# Improving Software Quality of the ALICE Data-Acquisition System through Program Analysis

**Jianlin Zhu, Jin Huang, Sylvain Chapeland, Daicui Zhou, Guoping Zhang**

**CHEP2012, Kimmel Center, New York University, May 21-25, 2012**

# Outline

- 1. Introduction
  - 1.1 ALICE Data Acquisition System
  - 1.2 Aspect Mining
- 2. Background
  - 2.1 Concern
  - 2.2 Crosscutting Concern
- 3. Aspect Mining Approaches
  - 3.1 Fan-In
  - 3.2 PageRank
  - 3.3 HITS
  - 3.4 SimCorr
- 4. Experiments
  - 4.1 Program Analysis Framework
  - 4.2 Fan-In
  - 4.3 The Similarity between elements
  - 4.4 Corsscutting Concerns
    - 4.4.1 Special Functions
    - 4.4.2 Identifying Crosscutting Concern
- 5. Future Work
- 6. Conclusion
- 7. Suggestions

# 1. Introduction

- *1.1. ALICE Data Acquisition System*

- The Data-Acquisition System designed by ALICE, which is the experiment dedicated to the study of strongly interacting matter and the quark-gluon plasma at the CERN LHC, is to handle the data flow from the sub-detector electronics to the archiving on different kinds of storage.
- The software framework of the ALICE data acquisition system is called DATE (ALICE Data Acquisition and Test Environment) and consists of a set of software packages which could be grouped into main logic packages and utility packages.

# 1.1. ALICE Data Acquisition System

- Modules in DAQ are banksManager, bufferManager, cole, commonDefs, dateStream, db, editDb, edm, eventBuilder, fec, hltAgent, infoLogger, logbook, monitoring, mrorc, mStreamRecorder, physmem, readList, readout, recordingLib, ReleaseNotes, rorc, runControl, simpleFifo, tdsm, trigger.
- In the Figure 1, the modules in DAQ could be separated into the three layers:
- (1) OS layer. This layer include DDL driver, Physmem driver.
- (2) Library space layer. This layer provides many basic interfaces to the hardware drivers, special structures designed for data taking and recording, libraries for logging, monitoring, memory managing, etc.
- (3) Application space layer. This layer consists of all kinds of applications which are applied in the different situation.

