Political Factions

Subtitle

Contents

1 Introduction

In this short analysis, we use the open data from the Chamber of the Deputies of the Italian Parliament to investigate the legislative co-sponsorship of the members of the Chamber of the Deputies. The analysis would cover the legislative process that ensued under Conte I and Conte II cabinets within the legislature XVIII of Italy.

The analysis aims to understand the presence of political factions underlying the legislative process in the Lower chamber of the Italian Parliament by employing social network analysis techniques. Moreover, we will analyse the cooperative behaviour of the Parliament parties computing the *intra-opposition party bill differentiation*, as shown by De Giorgi & Dias (2018).

Lastly, this project also strives to create a fully reproducible workflow documented, commented and hosted in a public Github repository.

2 Methods

The analysis is based on the research produced by De Giorgi and Dias; the study analysed the network created by the co-sponsorship of legislative acts during X and Y cabinets, in office, respectively, from 1900 to 1910 and 2016 to 2000.

The study does not provide the code used to produce the analysis nor define the software used. For these reasons, the analysis has been reproduced *ex nihilo* using R statistical computing language and its packages, combined with the provided documentation.

The data has been collected using `dati.camera.it` relying on a Virtuoso Endpoint via SPARQL language. The queries were prompted through `SPARQL` packages. The Social Network Analysis was held using `igraph` package. Data processing and preparation was carried out with a selection of packages from the Tidyverse collections. As a general rule, the code was written using the Tidyverse verbs and syntax.

3 Technical overview

3.1 Package loading

```
library(here)
library(SPARQL)
library(tidyverse)
library(vroom)
library(stringr)
library(lubridate)
library(ggplot2)
library(igraph)
```

3.2 Data collection

The data retrieval consist of two main steps: firstly, we must declare the endpoint to scrape:

```
endpoint <- "http://dati.camera.it/sparql"
```

Since the SNA needs a list of vertices and a list of edges, we scraped the information of the deputies who took office during the selected cabinets and all the bills proposed. SPARQL queries use semantic triples, which are built as a set of three entities `subject-predicate-object`. To makes things more manageable, we firstly declare a query to get the deputies information. To restrict the result only to the 18th legislature, we used the triple `?atto ocd:rif_leg <http://dati.camera.it/ocd/legislatura.rdf/repubblica_18>`.

```
bio <- "
SELECT DISTINCT (CONCAT(?cognome,\" \",?nome) AS ?name) , ?partito, ?s_office, ?e_office
WHERE { ?persona a ocd:deputato;
        foaf:firstName ?nome;
        foaf:surname ?cognome .
          ?persona ocd:rif_mandatoCamera ?mandato .

        #adesione a partito
        ?mandato ocd:rif_deputato ?deputato .
        ?deputato ocd:aderisce ?l .
        ?l rdfs:label ?partito .
    OPTIONAL{?l ocd:startDate ?s_office.}
    OPTIONAL{?l ocd:endDate ?e_office.}

        #restict to 18esima legislatura
        ?mandato ocd:rif_leg <http://dati.camera.it/ocd/legislatura.rdf/repubblica_18> .}
ORDER BY ?name"
```

The query retrieves the deputy's name, the party, and the office's dates. The dates are pivotal since a deputy could change party during the mandate; these changes must be considered when building the social network edges.

Then we could query the database:

```
SPARQL(endpoint, bio)
```

Subsequently, we retrieved the bills taking the number of the bill, the date of the first proposal, the first signatory and the joint signatories. The construction of this query encountered a significant problem: the endpoint could not provide more than 10000 results at once. So, we had to use a `subquery-offset` method to bypass this limitation. To restrict the results only to the first Conte's cabinet, we used the triple `?atto ocd:rif_governo <http://dati.camera.it/ocd/governo.rdf/g142>` as filter.

```
query_main <- "
SELECT DISTINCT ?num ?date(CONCAT(?primo_cognome, \" \",?primo_nome) AS ?signatory)
(CONCAT(?altro_cognome,\" \",?altro_nome) AS ?joint_signatory)
WHERE {
 {
  SELECT ?num ?date ?primo_nome ?primo_cognome ?altro_nome ?altro_cognome
  WHERE {
  ?atto a ocd:atto;
          dc:identifier ?num;
          dc:date ?date;
          ocd:rif_leg <http://dati.camera.it/ocd/legislatura.rdf/repubblica_18> ;
          ocd:rif_governo <http://dati.camera.it/ocd/governo.rdf/g142> .

  ?atto ocd:primo_firmatario ?primo .
  ?primo  a ocd:deputato;
            foaf:firstName ?primo_nome;
            foaf:surname ?primo_cognome .

  ?atto ocd:altro_firmatario ?altro .
  ?altro  a ocd:deputato;
            foaf:firstName ?altro_nome;
            foaf:surname ?altro_cognome .
    }
    GROUP BY ?atto
    ORDER BY ?num ?primo_cognome ?altro_cognome
}
  }

LIMIT 10000
OFFSET"
```

