Applying the Evolution Radar to PostgreSQL

Marco D'Ambros, Michele Lanza
Faculty of Informatics
University of Lugano, Switzerland
{marco.dambros, michele.lanza}@lu.unisi.ch

Categories and Subject Descriptors: D.2.7 [Software Engineering]: Maintenance, Version Control, Re-engineering, Reverse Engineering

General Terms: Measurements, Design.

Keywords: Evolution, Logical Coupling, Visualization.

1. GOALS

In this report we describe the results of the application of our approach, the Evolution Radar [2], on the PostgreSQL system. The mining questions we want to answer are:

1. What are the relationships among the system modules in terms of logical coupling? How are these relationships characterized? Which are the main responsibles for the logical couplings, *i.e.*, the best candidates for starting a reengineering process?

2. How have these relationships evolved over time? When have refactorings been applied on the modules? In which phase is the system in the current version?

2. INPUT DATA

To analyze the target system, *i.e.*, PostgreSQL, we use its whole history, as recorded by the CVS version control system, stored in a database called Release History Database (RHDB) [1, 3]. The database populating process, performed in batch mode, consists in (i) doing a checkout of the system, parsing it and storing the structure information in the database, (ii) parsing the CVS logs and storing all the commit-related information. The RHDB includes information about all the files in the system,*i.e.*, source code, documentation, make-files, *etc.* For our analysis we consider only the source code data, *i.e.*, `.c` and `.h` files (since PostgreSQL is written in c). We decompose the system using the top-most directories in the `src` directory tree, *i.e.*, we define a module as all the files belonging to a directory subtree.

MSR'06, May 22–23, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

3. THE EVOLUTION RADAR APPROACH

The Evolution Radar (see Figure 1) visualizes the logical coupling of one module with the others (see [2] for details). The module in focus is placed in the middle of a pie chart, where each sector represents one of the other modules. The size of each sector depicts the size of each module in terms of number of files. The modules are sorted according to this size metric.

The files of each of those modules are represented as circles and placed (and colored) according to the logical coupling they have with the module placed in the center. The closer the files are to the center (the hotter, from blue to red, the color is), the more coupled they are.

Given a module M, a file f, and a time interval (t_1, t_2), we define the logical coupling between the two as:

$$LC(M, f, t_1, t_2) = \max_{f_i \in M} \{sc(f_i, f, t_1, t_2)\} \quad (1)$$

where $sc(f_i, f, t_1, t_2)$ is the number of shared commits (performed at the same time with a tolerance of 200 seconds) between f_i and f during the time interval (t_1, t_2)[1].

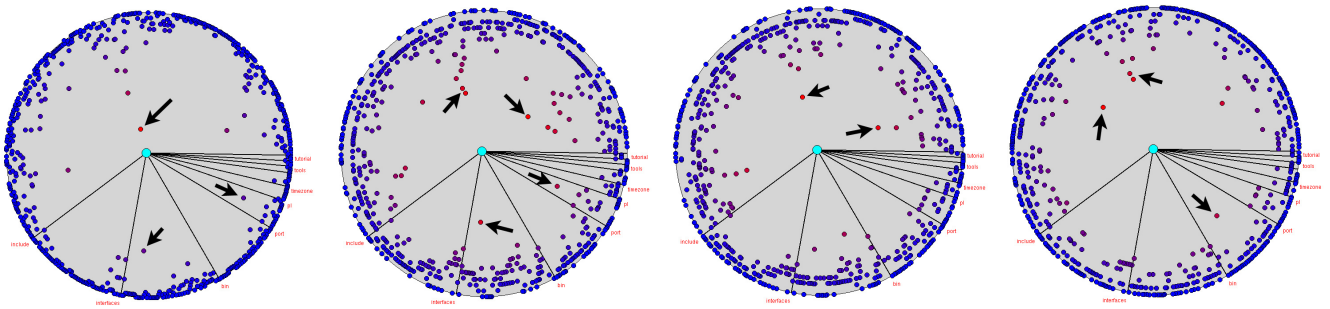
4. RESULTS

We consider the three biggest modules with respect to the number of files: backend (673 files), include (394 files) and interfaces (84 files). For each module we build four Evolution Radars (using the module as the center of the radar) corresponding to the last four years of development of the module. Then we study the relationships of the target module with the five other biggest modules in the system with respect to the logical coupling information. For this study we both analyze the view and compute some measures characterizing the evolution of the couplings. In details we define:

- **Strength (*s*)**: The total value of the logical coupling between the target module M_t and another module M (a slice). It is equal to the sum of the logical coupling of all the files of M with M_t.