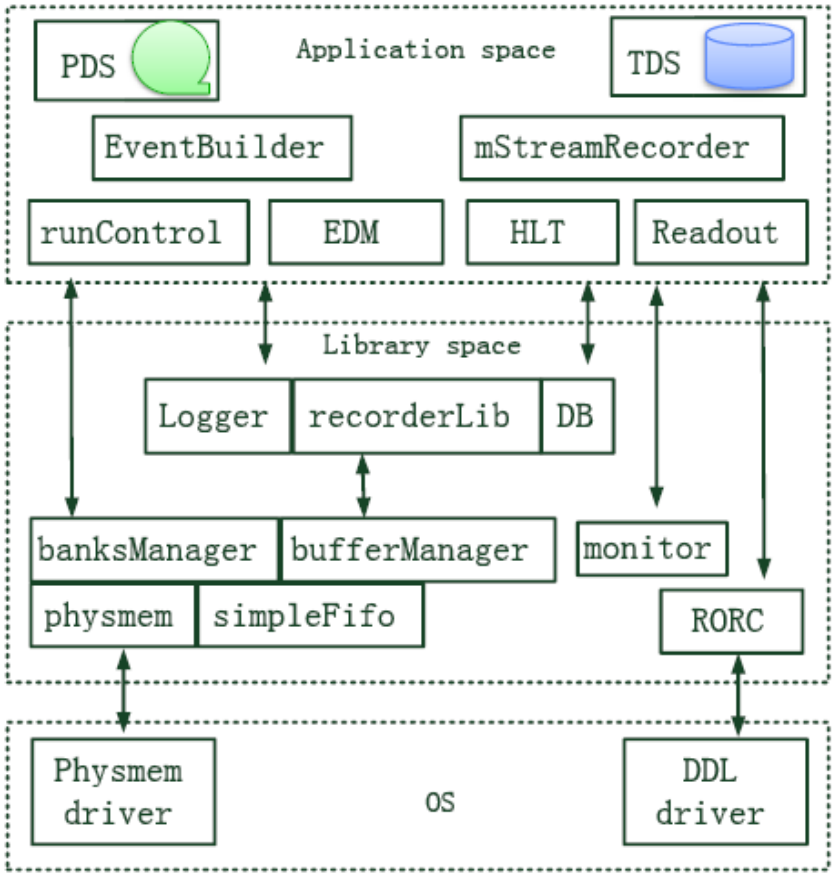


**Figure 1.** The layer of Core Modules in DAQ

# 1.2 Aspect Mining

- To improve the modularization of software system, the Aspect-Oriented Programming(AOP) has been introduced since 1990s. Software modification and maintenance benefit from the separation of crosscutting concerns(CCs).
- The process of refactoring legacy software system is divided into 2 steps: aspect mining(identifying crosscutting concerns in legacy software), and aspect refactoring(refactoring them into aspect).

# 1.2 Aspect Mining

- We propose a program analysis framework which includes many aspect mining approaches and could be used to analyze the DATE. For different research purposes, the suitable algorithms could be applied in DATE and the results could help people more understand the source codes even it has been developed for many years. From the program analysis framework, We could obtain statistical information about each program elements, find the similar elements related with each element, query graph of program and get the final ranking for aspect mining. Our framework is effective regardless of various programming languages of different type.I n this paper, we evaluation it on C projects like DATE.

# 2. Background

- 2.1 Concern

➢In the Software Management area, modularization is to manage the source codes well in order to efficiently query and edit. Managing codes is modularity.
➢In computer science, a concern is of similar function or the set of functions to accomplish one purpose.

# 2.1 Concern

The problems exist in source codes are described as follows:

- Code Scattering-”Scattering” is that similar code is distributed throughout many program modules. Software maintenance may require searching,querying and editing all affected code.
- Code Tangling- ”Tangling” is when two or more concerns are implemented in the same body of code or component, making it more difficult to understand. Changes to one implementation may cause unintended changes to other tangled concerns.

# 2.2 Crosscutting Concern

- The source codes includes two type of elements: core elements and crosscutting elements. The Crosscutting Concerns(CC) is not the Core modules, it is just the concern that provides aiding functions to core modules.
- CC are the features of a software system that are hard to isolate, and whose implementation spread across many different modules. One important feature of the CC is that it has a high degree of scattering.

# 2.2 Crosscutting Concern

There features about CC are summarized as follows:

- Fan-In: We observed that the Fan-In value of CC is high, so we could know that if the Fan-In value of an element is high, it has a high probability to be CC.
- Scattering : The implementation of CC is spread across many different modules.
- Popularity: the node of high Fan-In often has a high Popularity value, so we could know that if the Popularity value of an element is high, it has a high probability to be CC.

# 2.2 Crosscutting Concern

For measuring the characteristic of CC, many aspect mining approaches we could use to get the value for looking deep inside into the system. Such as , the following four values are very common in the recent research work.

- Fan-In value: do the counting of in-degree for each element.
- Scattering degree: couple graph which is extracted from the source codes based on the call relationship.
- Popularity value: the ranking value got from a couple graph in the direction of In-Degree.
- Significance value: the ranking value got from a coupe graph in the direction of Out-Degree.

# 3. Aspect Mining Approaches

- The quality of aspect mining play an important role in identifying code related to a CC. Several aspect mining approaches has been developed to help programmers to identify CCs.
- 4 algorithms are introduced, Fan-In, PageRank, HITS, SimCorr.  SimCorr is our new algorithm to calculate the similarity between elements.

# 3.1 Fan-In

- We define Fan-In of a method m as the number of distinct method bodies that can invoke m.
- We observed that the Fan-In value of CC is high, so we could know that if the Fan-In value of an element is high, it has a high probability to be CC.

# 3.1 Fan-In

- Although Fan-In analysis can find CC candidates, it is not accurate to measure scattering only with reference frequency. Let us consider the example in Figure (3). Element *A is referenced by many* elements in one module, while *B is referenced by less elements from 2 modules. Obviously B is more* likely to be a crosscutting concern than *A*