Then, we defined the vector with offset limits:

```
query_offset <- c("0", "5000", "10000", "15000",
                  "20000", "25000", "30000", "35000")
```

Lastly, we defined a for loop to make consecutive calls to the database, changing offset limit for each call. This method permits retrieval of more than 10000 observations:

```
for (i in 1:length(query_offset)) {
  law <- str_c(query_main,
               query_offset[i],
               sep = " ")
  result_law <- SPARQL(endpoint, law)
  df_law <- rbind(df_law, result_law$results)
  Sys.sleep(2)
}
```

Since scraping large chunks of data could be burdensome for the host, we set the offset to half of the endopoint's limit (5000 observations each), and, after every call, the loop sleeps for 2 seconds.

To retrieve the bills for the second Conte's cabinet, we used the same query structure, changing the "filter" triple to `?atto ocd:rif_governo <http://dati.camera.it/ocd/governo.rdf/g162` [from `g142` to `g162`].

Each result was exported into a CSV file after the retrieval[1]:

```
write.csv(result1, here::here("data/deputies.csv"))
write.csv(result2, here::here("data/conte_i.csv"))
write.csv(result3, here::here("data/conte_ii.csv"))
```

3.3 Data preparation

Since the resulting queries are not ready-made for the subsequent analysis, we must prepare the data. For this purpose, we could define functions for the needed purposes.

3.3.1 Deputies preparation

We define the function for preparing the deputies:

```
prep_deputies <- function(deputies, end_date = "2021-12-02") {

  deputies <- deputies %>%

  # removing office term included in the party name var party is provide as
  # "PartyX (1900-00-01-[...])" we need to remove what's after the first "("
    separate(col = partito,
             into = c("party", "trash"),
             sep = "\\(") %>%
    mutate(
      # arty column has space after the party name, then we must trim it!
      party = str_trim(party),

      # parsing offices data into a single interval format
      date = interval(start = ymd(s_office),
                      end = ymd(ifelse(test = is.na(deputies$e_office),
                                       #since the 18th leg is still in office
                                       #some MPs does not have ending date
                                       yes = 20220228,
                                       no = deputies$e_office)))) %>%

    # dropping working variables no more useful
    select(!c(trash, s_office, e_office)) %>%

    # drop deputies which have taken office *AFTER* Conte_ii cabinet
    filter(int_start(date) < lubridate::ymd(end_date))

  #reutурn value
  return(deputies)
}
```

The function's primary purpose is to deal with the query's problematic results. The query returned four raw columns:

[1] `here` package provides a relative path to the top-level directory of the project, simplifying the referencing of the data regardless of the OS or the absolute path of the directories

- name of the deputy
- party with which
- start date
- end date

The major problem is the party column because it includes the party name and the terms of the mandate, written between round brackets. With `separate()`, we can split the party name from the terms, which are unused since we have it already. Then, we parse the start and end dates into an interval object.

