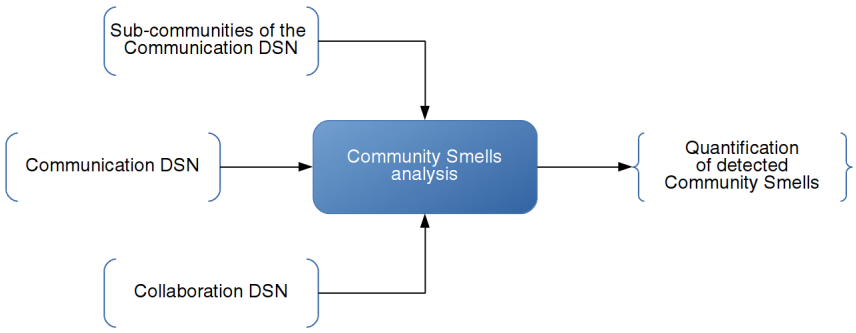

General functioning of the identification of Community Smells

Organisational Silo Effect Community Smell measures the number of collaboration links characterised by the absence of at least one of the two developers, constituting the link, within the communication channel; therefore, the identification of such Community Smell requires as input the collaboration and communication Developer Social Networks. The list of non-communicative developers is obtained performing the complement of the collaboration DSN with respect to the communication DSN and appropriate handling functions are implemented to consider a collaboration between two non-communicative developers only once. When every collaboration belonging to non-communicative developers are counted, the number of total collaboration links in which one or both of the considered developers are absent in the communication DSN is returned as the measurement that characterises the Organisational Silo Effect Community Smell. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.1.

Algorithm 7.1 Operationalisation of Organisational Silo Effect identification pattern

```
1  community.smell.organisational.silo <- function (mail.graph, code.graph) {
2    ## discover develpers not present in the communication DSN
3    non.communicative.ids <- setdiff(V(code.graph)$id, V(mail.graph)$id)
4    silos <- list()
5    ## for each non communicative developer, save his collaborations
6    for (vert in non.communicative.ids) {
7      for (collab in neighbors(code.graph, V(code.graph)[V(code.graph)$id == vert])$
           id) {
8        ## if both are non-communicative count the collaboration only once
9        ## due to the undirected nature of the graph
10       if ((collab %in% non.communicative.ids) & (collab < vert)) {
11         next()
12       }
13       ## organisational silo smell detected
14       silos[[length(silos) + 1]] <- c(vert, collab)
15     }
16   }
17   return(silos)
18 }
```

Missing Links Community Smell measures the number of collaboration links that do not have a communication counterpart, therefore the identification of such Community Smell requires as input the collaboration and communication Developer Social Networks. The identification process considers every developer within the collaboration DSN and detects every collaboration link in which the two collaborating developers are present within the communication DSN but they are not connected through an edge in the communication Developer Social Network. The presence or absence of a communication link between two collaborating developers is obtained checking the presence of the other developer in the list of neighbors of the developer considered within the collaboration DSN. This initial identification phase does not consider collaborations in which one or both of implicated developers are not present in the communication channel due to optimisation reasons later explained and it applies appropriate handling functions within the collaboration links analysis in order to consider a Missing Links detection only once, due to the undirected nature of the collaboration Developer Social Network. The information about Missing Links related to non-communicative developers can be extracted from precomputed Organisational Silo Effect Community Smell, that can be passed to this Community Smell identification function as a parameter or otherwise it will be computed directly. This optimisation is possible due to the fact that, as explained in Section 4.1, Organisational Silo Effect is contained in Missing Links Community Smell. This Community Smell operationalisation returns the list of edges that were classified as Missing Links and not the number of them, therefore to obtain the measurement of Missing Links Community Smell it is necessary to count the number of elements returned by this identification process. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.2.