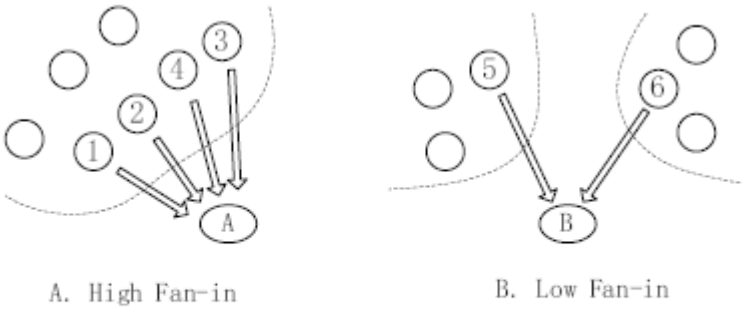


**Figure 3.** Fan-in Limitation

# 3.2 PageRank

- we review google's PageRank algorithm and interpret its functionality for decoupling while surfing the dependency graph of program. A program element obtains high probability of being a CC candidate if it is referenced by a large number of elements.
- In Equation (1), The factor d is the damping factor which 0 < d < 1,and (1−d)is the probability of jumping from each vertex to vertex u.

$$p_u(t+1) = \sum_{v \in pa(u)} \left(\frac{1-d}{N} + d\frac{1}{outdegree(v)}\right) \cdot p_v(t) \qquad (1)$$

# 3.2 PageRank

- PageRank are performed on the coupling graph extracted from the program sources. PageRank
- generates ranks reflecting the degrees of popularity and significance for each of the program elements on the coupling graphs. The node of high Fan-In often has a high popularity value, so we could know that if the popularity value of an element is high, it has a high probability to be CC.
- The advantage of PageRank is its transitivity feature, but it could not solve ”high scattering, low Fan-In” problem.

# 3.3 HITS

- we review HITS algorithm and interpret its functionality for decoupling dependency graphs for programs.
- The HITS algorithm is another link-based rank algorithm for co-citation and web hyperlinks.
- In this algorithm each vertex has two state variables: authority variable which means the its possibility of be a information source page, and hub variable which means the its possibility of be a page linking to information source.
- According to the two state variables of vertices, we consider the vertices of high hub-ranking the good linking vertices and the vertices of high authority-ranking the good
- information source vertices.

# 3.3 HITS

- In Equation (3), where $a_q$ is the authority variable of vertex q, $h_q$ is the hub variable of vertex q, pa(q) is the vertices set in which the vertices link to vertex q and ch(q) is the vertices set in which the vertices is linked by vertex q.
- For a specific program element, HITS can get the authority value which is probability of being a function implementation and the hub value which is the probability of being a core logic. HITS considers the integration and implementation property of a program element,

$$\begin{cases} a_q(t+1) = \sum_{p \in pa(q)} h_p(t) \\ h_q(t+1) = \sum_{p \in ch(q)} a_p(t+1) \end{cases} \quad (3)$$

# 3.4 SimCorr

- we propose our novel approach to measure the similarity between elements even when they are not connected. The SimCorr is very useful when we want to how many repeated functions existed in the source codes or whether the connection inside one e module is highly interrelated.

# 3.4 SimCorr

- Two elements are determined to be similar if they reference many common or similar elements, or they are referenced by many common or similar elements.
- In program dependency graph such as call graph, two methods are considered to be similar if they reference a lot of common methods or fields as Figure (4-A) shows. However, we can not say they are similar if they are called by the common methods, since methods from different concerns can be called in the same method shown in Figure (4-B). Furthermore, it is common that similar methods are not called by the same method since they maybe imple

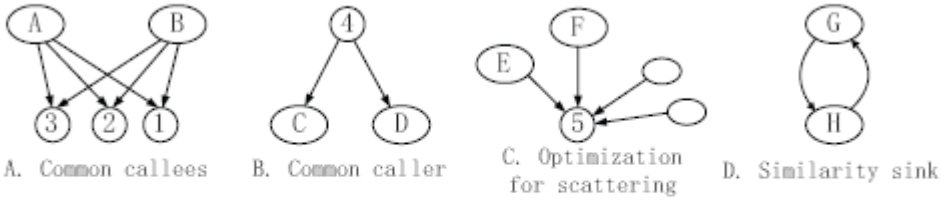


**Figure 4.** SimCorr Correlation

# 3.4 SimCorr

- We use the iterative formula (5) to compute the similarity scores between the program elements.
- In the Equation (5), s(a, b) is the similarity score between elements a and b. O(a) is the element set of a's successors, |O(a)| is the out-degree of a, $O_i(a)$ is the a's ith successor. $|I_i(a)|$ is the in-degree of a's ith successor. t is the adjustment parameter for optimization purpose.