3.3.2 Contributions preparation

```
prep_cabinet <- function(cabinet, cabinet_int, deputies) {
  `%!in%` <- Negate(`%in%`) # "not in" function declaration

  #define cabinet name
  name <- as.character(substitute(cabinet))


   # Due to the construction of dati.camera.it database it's easier preparing
   #data within R environment:
   cabinet <- cabinet %>%

     # removing duplicate rows
     distinct(.) %>%

     # Parsing dates
     mutate(date = ymd(date))

   # Extracting deputies and contributors
   cabinet_deputies <- deputies %>%
     filter(int_start(date) < int_end(cabinet_int)) %>%
     filter(!(int_end(date) < int_start(cabinet_int)))

   # Since some MPs, may have changed party or decayed in between
   # they must be coded as SWITCHER/DECAYED
   is_duplicate <- cabinet_deputies %>%

     # group by MPs names
     group_by(name) %>%

     # n as how many time a unique name appears
     summarise(n = n()) %>%

     # filter only the names who appear more than one time
     filter(n > 1) %>%

     # coding party as "SWITCHER/DECAYED"
     add_column(party = "SWITCHER/DECAYED") %>%

     #selecting only name an party to match the structure of cabinet_deputies
     select(name, party)


   cabinet_deputies <- cabinet_deputies %>%
```

```r
    # remove switcher from cabinet_deputies with an anti joint fun
    anti_join(is_duplicate, by = "name") %>%

    #drop the date column, no more useful
    select(!(date)) %>%

    #bind the Switcher to cabinet_deputies
    bind_rows(., is_duplicate)

  #remove eventual orphan nodes
  cabinet_deputies <-  cabinet_deputies %>%
     filter(!(name %!in% c(cabinet$signatory,cabinet$joint_signatory)))

  # since there is MPs no included in the nodes df we must debug it!
  debug <- cabinet[which(!(cabinet$joint_signatory %in% cabinet_deputies$name)), ]
  debug <- unique(debug$joint_signatory)
  cabinet_deputies <- rbind(cabinet_deputies,
                            deputies[deputies$name %in% debug, 1:2])

  #assign the contributions DFs
  assign(name,
         cabinet,
         envir = parent.frame())

  #assign the deputies DFs
  assign(stringr::str_c(name, "_deputies"),
         cabinet_deputies,
         envir = parent.frame())
  }
```

The function consists of two primary purposes: first, removing the duplicates observation on the contributions dataframe. Furthermore, it deals with a significant issue: Italian members of Parliament can change the party during their mandate. This, cause the presence of duplicated names (vertices in the SNA) which must be removed.

The issue has been addressed by filtering for the deputies whose office mandate started *before* the cabinet end date and dropping the deputies whose office mandate started *after* the cabinet end date. Then, the deputies who appear more than once are saved in a temporary object, coding the `party` variables as “SWITCHER/DECAYED”. Lastly, the duplicates deputies are removed from the `cabinet_deputies` dataframe using an `anti_join()`, and then they are appended again to the dataframe. This method permits to have solely unique observations in the `cabinet_deputies` dataframe. Hence, the observations that appeared more than once are now unique, with the party coded as “SWITCHER/DECAYED”

To remove eventual orphan nodes, the function removes the observations from `cabinet_deputies` dataframe, whose `name` is not present in the contributions dataframe, `cabinet`. In order to doing so, we used a user-defined operator: `%!in%`[2]. The operator returns the values of x that do not match in y. Applied to our case using a negate filter, it filters out all the deputies who are not present in the contributors' dataframe[3].

The function addresses another issue, also; some bills have latecomers associated as joint signatories since the lengthy legislative procedure. Possibly, these latecomers have not taken office under the cabinet in analysis and must be removed. The solution is implemented in a slightly convoluted manner which must be addressed in future releases.

[2] defined as `Negate(%in%)`
[3] `filter(!(name %!in% c(cabinet$signatory,cabinet$joint_signatory))`

At its last, the function returns the resulting dataframes to the parent environment using the `assign()` verb The name of the returned objects is built upon the dataframe given as the `cabinet` argument to the function.

Defined all the functions, we can now use them finishing the data pre-processing:

```
# Deputies function
deputies <- prep_deputies(deputies)

# Conte_i contributions and dates
conte_i_date <- interval(ydm("2018-31-05"), ydm("2019-04-09"))

prep_cabinet(cabinet = conte_i, cabinet_int = conte_i_date, deputies = deputies)

# conte_ii contributions and dates
conte_ii_date <- interval(ydm("2019-04-09"), ydm("2021-12-02"))

prep_cabinet(cabinet = conte_ii, cabinet_int = conte_ii_date, deputies = deputies)
```