Black-cloud Effect Community Smell measures the number of communication links that identify unique communication points toward other sub-communities, therefore the identification of such Community Smell requires as input the communication Developer Social Network and the subdivision of its community members in different sub-communities. It is important to notice that Black-cloud Effect is a temporal related Community Smell, since it needs to consider historical information in order to identify actual Black-cloud Effect occurrences. Within every range analysis every sub-community of the communication DSN is considered once at the time and the number of outgoing edges from every sub-community are counted. If the number of outgoing communication edges is exactly one, then a "potential" Black-cloud Effect Community Smell is detected. This Community Smell identification function returns the list of communication edges classified as "potential" Black-cloud Effect Community Smell and, in order to be classified as actual Black-cloud Effect Community Smell within a range analysis, a "potential" Black-cloud Effect has to be present in the list of "potential" Black-cloud Effect of the previously analysed

Algorithm 7.2 Operationalisation of Missing Links identification pattern

```
1  community.smell.missing.links <- function (mail.graph, code.graph, precomputed.
       silo=NA) {
2    missing <- list()
3    for (vert in V(code.graph)$id) {
4      if (!(vert %in% V(mail.graph)$id)) {
5        next() # the case of one dev not present in the mailing list is handled
             later
6      }
7      for (coll in neighbors(code.graph, V(code.graph)[V(code.graph)$id == vert])$id
           ) {
8        if (coll > vert) {
9          next() # avoid to check twice a graph due to its undirected nature
10       }
11       if (!(coll %in% V(mail.graph)$id)) {
12         next() # the case of one dev not present in the mailing list is handled
               later
13       }
14       ## if a missing communication link is found, it is saved
15       if (!(coll %in% neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id == vert
             ])$id)) {
16         missing[[length(missing) + 1]] <- c(vert, coll)
17       }
18     }
19   }
20
21   ## if no precoumputed organisational silo, we are done
22   if (length(precomputed.silo) == 0){
23     return(missing)
24   }
25
26   ## If organisational silo is not pre-computed, calculate it
27   if (is.na(precomputed.silo)){
28     precomputed.silo <- community.smell.organisational.silo(mail.graph, code.graph
           )
29   }
30   ## Add the missing links due to developers absence in the mailing lists
31   for (edge in precomputed.silo) {
32     missing[[length(missing) + 1]] <- edge
33   }
34
35   return(missing)
36 }
```

Algorithm 7.3 Operationalisation of Black-cloud Effect identification pattern

```
community.smell.potential.black.cloud <- function (mail.graph, clusters) {
  black.links <- list()
  memships <- membership(clusters)
  ## For every sub-community check how many edges connect it to another
  ## sub-community. If there is just one extra-cluster edge, we have
  ## a potential black cloud
  for (clust in 1:length(clusters)) {
    extra.clust.links <- list()
    for (vert in V(mail.graph)[memships == clust]$id) {
      for (neigh in neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id == vert])
          $id) {
        if (memships[V(mail.graph)[V(mail.graph)$id == neigh]] != clust) {
          extra.clust.links[[length(extra.clust.links) + 1]] <- c(vert, neigh)
        }
      }
    }
    if (length(extra.clust.links) == 1) {
      # Potential black cloud smell detected
      black.links[[length(black.links) + 1]] <- extra.clust.links[[1]]
    }
  }

  return(black.links)
}
```

range. Therefore, the measurement of Black-cloud Effect Community Smell is the number of communication links resulting from the intersection of the list of "potential" Black-cloud Effect of the actual range in analysis with the list of "potential" Black-cloud Effect of the previously analysed range, thus the Black-cloud Effect associated to the first analysed range will always be zero. The R implementation of "*potential*" Black-cloud Effect Community Smell in Codeface4Smells is proposed in Algorithm 7.3.