$$s(a,b) = \frac{c}{|O(a)| \cdot |O(b)|} \sum_{i=1}^{|O(a)|} \sum_{j=1}^{|O(b)|} \frac{s(O_i(a), O_j(b))}{(|I_i(a)| \cdot |I_j(b)|)^t} \quad (5)$$

# 3.4 SimCorr

- During the experiments, we have an observation that similarity score between elements will be reinforced if they reference each other, which is called *Similarity Sink, for example in Figure (4-D). Algorithm 1 is used to solve this problem.*

**Algorithm 1:** Similarity Sink (SS)

**algorithm** SS($G$, $a$, $b$, $s(a,b)$)
$G$ is directed graph representing software programs.
$a$ and $b$ are elements in graph $G$.
$O_i(a)$ is the $i$th successor of $a$.
$s(a,b)$: similarity score between $a$ and $b$.
**begin**
  **foreach** *node pair* $(O_i(a), O_j(b))$ **do**
    **if** $\neg\,(\,O_i(a) := b \wedge O_j(b) := a\,)$ **then**
      Add $s(O_i(a), O_j(b))$ to $s(a,b)$;
    **end**
  **end**
**end**

# 4. Experiments

- 4.1 Program Analysis Framework

➢ We implement PAF (Program Analysis Framework) to analyze the software architecture and software modularity.
➢The basic idea about PAF is recording the call relationships information among the important elements (i.e., functions, global variables, complex structures) firstly and then using the different analysis
algorithms to find the CCs which could destroy the modularity of the software from this recording information.

# 4.1 Program Analysis Framework

- The PAF is based on the API of Eclipse C/C++ Development Tooling(CDT)
- The version information of the main softwares and platform is listed as follows:

- Version of Java is above 1.6.
- Version of Eclipse is 3.6.2.
- The source code of DAQ is date-7.6.2.src.tar.gz. Version of DIM is 19.17. Version of SMI is 42.

# 4.2 Fan-In

- We calculate the fan-in value of all the program elements we extracted from DAQ. We give two examples to see top frequency called elements.
- In Table (1), top 10 highest called functions are shown. We could
- find that many elements with high fan-in provide logging and monitoring functions. Specially, many elements from rorc module are also used frequently.

**Table 1.** The Top 10 Functions with high Fan-In

| Function | Fan-in |
|---|---|
| /DAQ/infoLogger/libInfo.c/infoLog() | 470 |
| /DAQ/infoLogger/libInfo.c/infoLog_f() | 84 |
| /DAQ/logbook/DAQlogbook.c/errlog() | 65 |
| /DAQ/infoLogger/simplelog.c/slog() | 56 |
| /DAQ/logbook/DAQlogbook.c/logbook_mysql_query() | 54 |
| /DAQ/rorc/rorc_lib.c/rorcClose() | 48 |
| /DAQ/monitoring/clientInterface.c/monitorDecodeError() | 48 |
| /DAQ/recordingLib/recordingLib.c/checkLogLevel() | 42 |
| /DAQ/recordingLib/recordingLib.c/dumpSet() | 31 |
| /DAQ/rorc/rorc_ddl.c/ddlReadStatus() | 30 |

# 4.2 Fan-In

- From the Table (2), rcShm is the global variable, many elements are the complex variables (e.g., eventHeaderStruct, rorc_pci _dev_t, equipmentHeaderStruct, eventDescriptorStruct, rorcHandle_t).

**Table 2.** The Top 10 Variables with high Fan-In

| Function | Fan-in |
|---|---|
| /DAQ/commonDefs/event.h/eventHeaderStruct | 147 |
| /DAQ/rorc/rorc_lib.h/rorc_pci_dev_t | 96 |
| /DAQ/banksManager/banksManager.c/rcShm | 49 |
| /DAQ/rorc/rorc_ddl.h/stword_t | 47 |
| /DAQ/commonDefs/event.h/equipmentHeaderStruct | 39 |
| /DAQ/db/dateDb.h/dbLdcPatternType | 35 |
| /DAQ/monitoring/queues.h/queuePtr | 33 |
| /DAQ/eventBuilder/eventBuilder.h/eventDescriptorStruct | 31 |
| /DAQ/monitoring/monitorParams.c/monitoringLogLevel | 29 |
| /DAQ/rorc/rorc_lib.h/rorcHandle_t | 25 |

# 4.3 The Similarity between elements

- we give experiment to illustrate the effectiveness of our SimCorr approach to group similar elements.
- In Table (3), the similar elements to function – infoLog, the top 10 similar elements to infoLog are all from the file libInfo.c.