3.3.3 Italia Viva

A significant shortcoming was introduced with Conte's 2nd cabinet. Italia Viva party was founded (2019-09-18) fourteen days after the cabinet was sworn. For this reason, the function drop IV deputies as switchers. For this reason, we must add the deputies manually into the `cabinet_deputies` dataframe:

```
iv <- deputies %>%
  filter(party == "ITALIA VIVA") %>%
  filter(int_start(date) == ymd("2019-09-19") &
        int_end(date) > int_end(conte_ii_date)) %>%
  select(!(date))

conte_ii_deputies$party <- ifelse(conte_ii_deputies$name %in% iv$name,
                                  "ITALIA VIVA",
                                  conte_ii_deputies$party)
```

3.4 Social Network Analysis

The Social Network Analysis was held relying on the `igraph` package. In order to produce a network, we must provide a set of edges and a set of nodes. Since the data were pre-processed in the previous section, we just needed to select the two variables that contain the signatory and the related joint signatories from the contributions datasets.

```
# Edges preparations
conte_i_edges <- conte_i %>%
  select(!c(num, date)) %>%
  relocate(joint_signatory)

# Edges preparations
conte_ii_edges <- conte_ii %>%
  select(!c(num, date)) %>%
  relocate(joint_signatory)
```

Since we have already prepared the nodes' datasets, we needed to compute the networks with the `graph_from_data_frame()` function. We specify the directed relationship of the edges, which goes from the joint signatory to the related first signatory.

```
# Computating igraph network from edges df:
conte_i_network <- graph_from_data_frame(conte_i_edges,
```

```
                                        directed = T,
                                        vertices = conte_i_deputies)


conte_ii_network <- graph_from_data_frame(conte_ii_edges,
                                         directed = T,
                                         vertices = conte_ii_deputies)
```

The last step is to give meaningful colour coding based on the node party affiliation. Unfortunately, the implemented method was wordy, and no other solution then nested `if-else` statements were found:

```
# Coding the colours for Conte I network
V(conte_i_network)$color <- NA
V(conte_i_network)$color <-
  ifelse(V(conte_i_network)$party == "FORZA ITALIA - BERLUSCONI PRESIDENTE","lightblue",
  ifelse(V(conte_i_network)$party == "LEGA - SALVINI PREMIER", "forestgreen",
  ifelse(V(conte_i_network)$party == "PARTITO DEMOCRATICO","red",
  ifelse(V(conte_i_network)$party == "MOVIMENTO 5 STELLE", "yellow",
  ifelse(V(conte_i_network)$party == "FRATELLI D'ITALIA","#003366",
  ifelse(V(conte_i_network)$party == "LIBERI E UGUALI","orange",
  ifelse(V(conte_i_network)$party == "MISTO","#DDDDDD",
  ifelse(V(conte_i_network)$party == "SWITCHER/DECAYED", NA, "pink"))))))))

# Coding the colours for Conte I network
V(conte_ii_network)$color <- NA
V(conte_ii_network)$color <-
  ifelse(V(conte_ii_network)$party == "FRATELLI D'ITALIA","#003366",
  ifelse(V(conte_ii_network)$party == "MOVIMENTO 5 STELLE","yellow",
  ifelse(V(conte_ii_network)$party == "LEGA - SALVINI PREMIER","forestgreen",
  ifelse(V(conte_ii_network)$party == "FORZA ITALIA - BERLUSCONI PRESIDENTE","lightblue",
  ifelse(V(conte_ii_network)$party == "PARTITO DEMOCRATICO","red",
  ifelse(V(conte_ii_network)$party == "LIBERI E UGUALI","orange",
  ifelse(V(conte_ii_network)$party == "MISTO","#DDDDDD",
  ifelse(V(conte_ii_network)$party == "ITALIA VIVA","#C83282",
  ifelse(V(conte_ii_network)$party == "SWITCHER/DECAYED",NA, "pink")))))))))
```

3.4.1 Visualization

To visualize the resulting network, we plot it tuning the arguments to obtain the most precise result. We applied an algorithm that spread the nodes using a physical simulation to better position them. For this analysis, we emplyed the Fruchterman-Reingold (1991) algorithm.

```
# Fruchterman-Reingold layout computation:
l <- layout_with_fr(conte_i_network, niter = 2000)

# Plot network
plot(

  ## Network main arguments
  conte_i_network,
  frame = F,
  layout = l,

  ## Vertexes aes:
  vertex.size = 7,
```

```
  vertex.label = NA,
  vertex.label.cex = .7,
  vertex.color = adjustcolor(V(conte_i_network)$color, alpha.f = 0.9),

  ## Edges aes:
  edge.arrow.size = .05,
  edge.width = .3,
  edge.curved =0.3)
```

3.5 IOBD index

The last part of this section will illustrate how the *Intra-Opposition party Bills Differentiation* (IOBD) index was computed, following the definition provided by De Giorgi & Dias (2018):

$$IOBD = CSP + IPS \tag{1}$$

Where:

- CSP: the percentage of co-sponsors of bills initiated by party X that belong to the same party;
- ISP: of all the bills co-sponsored by MPs of party X, the percentage of those initiated by party X.

