant categories per development activity.

Finally, we study the dependence between the development activities (e.g., release planning) and the assigned dominant categories (e.g., mostly social). We test the following null hypothesis: H^2*: The category of communication channels used for a given development activity does not depend on said*

TABLE II: Number of used communication channels (↗ = higher, ↘ = lower)

Themes	Definition and *examples of codes* (highlighted in italic)	% mentioned*
Reach	↗ number of channels is needed to *notify* and *reach* all interested/involved parties from both the developer and non-developer communities, when needed (e.g., to announce a new release)	43%
Standards	The decision on the number of channels can be either a *personal preference*, a *company standard*, or a *customer preference*.	41%
Traceability	↘ number of channels is preferred when *tracking context*, *monitoring progress*, and *identifying accountability* are essential to the task (e.g., who is the owner of this code? or what is the reasoning behind this decision?)	37%
Organization	↗ number of channels is opted for when it is important to *separate artifacts* (e.g., informal discussions about an issue and the formal issue report), and for discussing design decisions with *different granularities* (e.g., slower longer term decisions on issue trackers vs. fast and small decisions on chatrooms)	19%
Not needed	In some cases, the best form of communication is no communication at all when automated processes are put in place (e.g., when continuous integration and continuous delivery is used).	11%

*The total percentages associated to the themes do not add up to 100%, because a survey response could result in zero or more themes, depending on how rich the response is.

TABLE III: The number of channels used per development activity (grouped based on Scott-Knott results) and the associated developers' satisfaction.

Group	Activity	Number of channels used				Percentage distribution of the satisfaction ranging from 1 (not effective at all) to 5 (extremely effective)				
		Mean	Median	Min	Max	1	2	3	4	5
G1	Release planning	2.9	3	1	9	1.6%	6.4%	16.8%	48%	27.2%
G2	Bug fixing	2.4	2	1	7	0%	2.4%	10.3%	43.7%	43.7%
	User support	2.1	2	0	8	4.5%	8.9%	28.6%	37.5%	20.5%
G3	Documentation	1.9	1	1	9	4.9%	7.8%	27.2%	35.9%	24.3%
	Recruiting developers	1.8	1	1	9	17%	17%	32%	20%	14%
G4	Project promotion	1.6	1	1	8	8%	12.6%	35.6%	31%	12.6%
	Code review	1.5	1	1	6	3.3%	6.7%	8.3%	40.8%	40.8%

TABLE IV: Communication channel categories

Category	Associated themes	% times mentioned*
Socially-enabled	Informality	61%
	Fast turnaround	44%
	Wide reach	29%
Digital	Widely-accepted/Simple	76%
	Targeted recipients	57%
	Easy moderation	25%
Non-digital	Optimal	74%
	Conflict-free	45%
	Volatile	22%
	Unfeasible	12%

*The percentages total associated to the themes for each category do not add up to 100, because a survey response could result in zero or more themes, depending on how rich the response is. In some cases, no justification of the channel is provided by the respondents

activity. To test the hypothesis, we use the Chi-squared test of independence [29] between the development activities and the assigned category for the communication channels. We want to investigate whether the different development activities have different requirements in terms of the dominant category of channels used. For instance, we conjecture that the code review process may require more social features to be effective. Indeed, social features have been correlated with a quicker integration of pull requests [30]. We further conjecture that the user support activity may be best handled with a combination of social, digital, and non-digital to accommodate all types of users. For example, some users could be best assisted over the phone, when immediate attention is required.

Results. The thematic analysis reveals clear and distinct uses for the different types of communication channels. Socially-enabled channels, such as Slack and Gitter, are used to allow for informal communication, a faster turnaround, and to achieve wider reach. Slack, specifically, has been praised by *R100* because it solves an issue they believe is time wasteful. In their own words: "*getting an LGTM* [Looks Good To Me] *for pull requests might get an enormous amount of time, just because reviewers forget them. This is by far the bottleneck of the development process.*" Slack is being effectively used in such instances to get the process to move faster. The digital channels, on the other hand, are widely-accepted and simpler to use. They are best suited when the recipients are known, and the communication requires low to no moderation. For example, *R69* mentions that the digital channels, such as email and the mailing lists "*tend to be more thought out, and can keep conversations together with attachments for later reference*". Lastly, optimality and absence of conflict are the strong suits of the non-digital channels (e.g., face-to-face). The other side of the coin is the volatility of the information exchanged, and the infeasibility in cases of remote work.

