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

Jianlin Zhu [1], **Jin Huang** [2], **Sylvain Chapeland** [3], **Daicui Zhou** [1], **Guoping Zhang** [1]

[1]College of Physical Science and Technology, Central China Normal University, 430079, China
[2]College of Computer Science and Technology, Huazhong University of Science and Technology, 430074, China
[3]CERN, CH 1211 Geneva 23, Switzerland

E-mail: Jianlin.Zhu@cern.ch

Abstract.

The ALICE Data-Acquisition System is to handle the data flow from the sub-detector electronics to the archiving on different kinds of storage. In this paper, it is analyzed with the techniques in Aspect Mining. We implement Program Analysis Framework(PRF) to analyze the software architecture and software modularity. The basic idea about PAF is to record the call relationships information among the important elements firstly and then use the different analysis algorithms to find the crosscutting concerns which could destroy the modularity of the software from this recording information. Finally we evaluate our framework through analyzing the software system of DATE. The analysis result proves the effectiveness and efficiency of our framework. With the help of PAF, the software quality of DATE could be improved from many aspects. Many optimization steps and suggestion could be applied to DATE. In addition, the PAF for DATE could also analysis the projects written in C language.

1. Introduction

1.1. ALICE Data Acquisition System

The Large Hadron Collider(LHC) comes across a lot of challenges in many areas, and one of the challenges is the data handling technology. ALICE(A Large Ion Collider Experiment) experiment at the LHC is designed to study the physics of strongly interacting matter and the quark-gluon plasma in nucleus-nucleus collisions and has a data acquisition system to deal with data handling challenge. The Data-Acquisition System designed by ALICE ([1]) , 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.

The main functions of DAQ include:

- Core function: Core function of DAQ is to realize the dataflow from the detector up to the data storage. It includes the dataflow from the detector electronics up to the DAQ computing fabric and to the High-Level Trigger(HLT) farm, the transfer of information from the HLT to the DAQ fabric, and the data archiving in the CERN computing centre.
- Data quality monitoring.

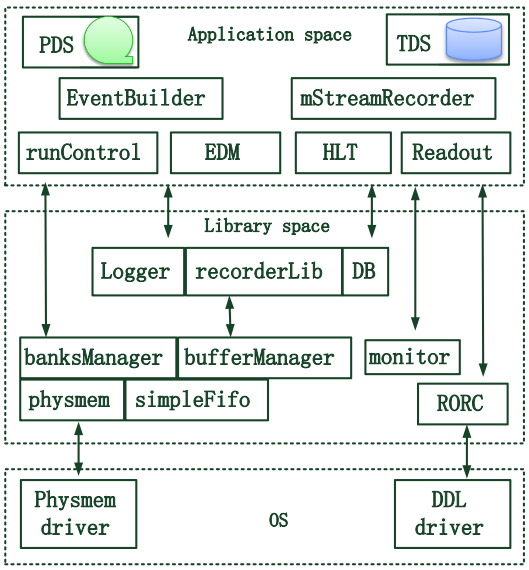

Figure 1. The layer of Core Modules in DAQ

- System performance monitoring.
- Overall control of the 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:

(i) OS layer. This layer include DDL driver, Physmem driver.

(ii) 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.

(iii) Application space layer. This layer consists of all kinds of applications which are applied in the different situation.

1.2. Aspect Mining

To improve the modularization of software system, the Aspect-Oriented Programming(AOP) [2] 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).

Several measurements ([3, 4, 5]) for crosscutting concern have been proposed to describe the cohesion and couple of CCs in software system. According to these measurements, many aspect mining approaches ([6, 7, 8, 9]) have been used to identify crosscutting concerns. Most of these approaches try to find seeds of crosscutting concern, but not the complete ones that consist of several elements of which non-seeds exist.

software metrics are used to measure the quality of software system. Aspect mining needs proper metrics to accurately locate the code related to a crosscutting concern. Marc Eaddy [10] design a

suite of metrics for identify CCs and prove its ability to find CC candidates. However, these metrics are computed with *line of code*(LOC) related to concerns, which is difficult to identify manually or automatically. For example in Listing (1), the simple expression in line 3 indicated several references to different components and is hard to decide which concern it belongs to. The LOC is not capable of representing the semantic references in programs.

```
1 public void draw(Graphics g) {
2         ......
3         Color frame = g.getframe().getColor();
4         ......
5 }
```

Listing 1. A sample

Otherwise, a simple assumption adopted by these metrics is that once an element of program is decided to related to some components, then the correlations between the element and components are the same(usually distance = 1). For example, let us consider a simple example in Figure (2). Intuitionally, we get the conclusion that the correlation between *A* and *B* is lower than that of *C* and *B*, since the *C* only reference *B* while *A* references several elements besides *B*. Usually the correlation between elements is decided by their context which can be local and even global.