3.5.1 CSP

To calculate the Cosponsorship index, we defined a simple function. First, we double joint the deputies df into contributions df to extract the deputies party. Then, grouping by the party of the first signatory, we divided the numbers of joint signatories of the same party by the total of joint signatories that had contributed to the party's bills:

```
cosponsor <- function(contributions,
                      deputies, signatory = "signatory",
                      joint_signatory = "joint_signatory",
                      mp_name = "name" ) {
   contributions %>%

      # double join to get the signatory and joint signatory party
      left_join(deputies,
                by = c(signatory = mp_name )) %>%
      left_join(deputies,
                by = c(joint_signatory = mp_name),
                suffix = c("_main","_joint")) %>%

      # group by the bill and the party of the signatory and computing the csp
      group_by(party_main) %>%
      summarise(n= n(),
                n_joint = sum(party_joint == party_main)) %>%
      mutate(csp = (n_joint/n)) %>%
      summarise(party = party_main,
                csp = csp)
}
```

3.5.2 IPS

We used the same structure as the previous function to calculate the intraparty index. This time, we grouped by the party of the joint signatories. So, we calculated all bills co-sponsored by MPs of a party, the percentage of those initiated by the same party:

```r
intraparty <- function(contributions,
                       deputies, signatory = "signatory",
                       joint_signatory = "joint_signatory",
                       mp_name = "name" ) {
   contributions %>%
      left_join(deputies,
                by = c(signatory = mp_name )) %>%
      left_join(deputies,
                by = c(joint_signatory = mp_name),
                suffix = c("_main","_joint")) %>%

      # group by party of the joint signatory
      group_by(party_joint) %>%
      summarise(n= n(),
                n_joint = sum(party_joint == party_main)) %>%

      # computing the ISP index
      mutate(ips = (n_joint/n)) %>%
      summarise(party = party_joint,
                ips = ips)
}
```

3.5.3 IOBD index

At last, we defined the last function, which calculates the sum of the two indexes defined before, relying on the above-explained functions:

```r
iobd_index <- function(contributions, deputies,
                signatory = "signatory", joint_signatory = "joint_signatory",
                mp_name = "name" ) {

   # call cosponsors function
   cosponsor(contributions, deputies,
             signatory, joint_signatory,
             mp_name) %>%

   #join the result with intraparty function by party name
      left_join(., intraparty(contributions, deputies,
                              signatory, joint_signatory,
                              mp_name),
                by = c("party" = "party")) %>%

   # summarize the results by the party name and IOBD var,
   # which is the sum of CSP and IPS
      summarise(party,
                IOBD = csp+ips)
}
```

4 Discussion

4.1 Network analysis

4.1.1 Conte I

The visual analysis of the produced SNA graph permits some considerations. As can be observed in **Figure 1**, during the first Conte's cabinet the majority of the parties were sheltered to their positions. Even if the two government parties, Lega Nord and Movimento 5 Stelle, have interacted extensively between them, they still constitute two distinct communities. Only the Partito Democratico results partially blended with Liberi e Uguali community. It must be noted that this visualization did not employ any modularity computation, and the nodes (which represent deputies) has been coloured by the party affiliation.