**Table 3.** The Similar Elements for /DAQ/infoLogger/libInfo.c/infoLog

| Rank | Function Name |
|---|---|
| 1 | /DAQ/infoLogger/libInfo.c/_infoLogTo(constchar,constchar,constchar,constchar,) |
| 2 | /DAQ/infoLogger/libInfo.c/infoLog_f(constchar,constchar,constchar,) |
| 3 | /DAQ/infoLogger/libInfo.c/infoLogTo_f(constchar,constchar,constchar,constchar,) |
| 4 | /DAQ/infoLogger/libInfo.c/infoLogS_f(constchar,constchar,) |
| 5 | /DAQ/infoLogger/libInfo.c/infoLogger_msg_xt(constchar,int,int,constchar,constchar,int,constchar,) |
| 6 | /DAQ/infoLogger/libInfo.c/infoPrintStat() |
| 7 | /DAQ/infoLogger/libInfo.c/infoSetStream(constchar,) |
| 8 | /DAQ/infoLogger/libInfo.c/infoSetFacility(constchar,) |
| 9 | /DAQ/infoLogger/libInfo.c/infoLog_handle_destroy(infoLogger_handle_t,) |
| 10 | /DAQ/infoLogger/libInfo.c/infoLogger_setParam(infoLogger_param_t,void,) |

# 4.3 The Similarity between elements

- In Table (4), there are top 10 similar elements to function – rorcClose. In fact, there are only around 10 elements which have similar value to rorcClose, the similarity value with other thousand elements is zero. There are 7 functions from the same file like rorcClose, 2 functions from other 2 modules.

**Table 4.** The Similar Elements for /DAQ/rorc/rorc_lib.c/rorcClose(rorcDescriptor_t,)

| Rank | Function Name |
|---|---|
| 1 | /DAQ/rorc/rorc_lib.c/rorcCheckOpen(int,unsignedint,) |
| 2 | /DAQ/rorc/rorc_lib.c/rorcOpenForMonitor(rorcDescriptor_t,int,int,) |
| 3 | /DAQ/rorc/rorc_lib.c/rorcMapChannel(rorcDescriptor_t,int,int,) |
| 4 | /DAQ/rorc/rorc_lib.c/rorcOpen(rorcDescriptor_t,int,) |
| 5 | /DAQ/rorc/rorc_lib.c/rorcMap(rorcDescriptor_t,int,) |
| 6 | /DAQ/rorc/rorc_lib.c/rorcOpenChannel(rorcDescriptor_t,int,int,) |
| 7 | /DAQ/readList/equipmentList.c/dumpRorcStatus(int,rorcDescriptor_t,) |
| 8 | /DAQ/rorc/rorc_lib.c/rorcQuickOpen(rorcDescriptor_t,int,int,) |
| 9 | /DAQ/rorc/rorc_receive.c/next_push(rorcDescriptor_t,int,int,) |
| 10 | /DAQ/trigger/ctpServer.c/rorc_readout_init(rorc_readout,) |

# 4.3 The Similarity between elements

- In Table (5), there are only 6 similar elements to function - /DAQ/logbook/DAQlogbook.c/errlog(constchar,). We calculate the similar value for every element in the DAQ, and this is very useful when the developers want to look deep into the source codes and hope the system could provide more valuable information about the program elements they care about.

**Table 5.** The Similar Elements for /DAQ/logbook/DAQlogbook.c/errlog(constchar,)

| Rank | Function Name |
|---|---|
| 1 | /logbook/DAQlogbook.c/DAQlogbook_verbose(int,) |
| 2 | /logbook/DAQlogbook.c/debuglog(constchar,) |
| 3 | /logbook/DAQlogbook.c/infolog(constchar,) |
| 4 | /logbook/DAQlogbook.c/logbook_mysql_query(constchar,) |
| 5 | /logbook/DAQlogbook.c/logbook_mysql_query_retry(int,constchar,) |
| 6 | /logbook/DAQlogbook.c/DAQlogbook_update_DAQ_active_components(unsignedint,constchar,void,int,) |