Prima-donnas Effect Community Smell measures the number of communication links that identify unique communication points toward other sub-communities in a situation in which two analysed sub-communities can be considered collaborating within the software source code development, therefore the identification of such Community Smell requires as input the collaboration and communication Developer Social Networks, the subdivision of community members belonging to the communication DSN into different sub-communities and the threshold needed to consider two distinct sub-communities as collaborating. *The default value of the collaboration threshold is setted to the 20% of total possible collaborations.* As explained in Section 4.3, the identification process of communication links that are possibly involved in a Prima-donnas Effect is the same as the "potential" Black-cloud Effect, thus in order to enable a computational optimisation it is possible to specify a precomputed list of "potential" Black-cloud Effect Community Smell, otherwise the related Community Smell identification function will be invoked. Every communication link present in the "potential" Black-cloud Effect list is considered and the number of collaborations between the two different sub-communities identified by a "potential"

Black-cloud Effect is computed. Then, the number of total possible collaborations is computed multiplying the numbers of community members constituting the two sub-communities and if the percentage of actual inter-collaborations, thus the result of the number of collaborations between the two sub-communities over the total number of possible collaborations, is greater than the given threshold then the two sub-communities are considered collaborating within the software development activity and a Prima-donnas Effect is effectively detected. Therefore, Prima-donnas Effect identification function returns the list of communication links of the communication DSN that identify the occurrence of such Community Smell and in order to obtain the related measurement it is necessary to count the number of elements that constitute the returned list. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.4.

Radio Silence Community Smell measures the number of unique knowledge brokers toward different sub-communities, therefore the identification of such Community Smell requires as input the communication Developer Social Network and the subdivision of its community members into different sub-communities. The identification process considers one by one every sub-community of the communication Developer Social Network and considers every outgoing communication link toward other sub-communities. If a sub-community is composed by only one community member, he or she is considered a unique boundary spanner without further computations, otherwise the analysis continues considering two sub-communities at the time and, if one sub-community communicates with the other one through only one community member, him or her is identified as a knowledge broker and a Radio Silence Community Smell is detected. Therefore, Radio Silence identification function returns the list of unique knowledge brokers within the sub-communities belonging to the communication Developer Social Network and in order to compute its associate measurement it is necessary to count the number of elements that constitute the returned list. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.5.

7.4 Socio-technical Quality Framework implementation

This section explains how the 40 socio-technical quality factors that constitute our Socio-technical Quality Framework proposed in Chapter 5 are implemented in CodeFace4Smells. The complete set of socio-technical quality factors is summarised in Table 5.1 and it is computed for every analysed range and obtained measurements are summarised in a socio-technical analysis report generated at the end of the global analysis performed by Codeface4Smells.

Community dimensions. Some community dimensions that consider the number of developers and members who are involved within the software project

Algorithm 7.4 Operationalisation of Prima-donnas Effect identification pattern

```
community.smell.primadonnas <- function (mail.graph, clusters, code.graph,
    collaboration=0.2, precomputed.black=NA) {
  primadonnas <- list()
  memships <- membership(clusters)
  comms <- communities(clusters)
  ## For every potential black-cloud, check collaborations of involved sub-
      communities;
  ## if it is greater than the threshold, we have two prima-donnas

  ## if no potential black-cloud, we are done
  if (length(precomputed.black) == 0){
    return(primadonnas)
  }

  if (is.na(precomputed.black)) {
    ## If potential black-cloud is not pre-computed, calculate it
    precomputed.black <- community.smell.potential.black.cloud(mail.graph,
        clusters)
  }
  for (black.link in precomputed.black) {
    sub.comm.connections <- 0
    ## retrieve cluster identifier of the two sub-communities
    id.clust1 <- memships[V(mail.graph)[V(mail.graph)$id == black.link[1]]]
    id.clust2 <- memships[V(mail.graph)[V(mail.graph)$id == black.link[2]]]
    ## count inter-collaborations of the two sub-communities
    for (dev.clust1 in V(mail.graph)[memships == id.clust1]$id) {
      if (!(dev.clust1 %in% V(code.graph)$id)) {
        next() # ignore devs present only in the communication graph
      }
      for (dev.clust2 in V(mail.graph)[memships == id.clust2]$id) {
        if (!(dev.clust2 %in% V(code.graph)$id)) {
          next() # ignore devs present only in the communication graph
        }
        if (dev.clust1 %in% neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id
            == dev.clust2])$id) {
          sub.comm.connections <- sub.comm.connections + 1
        }
      }
      ## If the fraction of present collaborations over the total possible
          collaborations
      ## (Number of devs of clust1 * Number of devs of clust2) is greater than
      ## the given threshold then we have two prima-donnas
      tot.possible.collaborations <- length(comms[[id.clust1]]) * length(comms[[id
          .clust2]])
      if ((sub.comm.connections / tot.possible.collaborations) > collaboration) {
        ## prima-donnas effect detected
        primadonnas[[length(primadonnas) + 1]] <- c(id.clust1, id.clust2)
      }
    }
  }

  return(primadonnas)
}
```