The frequency analysis of the dominant categories reveals that the socially-enabled communication channels are the most used for the development activities, except for user support. The code review process is the most (69.4%)

and second most (77.3%) 'social' activity in the open and private repositories, respectively. The social features enable a transparent identity, which is important in code review as it promotes accountability. The digital channels are the most frequent in the user support process for both public and private repositories, as reported by 30.8% and 28.5% of the respondents for public and private repositories, respectively. In the second place comes the socially-enabled channels (for both repository types), and the non-digital channels (for the private repositories only). As conjectured, the user support process needs to accommodate the needs of the users, by adapting the choice of the communication channels to the users' preferences.

We find that there is a dependence between the development activities and the types of communication channels used by the projects. The Chi-squared dependence test confirms the dependence between the development activity and the category of channels, for both the open and private repositories (p-value $<<$ 0.001). Therefore, we reject the null hypothesis H^2, and conclude that different activities are catered to using different types of communication channels.

RQ3. What are the combinations of channels that work best?

Approach. First, we report further themes from the thematic analysis, specifically related to justifying the uses of the specific communication channels. We also compute the number of times that the themes were mentioned by the respondents. We show in Table V a summary of the themes associated with each of the communication channels considered in this study. The themes reflect the reasons for (not) choosing the channels.

Then, we identify the most frequent combinations of channels. We use the Apriori algorithm [31] to identify the most *frequent sets* of communication channels. A *frequent set* is defined as a set with a support and confidence greater than the specified minimum values. A maximally frequent set is a frequent set which is not contained in another frequent itemset. To extract the frequent sets of channels, we set the support to 0.1 (i.e., a set has to appear in at least 10% of the survey responses), and the confidence to 0.8 (i.e., channels *X* and *Y* from a frequent set are used together 80% of the time).

To identify the most suited frequent channel sets in the release planning, bug fixing, and code review activities, we compute the performance metrics, listed in Section III-B. We then examine the association between the frequent channel sets and the performance metrics and test the following null hypothesis: H^3*: The presence or absence of a set of communication channels in a project has no association with the performance metrics of the projects.*

To test the hypothesis, we use the Mann-Whitney U test [32]. The test is used to compare the distribution of a performance metric, when a certain frequent set of channels is used, against the distribution of that same performance metric without the set of channels. To assess the magnitude of the possible difference, we compute the effect size using the Cliff's delta [33], which is a non-parametric measure. The Cliff's delta ranges from -1 (i.e., all values in the first distribution are higher than the second) to +1 (i.e., all values in the first distribution are lower than the second). We interpret the effect size eff as small for $0.147 < eff < 0.33$, medium for $0.33 < eff < 0.474$, and large for $eff \geq 0.474$, by following the guidelines from prior work [34].

TABLE V: Thematic analysis for the studied communication channels.

	Communication channels	**Associated themes (% of times the theme appeared*)**
Socially-enabled	Pull requests	Recording decision process[55%] Planning future additions[45%] Providing context to bug fixing[30%] Slow progress[20%]
	Slack/Gitter	Synchronous communication[35%] Addressing delays[30%] Lost knowledge[23%] Prioritization[17%] Informal requirement development[10%]
	Google hangouts	Video call meetings[36%]
	Twitter/Facebook	Wide reach[33%] User support[22%] Lightweight[21%] Volatile[18%]
	StackOverflow	User support[16%] Documentation resource[29%]
Digital	Email/Mailing lists	Long-format design issues[56%] Specific recipients[46%] Addressing delays[37%] Widely accepted[33%]
	IRC	Widely accepted[50%]
Non-digital	Face-to-face/Phone	Optimal decision making[74%] Conflict-free[45%]

