Appendices

Appendix A The statistics of the experiment projects

Table 1. The statistics of the experiment projects.

project name	number of prs	percentage of the prs containing issue report ids	median number of active reviewers for each pr
acemod_ACE3	3625	0.211862069	64
adobe_brackets	5275	0.375924171	52
akka_akka	7500	0.312666667	64
alphagov_govuk-puppet	4523	0.008180411	70
angular_angular	10214	0.270315254	189
ArduPilot_ardupilot	5800	0.072586207	90
aspnet_EntityFramework	4450	0.453483146	48
aspnet_Mvc	3423	0.371311715	49
atom_atom	4025	0.340124224	89
Automattic_jetpack	6075	0.297777778	103
Azure_azure-content	8772	0.016529868	376
CartoDB_cartodb	6300	0.529365079	43
cfpb_cfgov-refresh	4150	0.044819277	39
chapel-lang_chapel	9149	0.09859001	30
chef_chef	5474	0.120569967	112
cisco_openh264	2475	0.017777778	18
cms-sw_cmsdist	3850	0.052467532	45
department-of-veterans-affairs_vets-website	7072	0.271634615	77
diaspora_diaspora	3350	0.264477612	88
docker_docker	19166	0.26343525	491
DynamoDS_Dynamo	6449	0.106838269	32
eclipse_che	5424	0.530051622	51
EFForg_https-everywhere	13650	0.10967033	50
elastic_kibana	13600	0.265735294	118
elastic_logstash	5100	0.221960784	68
Elgg_Elgg	4375	0.336457143	26
emberjs_data	3224	0.200682382	187
emberjs_ember.js	7623	0.204250295	219
emberjs_website	2975	0.078991597	63
facebook_react-native	7325	0.19003413	420
galaxyproject_galaxy	4550	0.206593407	44
gocd_gocd	2885	0.11507799	42
gradle_gradle	4100	0.265609756	57
greenplum-db_gpdb	5375	0.048	109
grpc_grpc	9664	0.212748344	88
guardian_frontend	19149	0.016554389	112
hashicorp_terraform	8173	0.299033403	269
hazelcast_hazelcast	8850	0.311977401	49
ipython_ipython	5525	0.338823529	92

Continued on next page

Table 1 – continued from previous page

project name	number of prs	percentage of the prs containing issue report ids	median number of active reviewers for each pr
JabRef_jabref	2475	0.309090909	21
jquery_jquery	2375	0.145263158	66
jquery_jquery-mobile	2150	0.313023256	69
JuliaLang_julia	14221	0.285352647	218
KSP-CKAN_NetKAN	5389	0.05771015	50
learningequality_ka-lite	2650	0.386415094	25
mamedev_mame	3699	0.00486618	59
mantidproject_mantid	6550	0.645801527	47
mapbox_mapbox-gl-native	6321	0.461477614	59
meteor_meteor	2399	0.300541892	105
Microsoft_TypeScript	8498	0.522358202	121
Microsoft_vscode	4124	0.524490786	62
Microsoft_vsts-tasks	5650	0.033097345	109
minetest_minetest	3950	0.215443038	132
molgenis_molgenis	5200	0.045769231	18
mozilla_addons-server	5900	0.382542373	48
mozilla_fxa-content-server	3299	0.598060018	29
neo4j_neo4j	9600	0.026145833	61
nextcloud_server	6220	0.253536977	96
ocaml_opam-repository	12199	0.046397246	88
odoo_odoo	16690	0.069562612	290
openlayers_ol3	5525	0.21158371	54
OpenRA_OpenRA	6211	0.333923684	75
palantir_atlasdb	2675	0.131214953	41
pingcap_tidb	6297	0.161505479	40
PrestaShop_PrestaShop	9545	0.024096386	92
rapid7_metasploit-framework	200	0.11	20
realm_realm-java	2598	0.25134719	32
RIOT-OS_RIOT	8000	0.155375	70
SatelliteQE_robottelo	4100	0.225121951	23
scikit-learn_scikit-learn	6324	0.399905123	102
scipy_scipy	4175	0.316407186	67
SciTools_iris	2200	0.14	25
silverstripe_silverstripe-framework	5750	0.177913043	80
spree_spree	5448	0.126651982	100
vmware_vic	3750	0.4944	34
wordpress-mobile_WordPress-iOS	5496	0.594432314	34
xbmc_xbmc	14275	0.007845884	273
Yoast_wordpress-seo	3200	0.540625	27
zooniverse_Panoptes-Front-End	2625	0.39847619	32
zurb_foundation-sites	3799	0.239273493	92