# 4.4 Crosscutting Concerns

- 4.4.1 Special Functions. There are many methods with the same name and function which distribute in many modules, such as usage, C message, doFatalExit, dumpEventId, reportEnd, UPPER, dumpEventType, initVars, signal handler, decodeEventId, etc. We give 10 examples in Table (6) with the information of Function name, Times which means how many times it has been used, Distribution

**Table 6.** The Special Functions

| Function | Times | Distribution |
|---|---|---|
| usage | 70 | recordingLib,monitoring,simpleFifo,rorc,physmem,runControl,infoLogger,etc |
| C_message | 41 | runControl |
| doFatalExit | 38 | eventBuilder,edm,readout |
| dumpEventId | 26 | edm,eventBuilder,runControl |
| reportEnd | 24 | monitoring |
| UPPER | 20 | runControl |
| lock_queue/unlock_queue | 18 | runControl |
| requestEor | 18 | eventBuilder,mStreamRecorder |
| dumpEventType | 12 | readout,monitoring,eventBuilder |
| initVars | 12 | monitoring,simpleFifo,dateStream,recoringLib |

# 4.4.1 Special Functions

- There are main three situations for these kind of functions which could be summarized as follows:
- (1) Some functions which are defined in many files have the same context. These kind of functions exist in many modules. In runControl module, runControl.c , rcServer.c and runControlHI.c share many the same context functions. monitorPartitions.c, monitorGdcs.c, and monitorByDetector.c in monitoring module are the same.
- (2) Different functions with the same name appear in different modules and files, such as dumpBuffer in readList and monitoring modules.
- (3) the same function with different names, such as print usage and usage.

# 4.4.2 Identifying Crosscutting Concern

- In order to define crosscutting features in DAQ, we have defined various levels at which a concern crosscuts through the system functionality. These levels are defined in Table (7). The purpose of such a classification is to know how scattered the concerns are across the code base.

**Table 7.** Crosscutting Levels

| Crosscutting | Concern code scattering |
|---|---|
| Intra-Function | > 1 function of same file |
| Intra-File | > 1 file of same module |
| Intra-Module | > 1 module of same system |

# 4.4.2 Identifying Crosscutting Concern

- We summarize below the DAQ aspects with their functionality and crosscutting natures. Figure 7 shows the crosscutting of the DAQ aspects. Detail information is described in our paper. We give 1 example.
- *RecordingLibrary(R): This concern is the low-level recording library which is used by processes* who need full control over their output channels. It provides the functions to handle a set of channels (for multiple part parallel output streams) and allows both synchronous and asynchronous output. In the DAQ system, the code crosscuts through 2 modules( eventBuilder, recordingLib ) for controlling over the output channels. In eventBuilder module, the ldcHandler.c and recordingHandler.c use the RecordingLibrary crosscutting concern; In recordingLib module, datenetperfldc.c, validator.c, dateRec.c and recordingLib.c use the RecordingLibrary crosscutting concern.