*The percentages total associated to the themes for each communication channel do not add up to 100, because a survey response could be used to derive zero or more themes, depending on how rich the response is. In some cases, no justification of the channel is provided

Results. The release planning, bug fixing, and documentation activities are more likely to use a central communication channel that appears in the most frequent sets. In the case of *release planning*, the email appears to be a central channel to communicate releases (as shown in Table VI). In combination with emails, the projects communicate face-to-face, with pull requests, with Slack, or with issue tracking systems. As revealed by our respondents, issue tracking and pull requests are a preferred way to **plan future additions** of features, and to record the decision process. However, the additional **communication overhead** in a pull request occurs through email and face-to-face. In release planning, Slack (or similar) is suited for **announcements**, and for quick exchanges to solve **ephemeral issues** that do not need to be revisited later. In both *bug fixing* and *documentation*, the issue tracking system is the central medium of communication, and is frequently paired with pull requests, Slack or email. In regards to these possible combinations, our survey respondents explain that although issue tracking systems are effective for **bugs collection**, they can be inadequate for **prioritization**. For instance, *R58* explains that: "*there are lots of bugs reported [in the issue tracking system] more than a year ago, with no*

TABLE VI: Frequent sets of communication channels for every development activity

Development activity	Frequent sets	Support
Release planning	1- {**Email**,Face to face}	0.15
	2- {**Email**,Pull requests}	0.15
	3- {**Email**,Slack}	0.14
Bug fixing	1- {**Issue tracking systems**,Pull requests}	0.34
	2- {**Issue tracking systems**,Slack}	0.16
	3- {**Issue tracking systems**,email}	0.15
Code review	1- {Face to face}	0.22
	2- {Slack}	0.13
	3- {Issue tracking systems,Pull requests}	0.12
User support	1- {Email,Issue tracking systems}	0.21
	2- {Face to face}	0.13
	3- {Mailing lists}	0.12
Recruiting developers	1- {Email,Face to face}	0.12
	2- {Issue tracking systems,Pull requests}	0.10
Project promotion	1- {Twitter}	0.24
	2- {Face to face}	0.20
	3- {Email}	0.18
Documentation	1- {**Issue tracking systems**,Pull requests}	0.21
	2- {**Issue tracking systems**,Slack}	0.12
	3- {**Issue tracking systems**,Email}	0.10

record of why they have never been addressed". Therefore, Slack (or similar) is used as a **synchronous**, **fast**, and **non-invasive** form of communication for back and forth discussions around the bugs, and for obtaining further information on how to address a bug. Email is best suited for **long-format** discussions of **design aspects**, when the right approach for fixing a bug is not obvious. In some cases, the bugs are informally **reported** by email or on Slack, before the formal creation of the bug report. In bug fixing, pull requests are linked to the bugs and are associated to tests that confirm the bugs. The linking allows to keep track of the **history of a bug fix**, with confidence and with **context**. In terms of *developers' recruitment*, we observe two distinct types of channels combination that reflect *a)* a *traditional* recruitment strategy (i.e., email and face-to-face), and *b)* **contribution-based** recruitment strategy which considers prior contributions of a developer for recruitment purposes.

In code review, user support, and project promotion, it is less likely to observe frequently-used combinations of channels across the projects. In the code review, user support, and project promotion activities, the maximally frequent sets generated by the Apriori algorithm are mostly composed of a single channel only (as shown in Table VI). In *code review*, for instance, the most common combination of channels is composed by the issue tracking systems and the pull requests, for a **formal** and **structured** code review process. In the issue tracking systems and pull requests combination, the projects further use face-to-face, Slack, and email communication, with no combination occurring frequently enough (i.e., with a support > 0.1 and confidence > 0.8). Although face-to-face remains the most **optimal** form of communication in the process of **decision making**, it is not always achievable. Therefore, Slack and email are especially useful when there is a serious **delay** on a given pull request to be merged by the assigned reviewer. Regarding *user support*, it is common across the projects to use email paired with the issue tracking system, to address the issues faced by the users. This combination of channels is common because it allows a **separation** between the *help* questions broadcasted using email, and the 'actionable' technical issues posted on the issue tracking system. However, email can be misused when the users submit their problems, often without sufficient details and explanation. In such cases, the use of Slack or the phone could be preferred because it allows for a **back and forth exchange**, to clarify the issues, although both channels **lack context** and require the participants to reiterate the issue details. Nevertheless, in some cases, the developers **do not have full control** of how the information is sourced, as the users could reach out in a variety of ways. Finally, it is common in *project promotion* to employ **less technical** and **lightweight** communication channels (e.g.,Twitter) to reach the largest possible audience. In other cases, preference is given to mailing lists (or blogs) because they allow for the **inclusion of more details**, and can **'survive' longer**, contrary to social media where content is **volatile**.