Algorithm 7.5 Operationalisation of Radio Silence identification pattern

```
community.smell.radio.silence <- function (mail.graph, clusters) {
  brockers <- c()
  memships <- membership(clusters)
  ## consider every communication outside each cluster and if there is just one
  ## communication edge from a sub-community toward another one, we have a
  ## radio silence smell (unique boundary spanner)
  for (clust in 1:length(clusters)) {
    ## If a cluster has only one dev, he is an unique boundary spanner
    if (length(V(mail.graph)[memships == clust]$id) == 1) {
      brockers[length(brockers) + 1] <- V(mail.graph)[memships == clust]$id
      next()
    }
    extra.clust.links <- list()
    for (vert in V(mail.graph)[memships == clust]$id) {
      for (neigh in neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id == vert])
          ) {
        ## Note: neigh is the local graph vertex id, not the developer id
        if (clust != memships[neigh]) {
          ## for each outgoing edge, save the cluster developer id and the
              destination
          ## sub-community id
          extra.clust.links[[length(extra.clust.links) + 1]] <- c(vert, memships[
              neigh])
        }
      }
    }
    ## for each outgoing edge, substitute destination vertex with its community
    if (length(extra.clust.links) > 0) {
      ## change format to enable comparisons
      extra.clust.links <- matrix(unlist(extra.clust.links), ncol=2, byrow=TRUE)
      for (outClust in unique(extra.clust.links[, 2])) {
        from.dev <- which(extra.clust.links[, 2] == outClust)
        if (length(from.dev) == 1) {
          ## radio silence community smell detected
          brockers[length(brockers) + 1] <- extra.clust.links[from.dev, 1]
        }
      }
    }
  }

  return(unique(brockers))
}
```

are retrieved considering the number of nodes that constitute the global, communication and collaboration Developer Social Networks. The total number of people involved in any possible and analysable way within the considered community in a specific range (**dev**) is obtained counting the number of nodes that constitute the Global DSN, while the number of members who are present in every aspect of a FLOSS project development (**ml.code.devs**) is obtained counting the number of nodes that constitute the intersection of the collaboration and communication Developer Social Networks. Finally, the number of developers who contribute to a project's source code development but do not participate in the communication channel (**code.only.devs**) and the number of members who participate to every activities of a community with the exception of the development phase (**ml.only.devs**) are retrieved respectively, subtracting the number of members present in every community phase (ml.code.devs) to the number of nodes that constitute the collaboration DSN and subtracting the number of members present in every community phase (ml.code.devs) to the number of nodes that constitute the communication DSN. Therefore, given the measured dimensional characteristics of a FLOSS community, it is possible to retrieve the dimensions of the communication or collaboration Developer Social Networks summing two different available metrics (ml.code.devs and ml.only.devs; ml.code.devs and code.only.devs); these aggregate dimensions were not considered within this master thesis study because we considered previously listed dimensions in their disaggregate and finer grain details level. Other insights that could help a researcher to understand how a community is structured and subdivided between communication and collaboration activities, can be the identification of how community members are spread into previously classified participation typologies. In order to capture such community's characteristics, it is calculated the percentage of people involved in code source development who communicate on the project's mailing list (**perc.ml.code.devs**), people present in the mailing list but that do not commit code contributions (**perc.ml.only.devs**) and developers that contribute to the community only by committing contributions to a project's source code (**perc.code.only.devs**).

