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 responsible 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.

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:

\[
LC(M, f, t_1, t_2) = \max_{f_i \in M} \{sc(f_i, f, t_1, t_2)\}
\]

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 \((t_1, t_2)\).

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.
and coupling with interfaces and `pl` decreased and (iii) stable at high values of strength and distribution, (ii) the logical coupling with the design of the backend module decreased as well.

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