TABLE VII: Effect size and direction of the significant frequent channel sets on the performance metrics.

Release planning	Set 1	Set 2	Set 3
Release frequency	large (↗)	medium (↗)	medium (↗)
Bug fixing	Set 1	Set 2	Set 3
Fixing time	large (↘)	large (↘)	large (↘)
Fixing rate	small (↘)	medium (↘)	n.s
Code review	Set 1	Set 2	Set 3
Merge time	large (↘)	medium (↘)	medium (↘)
Merge rate	n.s	n.s	n.s

The use of certain channel sets is positively associated with performance metrics. In release planning, for instance, the frequency of the releases has a significant association (p-value = 1.78e-07) with the use of the set {email, face-to-face} (i.e., *Set 1* as numbered in Table VI), with a large effect size according to Cliff's delta. Specifically, projects that use *Set 1* have more frequent releases compared to the projects that do not use *Set 1*. We show in Table VII the channel sets that exhibit a significant association with the release frequency of the subject projects (along with the effect size and direction). In bug fixing, the successful sets of channels (i.e., associated with a lower bug fixing time and a higher bug fixing rate) are Sets 1, 2, and 3, with large effect sizes. Therefore, the pairing of structured tools, such as issue tracking systems, with less structured tools, such as Slack, appears to be effective for the bug fixing process. In terms of code review, face-to-face (i.e., *Set 1* in code review) is associated with a lowered merge time with a large effect size, thus confirming the efficiency of face-to-face interaction in the decision making process. Based on the obtained results, we can reject the null hypothesis H^3, and conclude that there exists an association (*not causation*) between the communication channels used and some of the performance metrics of the subject projects.

V. Discussion

In this section, we synthesize our findings to provide a set of recommendations to the project maintainers, to help in setting up an efficient communication flow.

It is best to tailor the communication channels used for each development activity. The developers' feedback reveals a strong affinity with the socially-enabled channels for most development activities, as discussed in RQ2. However, not all development activities are equally 'social.' For instance, the code review process, which by nature involves extensive discussions, is the most (69.4%) and second most (77.3%) 'social' activity in the open and private repositories as opposed to user support or developers recruitment. Therefore, it is best to tailor the choice of channels to the intended purpose (e.g., how much discussion is involved in the process?), and audience (e.g., where are the project users most active?).

For every development activity, it is important to have both formal and informal communication. Formal mechanisms are used to *maintain a permanent record* and for *long-format content*; whereas informal mechanisms are considered by our respondents as '*low friction*' and allowing a *quick feedback loop*. R86 states "*Email / lists are the most 'widely accepted,' but Twitter hits people fastest, and then Issues / PRs are usually where those invested more heavily in specific things would care to read*." Two respondents also reveal that the review time of pull requests is "by far the bottleneck of the development process." Therefore, it is essential to use synchronous and informal channels, such as Slack, to draw the attention of the parties involved when an action is required.

For accountability and traceability, using fewer channels is best. Having multiple options to reach team members or project users is valuable. However, information, such as rationale behind decisions, can drown in the midst of incessant streams of conversations. Therefore, using fewer channels is key to keeping track of the collective thought process in an organized and traceable manner. Similar to traceability, accountability can be lost if too many channels are used to communicate about some activities, such as code review.