- **Distribution (*d*)**. The percentage of files involved in the logical coupling. It is equal to the number of files of M having a logical coupling with M_t divided by the total number of files of M.

[1]We don't need a normalized value, *i.e.*, weighting the logical coupling with the number of commits, because we study the evolution of the logical couplings, thus we compare absolute values of logical couplings over time instead of analyzing one value only.



(a) 2002 (b) 2003 (c) 2004 (d) 2005

Figure 1: Evolution Radars for the backend module from 2002 to 2005.

- **Outliers (*ol*)**. The files of M having a logical coupling with M_t much higher than all the others. Those are detected directly on the view instead of using a formal definition.

Figure 1 shows the four Evolution Radars for the backend module (where the arrows highlight the outliers), while the computed measures and the detected outliers are listed in Table 1.

		2002	2003	2004	2005
include	*s*	832	928	957	431
	d	53%	73%	71%	45%
	ol	parsenodes.h	parsenodes.h nodes.h bufmgr.h	parse-nodes.h guc.h	parsenodes.h nodes.h executor.h
interfaces	*s*	81	161	107	95
	d	31%	81%	63%	55%
	ol	tabcomplete.c	tabcomplete.c	none	none
bin	*s*	105	212	183	91
	d	58%	84%	78%	64%
	ol	none	none	none	none
port	*s*	31	63	70	37
	d	30 %	62%	53%	38%
	ol	none	none	none	path.c
pl	*s*	36	52	38	32
	d	40%	45%	50%	40%
	ol	pl_exec.c	pl_exec.c	none	none

Table 1: Results for the backend module.

Conclusions on the Backend Module. As we can see from Figure 1 and Table 1 the backend module was initially (2002) logically decoupled from all the other modules but the include one. For this module the distribution value was relatively low (53%) and the coupling was mainly due to outliers, mostly `parsenodes.h`. In the following year the logical coupling with all the other modules increased, for both strength and distribution, implying that the quality of the design of the backend module decreased as well.

In 2004 we observe that: (i) the dependency with include stayed stable at high values of strength and distribution, (ii) the logical coupling with interfaces and pl decreased and (iii)`tabcomplete.c` and `pl_exec.c` which were outliers up to this moment were not outliers any more. We deduce that a refactoring phase was previously (2003) applied for these two modules (interfaces and pl). This is one of the reasons of the high logical coupling values in the previous year.

In the last year the dependencies with all the other modules decreased (for both strength and distribution), especially with include, bin and port. We deduce that in 2004 the backend module was refactored, since the logical coupling decreased for all the other modules.

Suggestions for the Backend Module. The files `parsenodes.h` and `nodes.h` of the include module should be further analyzed and, in case, moved to the backend module. The first was an outlier from 2002 to 2005 while the second in 2002 and 2005. They were coupled with backend for a long time and they were still coupled in the last year. The file `path.c` of the port module and `executor.h` of the include module should also be analyzed. They were only recently coupled with backend, implying that the dependencies are due to recent changes. We suggest to analyze them because an early refactoring is less expensive.

5. CONCLUSION

The Evolution Radar allows us to study the evolution of the dependencies among system modules and to detect the outliers, the best starting points for the refactoring process. It is also helpful to understand if the dependencies of the outliers are due to recent changes or they were coupled with the target modules for many years. We have shown the results of the application of our approach on the biggest module of PostgreSQL. We have found candidates for reengineering and refactoring phases in the evolution of the modules. For the other two biggest modules we have found similar results but we have not presented them for lack of space.

6. REFERENCES

[1] M. D'Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationships. In *Proceedings of CSMR 2006 (10th European Conference on Software Maintenance and Reengineering)*, pages xxx–xxx. IEEE CS Press, Mar. 2006.

[2] M. D'Ambros, M. Lanza, and M. Lungu. The evolution radar: Visualizing integrated logical coupling information. In *Proceedings of MSR 2006 (International Workshop on Mining Software Repositories)*, pages xxx–xxx, May 2006.

[3] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings International Conference on Software Maintenance (ICSM 2003)*, pages 23–32, Los Alamitos CA, Sept. 2003. IEEE Computer Society Press.