Sponsored developers. The list of developers whom are supposed to be sponsored by commercial companies or whom can be considered self-sponsored developers with respect to the project in analysis, is retrieved applying an approach proposed by Riehle et al. [8], that considers information related to every commit pushed into a project's source code within the range in analysis. *A developer is associated with a sponsored status if at least the 95% of his or her commits are executed in working time*, from 9am to 5pm (local time) and from Mondays to Fridays. In their research Riehle et al. tried different threshold combinations in order to model different working habits present world-wide and concluded that the considered definition of working time provided an accurate approximation of the concept on a global work-

ing scale. The computed list of sponsored developers is used to compute the total number of sponsored developers within the window of analysis (**sponsored.devs**) and its related ratio with respect to the total number of developers whom contribute to a project's source code (**ratio.sponsored.devs**).

Core community members. The identification of core community members of the global, communication and collaboration Developer Social Networks is founded on a methodological approach proposed in 2016 by Joblin et al. [60]. Such identification methodology considers the degree centrality measure of every developer since, in their research, it was demonstrated that *core developers exhibit a higher global centrality in the developer network* and that they are likely to coordinate with other core developers, while peripheral developers are likely to coordinate with core developers. The method proposed by Joblin et al. uses social network analysis methodologies and it was proven to provide a better reflection of developer perception rather than count-based approaches (e.g. commit count, LOC count and mail count) [60]. Codeface was already able to classify as core or peripheral a developer belonging to the collaboration Developer Social Network, but this functionality was applied only within the collaboration analysis to developers whom contributed to a project's source code and it supported only directed graphs. Since the methodologies we proposed are based on undirected graph topologies, the preexistent solution to identify core and peripheral community members involved in a project was extended to support undirected graphs and the ability to apply such classification functionality to communication, collaboration and global Developer Social Networks. Once that core community members of the collaboration, communication and global Developer Social Networks are identified, it is possible to count the number of core members present in a community for each typology of analysed network (**core.global.devs**, **mail.global.devs**, **core.code.devs**). Since the list of developers sponsored by commercial companies or self-employed is retrieved from the collaboration DSN, it is possible to compute the number of sponsored developers whom are classified as core developers within the collaboration DSN (**sponsored.core.devs**), intersecting the two relative information and computing its related ratio (**ratio.sponsored.core**) with respect to the total number of core developers present in the collaboration Developer Social Network. A deeper understanding of how core members behave within different community activities can be achieved counting how many community members are characterised by core status both in the collaboration and in the communication Developer Social Networks (**ml.code.core.devs**), how many core members of the communication DSN are not core developers in the collaboration DSN (**mail.only.core.devs**) and how many core developers of the collaboration DSN are not core members in the communication DSN (**core.only.core.devs**). These three dimensional metrics related to core members distributions within considered Developer Social Networks are then used to compute the related ratio with

respect to the total number of unique core members presents in the communication and in the collaboration Developer Social Networks and in the two different generated Developer Social Networks alone (**ratio.ml.code.core**, **ratio.mail.only.core**, **ratio.core.only.core**).

Truck number. The truck number represents the ratio of people that an activity can lose without entering into a stagnation phase. It does not exist a formal definition to calculate truck number (truck factor) [57] but within a FLOSS development community we can define as vital members associated with a core status with respect to each generated network typology. The number of core members present in the communication, collaboration and global developer Developer Social Networks previously obtained are then used to calculate the truck number relative to each network typology (**mail.truck**, **code.truck** and **global.truck**), using the following formula:

$$\textbf{Truck number} = \frac{\#peripheral\,members}{\#members} = \frac{(\#members - \#core\,members)}{\#members}$$

Turnover. Different typologies of turnover are calculated using the number of community members of the current and of the previously analysed ranges. Therefore, the turnover of the first range of an analysis will always be zero. The following formula is applied to compute turnover:

$$\textbf{Turnover} = \frac{NEEDLY}{(NEBY + NEEY)\,/\,2} * 100\%$$

Where:

- *NELDY* is the number of members who left the project in the analysed range. It is obtained counting the number of members resulting from the intersection of members of previously analysed range and members of the actual range in analysis;
- *NEBY* is the total number of members who constituted the community in the previously considered range;
- *NEEY* is the total number of members who constitute the actual range in analysis.