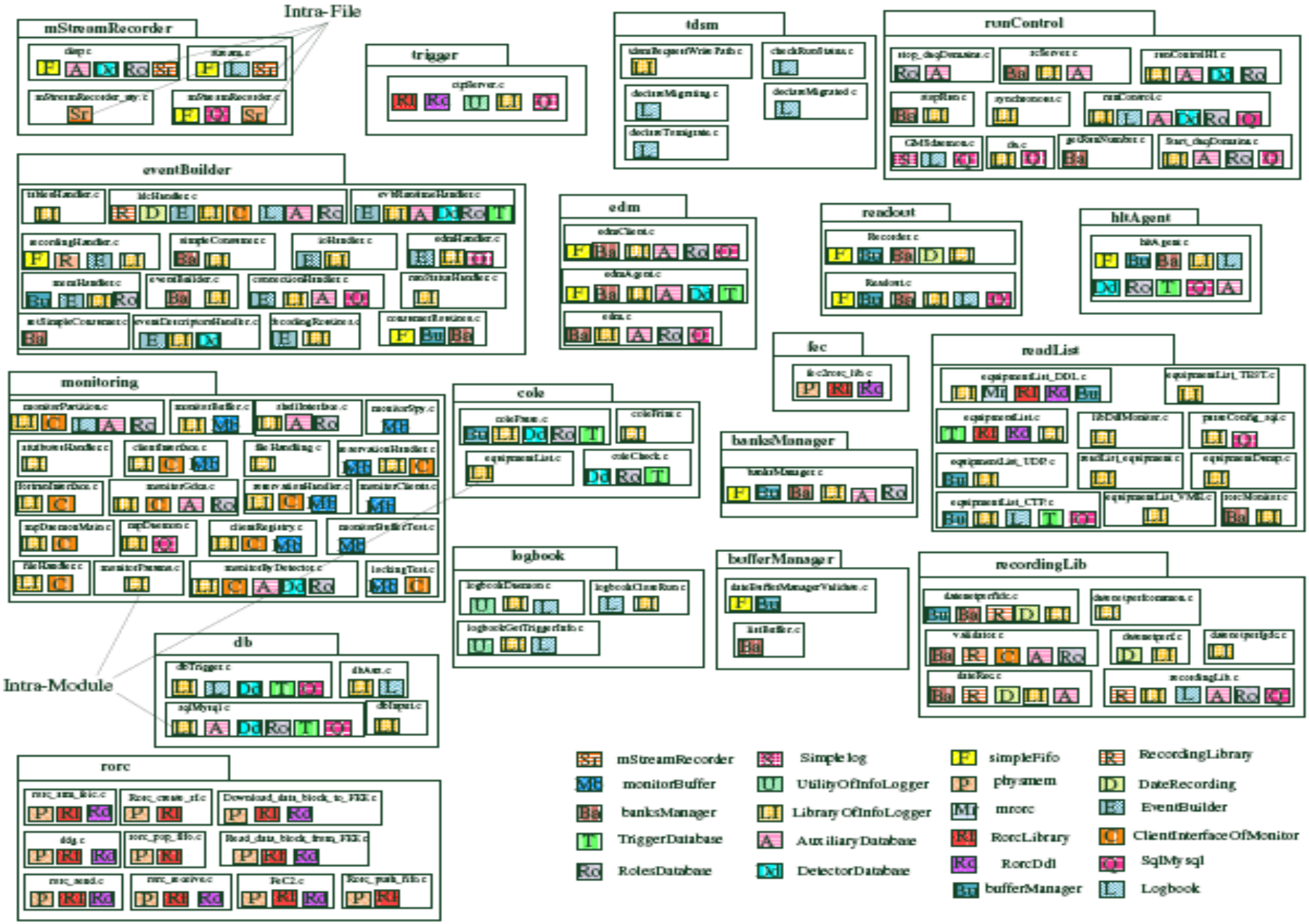


**Figure 5.** Simple Crosscutting Concerns in DAQ

# 4.4.2 Identifying Crosscutting Concern

- The aspect crosscutting level of crosscutting concerns mentioned above is summarized in Table (8).

**Table 8.** Aspect Crosscutting Levels in DAQ

| Aspect | Intra-Function | Intra-File | Intra-Module |
|---|---|---|---|
| Sr | √ | √ | |
| Mb | √ | √ | |
| R | √ | √ | √ |
| D | √ | √ | √ |
| E | √ | √ | √ |
| C | √ | √ | √ |
| Q | √ | √ | √ |
| L | √ | √ | √ |
| F | √ | √ | √ |
| P | √ | √ | √ |
| Mr | √ | √ | √ |
| Rl | √ | √ | √ |
| Rd | √ | √ | √ |
| Bu | √ | √ | √ |
| S | √ | √ | √ |
| U | √ | √ | √ |
| Ll | √ | √ | √ |
| A | √ | √ | √ |
| Dd | √ | √ | √ |
| Ba | √ | √ | √ |
| T | √ | √ | √ |
| Ro | √ | √ | √ |

# 5. Future Work

- Our goal in the future is to do the research work on the complex network and try to automatically analyze the software systems with the graph clustering approaches.
- A longer term plan is an aspect mining tool that combines several analysis techniques to accomplish a higher degree of completeness and precision.

# 6. Conclusion

- The Program Analysis Framework is built for analyzing the C projects
- This is the first time for us to analyze the DAQ system with many aspect mining approaches.
- SimCorr is adopted to measure the similarity between the program elements.

# 7. Suggestions

- We hope this work could get more advices and suggestions, so we could try to mine more valuable information from the software systems and more optimization steps could be applied into the source codes.

Thank you so much , the end of presentation!