Appendix B The calculation of the features

B.1 File path similarity feature

Algorithm 1 shows the detailed steps to compute the file path similarity feature denoted as $FpSim(PR_{new}, RE)$. It takes a new pull request (i.e., PR_{new}) and an active reviewer (i.e., RE) as input. First, the algorithm searches the previous pull requests that are reviewed by the reviewer before the submission time of the new pull request (i.e., PR_{new}) (Line 1). For each previous pull request (i.e., PR_{prev}) reviewed by the reviewer, the algorithm computes the file path similarity between the previous pull request (i.e., PR_{prev}) and the new pull request (i.e., PR_{new}) (Lines 2 to 13). The algorithm retrieves the file paths (i.e., $FilePaths_{prev}$) of the previous pull request (i.e., PR_{prev}) (Line 4) and the file paths (i.e., $FilePaths_{new}$) of the new pull request (i.e., PR_{new}) (Line 5). Next, For each file path (i.e., fp_{new}) from the new pull request (i.e., PR_{new}) and each file path (i.e., fp_{prev}) from the previous pull request (i.e., PR_{prev}), the algorithm computes the similarity. Then, the similarity between the previous pull request (i.e., PR_{prev}) and the new pull request (i.e., PR_{new}) is the sum of the similarities of all possible pairs of file paths (Line 7 to 11). After obtaining the file path similarities between all previously reviewed pull requests and the new pull request (i.e., PR_{new}), we sum the similarities to compute the file path similarity feature for the active reviewer regarding the new pull request (Lines 14 to 15).

To compute the similarity between two file paths, i.e., a file path (i.e., fp_{new}) of the new pull request (i.e., PR_{new}) and a file path (i.e., fp_{prev}) of the previous pull request (i.e., PR_{prev}), the algorithm first splits the file paths into components (i.e., directories and file name) using the slash character ('/') as the delimiter. Then, the algorithm computes the longest common prefix components between the two file paths. The longest common prefix is the number of common components that occur in both file paths from the root directory to the file name. Next, the value of the longest common prefix is normalized by the maximum number of components of the file paths (i.e., fp_{new} and fp_{prev}). For example, given a file path for a new pull request fp_{new} = "*lib/chef/fs/file_system/cookbooks_dir.rb*" and a file path for a previous pull request fp_{prev} = "*lib/chef/formatters/doc.rb*". The common directories of the two file paths are "*lib/chef/*" and the length is 2, which signifies that the longest common prefix is 2. The number of components in the file paths fp_{new} and fp_{prev} are 5 and 2, respectively. Thus, the maximum number of components is 5. Finally, the similarity between the file paths fp_{new} and fp_{prev} is $\frac{2}{max(5,2)} = 0.4$.

B.2 Textual similarity feature

First, for an active reviewer, we compute the textual similarities between the previously reviewed pull requests and the new pull request. Next, we sum the similarities to compute the textual similarity feature for the active reviewer. To compute the textual similarities between pull requests, we use the vector space model to represent each pull request as a vector of term weights. Next, for each pair of pull requests (i.e., one pull request from the previously reviewed pull requests

Algorithm 1 Calculate the value of the file path similarity feature

INPUT:
PR_{new}: The new pull request.
RE: A reviewer in the project
OUTPUT: file path similarity feature of RE to PR_{new}: FpSim(PR_{new}, RE)
Method:

historyReviews = A list of previous pull requests reviewed by RE before PR_{new} submission
for Review $PR_{prev} \in$ historyReviews **do**
  Sim = 0
  $FilePaths_{prev}$ = getFilePaths(PR_p)
  $FilePaths_{new}$ = getFilePaths(PR_{new})
  # Compute file path similarity between PR_{new} and PR_{prev}
  for $fp_{new} \in FilePaths_{new}$ **do**
    for $fp_{prev} \in FilePaths_{prev}$ **do**
      Sim = Sim + $\frac{prefixCommon(fp_{new}, fp_{prev})}{Max(Length(fp_{new}), Lenght(fp_{prev}))}$
    end for
  end for
  Scores[PR_{prev}] = Sim
end for
FpSim(PR_{new}, RE) = sum(Scores)
return FpSim(PR_{new}, RE)

and the new pull request), we use the cosine similarity between the vectors of the two pull requests to represent the textual similarity between them.

To represent pull requests as vectors, we first process the textual content of pull requests to remove stop words and perform stemming using the Python gensim[1] library. Then, we build the vector space model and produce vectors to represent pull requests using the Python sklearn library[2]. The vector of each pull request consists of the weights of the terms in the pull request. We capture the weight of each term using term frequency-inverse document frequency (tf-idf) as shown in Equation 1.

$$tf - idf(t, pr, PRs) = log(\frac{n_t}{N_{pr} + 1}) * log\frac{N_{PRs}}{pr_t \in PRs : t \in pr_t} \tag{1}$$

where t is a term in a pr, and PRs is the corpus of all pull requests in a project. n_t and N_pr are the number of occurrences of a term t in a pr and the total number of terms in a pr, respectively. N_PRs is the total number of pull requests.

[1] https://radimrehurek.com/gensim/
[2] https://scikit-learn.org/stable/