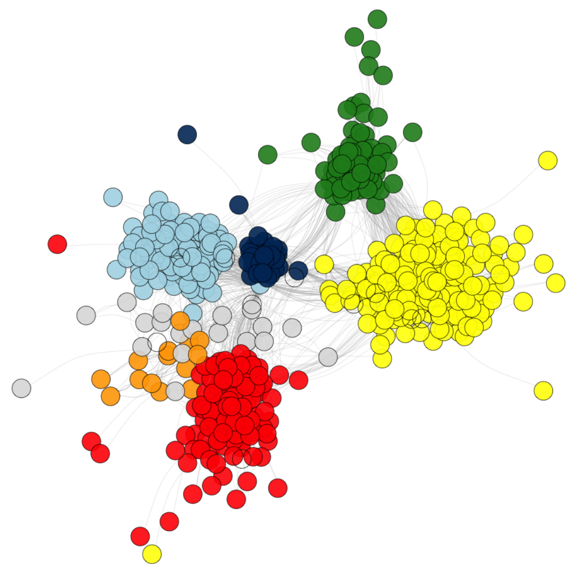

Figure 1: Conte I contributions network graph

4.1.2 Conte II

Figure 2 represent the network formed of co-authorship formed under the second Conte cabinet. The ruling parties were Movimento 5 Stelle, Partito Democratico, Liberi e Uguali, and Italia Viva. Again, we can obverse the formation of significant clusters within the Chamber of the Deputies. M5S once again had a strong relationship with other ruling parties; however, it formed a self-standing community from them. Notably, other ruling parties (PD, LeU, IV) were firmly held together, to the point that they constituted a community by themselves. As for the first network graph, we did not employ any modularity computation; therefore, the colour and the size of the nodes are provided manually and do not correspond to statistics computation.

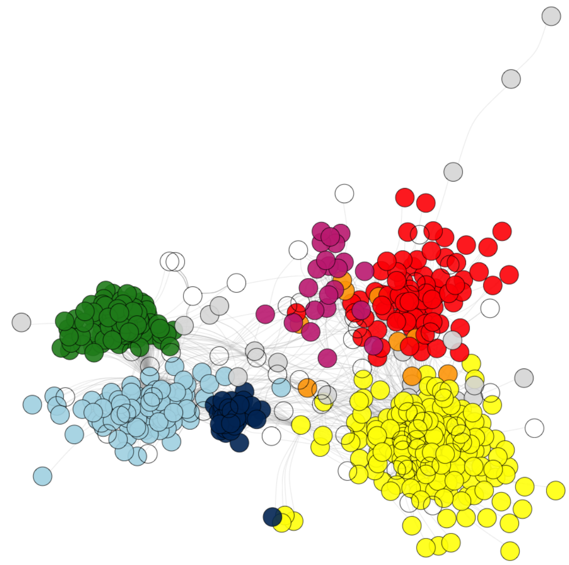

Figure 2: Conte I contributions network graph

4.2 IOBD index

For the last part of the project, we calculated the IODB index for the two periods under analysis using the defined functions. The results, shown in **Table 1**, provide evidence in favour of our previous findings.
Under Conte I, all the parties demonstrated strong ties within them. Under Conte II, the parties had strong ties within them; however, the ties between the ruling parties were stronger than Conte I cabinet.

Table 1: IOBD index

party	conteI	conteII
FORZA ITALIA - BERLUSCONI PRESIDENTE	1.830	1.709
FRATELLI D'ITALIA	1.898	1.781
LEGA - SALVINI PREMIER	1.971	1.947
LIBERI E UGUALI	1.429	0.700
MISTO	1.067	0.351
MOVIMENTO 5 STELLE	1.885	1.711
PARTITO DEMOCRATICO	1.873	1.759
SWITCHER/DECAYED	0.099	0.289
ITALIA VIVA	NA	1.508

5 Conclusion

In conclusion, in this short paper, we failed to provide enough evidence favouring our hypothesis. Nevertheless, we can assert that the parties were polarized and cohesive in the bill proposal and that no prominent political factions were noticeable.

More in-depth analyses are required to provide a more explicit demonstration. Further studies must break up the bill proposal into the different topics exposed to understand if the variation of the topic provokes a variation in the IODB index.

Moreover, a more reliable and flexible database type is needed. We allow ourselves to suggest the usage of a network database powered by a management system such as Neo4j.

Word count

Method	koRpus	stringi
Word count	1958	1950
Character count	12228	12331
Sentence count	123	Not available
Reading time	9.8 minutes	9.8 minutes

References

De Giorgi, E., & Dias, A. (2018). Standing apart together? Common traits of (new) challenger parties in the Italian parliament. *Italian Political Science*, *13*(2).

Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph Drawing by Force-directed Placement. *Software-Practice and Experience*, *21*(1).