Complementing (not replacing) traditional channels is a possible winner. The feedback from our respondents reveals that non-digital channels, such as face-to-face or phone, lead to little to no misunderstandings. The analysis of the responses indicates that an equal use of socially-enabled and non-digital communication channels is possibly the most satisfying combination. As such, it is best to find a middle ground between going 'social' and remaining 'traditional' when possible.

The use of social features should be accompanied by clearly outlined protocols of use. It comes as no surprise that going overly 'social' can be disruptive to the development process. 10 of our respondents mentioned that socially-enabled tools can open the way for unwanted, overwhelming, and irrelevant communications. It is particularly applicable for teams that do not restrict access to their communication channels (such as Gitter chatrooms). Therefore, it is recommended to outline a set of rules and broadcast it among the participants, to limit unwanted and intrusive behaviours.

VI. Threats to Validity

Our survey inclusion criterion is activity in the GitHub social coding platform. This possibly suggests a bias towards the developers that favor the use of the socially-enabled platforms (e.g., GitHub) in the open repositories. Fortunately, 17.8% of our respondents are most active in private repositories, giving a glimpse of the workings of the private projects that either limit the access of their repositories to the members, or are corporate entities that do not adopt social coding platforms. In terms of demographics, our respondents have mostly over 10 years of software development experiences (50.3%), and maintain the projects (51.2%). Hence, our findings may be biased in terms of showing the perceptions of the more experienced developers (e.g., maintainers).

The data collected about the use of the communication channels is binary, i.e., a team uses a channel X (or not) for a given activity. However, the data collected does not reflect the amount of communication occurring over a channel X. To address part of this concern, we have specified in our survey questions that we are interested in the *primary* communication channels used.

The low response rate to our survey (5.37%) is a limitation of our results, as the sample might not be representative of the entire population. There are two elements that mitigate part of this risk. First, the responses are associated to 129 different projects, providing insights regarding a large pool of projects. Second, despite collecting feedback from one respondent about each of the 129 projects, the respondents are confident about their knowledge of the development activities within their projects. 88% of the respondents report that they are either *familiar* or *very familiar* with the development activities included in the survey.

Despite the popularity of the Likert scale ratings, it has issues such as the assumption of an even distance between the various points (e.g., distance between *good* and *excellent* is similar to the distance between *neutral* and *good*). Respondents may also choose the midline response on a Likert scale in case they feel confused. To offset these challenges, we refrain from using statistical tests to evaluate the scales as normally distributed, parametric data. To mitigate the subjectivity inherent to the Likert scale, we ask the respondents follow-up open-ended questions to justify or clarify their ratings. Lastly, in RQ3, we observe significant associations between specific combinations of channels and certain performance metrics. The performance of projects is a result of a complex mix of factors, both technical and social. Therefore, we do not claim that the choice of communication channels can fully explain the performance of projects, and only report the observed significant associations to highlight potential successful combinations of channels.

VII. Conclusion

In this study, we aim to capture the experience of developers in using a set of communication channels to perform development activities. We examine the feedback from the survey respondents from two perspectives: *a)* the complexity of the

communication in terms of the number of channels used, and *b)* the nature of the channels used. We find that the different development activities call for personalized communication flows, in terms of both the number of channels used and the nature of the channels. Most development activities show higher satisfaction with a mix of socially-enabled and non-digital channels. We also identify the most frequent combinations of channels used by the developers, and assess their impact on a set of performance measures. Our work reveals that setting up the communication flow of software projects must not be an ad-hoc activity (e.g., choosing a tool that is trending), but rather a thought-out process that considers the particular needs of the project and all the stockholders involved. A recurrent theme that has emerged related to the use of the socially-enabled channels is the volatility and overload of information. In future studies, we hope to investigate possible solutions to consolidate the knowledge shared on such channels.

REFERENCES

[1] M. Storey, A. Zagalsky, F. F. Filho, L. Singer, and D. M. German, "How social and communication channels shape and challenge a participatory culture in software development," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 185–204, Feb 2017.

[2] J. D. Herbsleb and D. Moitra, "Global software development," *IEEE software*, vol. 18, no. 2, pp. 16–20, 2001.

[3] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (r) evolution of social media in software engineering," in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 100–116.