Turnover metrics are characterised by a temporal nature because in order to be computed they need to have access to historical development analysis information. The following typologies of turnover are calculated using previously explained formula:

1. turnover of global members (**global.turnover**);
2. turnover of collaboration members (**code.turnover**);

3. turnover of global core members (**core.global.turnover**);
4. turnover of communication core members (**core.mail.turnover**);
5. turnover of collaboration core members (**core.code.turnover**).

Temporal and geographic dispersion. The temporal and geographic dispersion of a software project is calculated as the number of different and unique time-zones involved in every source code contribution to the source code within the range in analysis (**num.tz**). Codeface's collaboration analysis populates a database table with all retrievable details of commits and their relative author, hour, date and time-zone. Codeface4Smells comes with a functionality capable to query such database table in order to retrieve all the commits information related to the range in analysis, extract their associated time-zones and return the number of unique different time-zones that were involved within the project development in the considered range.

Socio-technical congruence. Socio-technical congruence (**st.congruence**) is measured as the number of development collaborations that do have a communication counterpart over the total number of collaboration links present in the collaboration Developer Social Network. Development collaborations that do have a communication counterpart are identified analysing one by one the collaboration links that connect different developers present in the collaboration Developer Social Network and check within the communication Developer Social Network if such developers are present and connected through a communication link. Therefore, socio-technical congruence can be computed using Missing Links Community Smell metric as follows:

$$\textbf{Socio-technical congruence} = \frac{\#collaborations - \#missingLinks}{\#collaborations}$$

Communicability. Each collaboration between two developers (A and B) in the software development network is considered as a possible source of architectural and design decision, therefore a developer is considered aware of a decision if he or she is strongly connected to at least one of the two developers whom generated the decision. In-communicability is related to every collaboration within the collaboration DSN and it is based on Tamburri et al.'s formulation [DEBT-2]:

$$\text{MAI} = DEM - DAM$$

$$\text{DEM} = \frac{\#collaborators\ of\ the\ two\ developers}{\#developers}$$

$$\text{DAM} = \frac{\#collaborators\ of\ the\ two\ developers\ whom\ communicate\ with\ them}{\#developers}$$

Therefore, global in-communicability can be defined as the mean MAI over the entire collaboration network. Communicability is a global indicator which consists

in the mean of all local communicability measures, calculated for every collaboration between two developers within the collaboration Developer Social Network in the range in analysis. Communicability was preferred to in-communicability in order to simplify measurement comprehension, because in-communicability tend to be characterised by measurements that tend to zero. Communicability is computed as:

$$\textbf{Communicability} = 1 - \textit{incommunicability} = 1 - \frac{1}{n} \sum MAI$$

Social Network Analysis metrics. We used some Social Network Analysis methodologies available in R language to calculate the following factors:

- centrality of the global Developer Social Network computed considering closeness (**closeness.centr**), betweenness (**betweenness.centr**) and degree (**degree.centr**);
- density of the global Developer Social Network (**density**);
- modularity of the global Developer Social Network (**global.mod**), communication Developer Social Network (**mail.mod**) and collaboration Developer Social Network (**code.mod**).

Smelly developers and smelly quitters. Two socio-technical quality metrics related to the outcome of Community Smells identification analysis, specifically computed using the list of unique community members involved within at least one Community Smell, are the ratio of smelly developers and the ratio of smelly quitters. The ratio of smelly developers (**ratio.smelly.devs**) is the ratio of community members who are involved in at least one Community Smell with respect to the total number of unique members who constitute the global Developer Social Network. The ratio of smelly quitters (**ratio.smelly.quitters**) represents the ratio of developers who were involved in at least one Community Smell in the previously analysed range that left the software development community within the range in analysis. The ratio of smelly quitters is characterised by a temporal characteristic because in order to be computed it needs to have access to historical development analysis information, therefore a list of every community member and of smelly developers of the previously analysed range is kept and passed to the next range analysis.