[4] R. L. Daft and R. H. Lengel, "Organizational information requirements, media richness and structural design," *Management science*, vol. 32, no. 5, pp. 554–571, 1986.

[5] O. Baysal, R. Holmes, and M. W. Godfrey, "No issue left behind: Reducing information overload in issue tracking," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 666–677.

[6] M. Goldman, G. Little, and R. C. Miller, "Real-time collaborative coding in a web ide," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 155–164.

[7] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM, 2004, pp. 72–81.

[8] P. C. Rigby and M. Storey, "Understanding broadcast based peer review on open source software projects," in *2011 33rd International Conference on Software Engineering (ICSE)*, May 2011, pp. 541–550.

[9] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in open source software development mailing lists," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 277–286.

[10] Y. Dittrich and R. Giuffrida, "Exploring the role of instant messaging in a global software development project," in *2011 IEEE Sixth International Conference on Global Software Engineering*. IEEE, 2011, pp. 103–112.

[11] E. Shihab, Z. M. Jiang, and A. E. Hassan, "On the use of internet relay chat (irc) meetings by developers of the gnome gtk+ project," in *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 2009, pp. 107–110.

[12] B. Lin, A. Zagalsky, M. Storey, and A. Serebrenik, "Why developers are slacking off: Understanding how software teams use slack," in *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, ser. CSCW '16 Companion. New York, NY, USA: ACM, 2016, pp. 333–336. [Online]. Available: http://doi.acm.org/10.1145/2818052.2869117

[13] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2011, pp. 2857–2866.

[14] C. Parnin, C. Treude, and L. Grammel, "Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow," georgia institute of technology," Tech. Rep., 2012.

[15] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *2013 International Conference on Social Computing*, Sep. 2013, pp. 188–195.

[16] L. Singer, F. Figueira Filho, and M.-A. Storey, "Software engineering at the speed of light: how developers stay current using twitter," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 211–221.

[17] M. E. Mezouar, F. Zhang, and Y. Zou, "Are tweets useful in the bug fixing process? an empirical study on firefox and chrome," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1704–1742, Jun 2018.

[18] T. Turner, P. Qvarfordt, J. T. Biehl, G. Golovchinsky, and M. Back, "Exploring the workplace communication ecology," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 841–850. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753449

[19] S. Black, R. Harrison, and M. Baldwin, "A survey of social media use in software systems development," in *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*, ser. Web2SE '10. New York, NY, USA: ACM, 2010, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1809198.1809200

[20] R. Giuffrida and Y. Dittrich, "A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams," *Information and Software Technology*, vol. 63, pp. 11 – 30, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491500049X

[21] ——, "Empirical studies on the use of social software in global software development – a systematic mapping study," *Information and Software Technology*, vol. 55, no. 7, pp. 1143 – 1164, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584913000153

[22] D. A. da Costa, S. McIntosh, C. Treude, U. Kulesza, and A. E. Hassan, "The impact of rapid release cycles on the integration delay of fixed issues," *Empirical Software Engineering*, pp. 1–70, 2018.

[23] C. Treude, F. Figueira Filho, and U. Kulesza, "Summarizing and measuring development activity," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 625–636.

[24] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[25] T. Tilley, R. Cole, P. Becker, and P. Eklund, "A survey of formal concept analysis support for software engineering activities," in *Formal concept analysis*. Springer, 2005, pp. 250–271.

[26] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and software technology*, vol. 46, no. 4, pp. 243–253, 2004.

[27] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 2012, pp. 1277–1286.

[28] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman, "Scottknott: a package for performing the scott-knott clustering algorithm in r," *TEMA (São Carlos)*, vol. 15, no. 1, pp. 3–17, 2014.

[29] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[30] G. Zhao, D. A. da Costa, and Y. Zou, "Improving the pull requests review process using learning-to-rank algorithms," *Empirical Software Engineering*, pp. 1–31, 2019.

[31] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[32] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[33] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[34] K. J. Goulden, "Effect sizes for research: a broad practical approach," *Journal of Developmental & Behavioral Pediatrics*, vol. 27, no. 5, pp. 419–420, 2006.