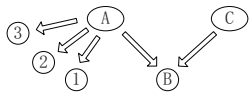

Figure 2. Correlation Between Elements

The accuracy of measurement is dependent on the modularity quality of software system. Recent work of aspect mining believe that basic program element such as method(function) and class are well modularized. It quite possible that there exist bad smells in program which affect the precision of measurement. Furthermore, scatter information has side effect on the detecting modularity quality of software.

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.In this paper, we evaluation it on C projects like DATE.

This paper makes the following main contributions:

(i) The Program Analysis Framework is built for the program analysis, specially for the C projects.

(ii) Many aspect mining approaches are included in the framework for providing information about the software.

(iii) SimCorr provides a heuristic distance to improve the measurements between for software system.

(iv) link analysis is adopted to get scattering measurement for CCs, which is adopted for aspect mining.

(v) Many experiments show that our framework is useful and could be applicable to the C software system.

This paper is organized as follow. In the next section we will introduce some back ground knowledge. In section 3 we introduce the aspect mining approaches related to our experiments and proposed our approach to measure similarity between elements. In section 4, we show the performance of many

aspect mining approaches. The future work is discussed in section 5, conclusion in section 6 and Acknowledgments in section 7.

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. Module is the method to implement concern. In computer science, a concern is of similar function or the set of functions to accomplish one purpose.

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.

The reasons why we need to find the CCs are to improve modularity, to maintain software codes, and to understand the source codes easily. For the ALICE DAQ system, it has the constant demand for system change and upgrades comes the need to simplify and ensure accuracy. Finding the Crosscutting Concern(CC) automatically can benefit the maintainability of the DAQ.

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.

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.

With the different measurement values and analysis approaches, the Crosscutting Concerns can be discovered in the absence of domain knowledge of the investigated application. The Steps to find CCs could be summarized as follows:

- Step 1: Finding CC seeds. These seeds are key elements of CC and could be used to inspire the finding of total CC. There are many approaches are used to find the CC seeds, such as Fan-In, Random Walk, Link Analysis,etc.

- Step 2: Finding CC core. CC core is the set of CC seeds which are belonging to the same CC. Graph Clustering, Complex Network could be the approaches to finding CC core.
- Step 3: Composite CC. The Graph Clustering, SSL(Semi-Supervisor-Learning) are the approaches which are used to find Composite CC.

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. Fain-in analysis can identify CC candidates that are unrelated seeds, the code of a complete CC needs to be confirmed manually. Random walk model uses *popularity* and *significant* to characterize vertices in a network and computes a ranking for aspect mining. It takes the structure information of system coupling graph into account and improves the concept of scattering. However, it is still not able to obtain code related to a CC. Recently several clustering based approaches try to collect related code into the same groups, however these approaches are lexical-based which is not reliable. In this section, we mainly introduce 4 algorithms which are Fan-In, PageRank, HITS, SimCorr.

3.1. Fan-In

We define Fan-In of a method m as the number of distinct method bodies that can invoke m [6]. 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.

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*. Fan-in analysis relies on the assumption that programmers well modularize software system, which is in fact always full of *bad smell* and needs clustering to find correct modules. That is also the reason why it can not find the crosscutting concerns only with a few degree of fan-in. Not only fan-in analysis, most of aspect mining approaches are based on this assumption, which make the mining result inaccurate, even fallacious.

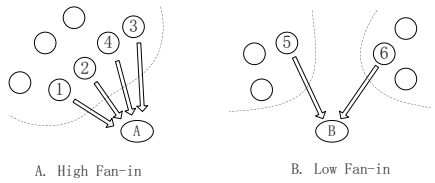

Figure 3. Fan-in Limitation

3.2. PageRank

In this section we review google's PageRank algorithm [11] 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 the original page-rank algorithm, the probability of visiting vertex u is expressed as:

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

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. The matrix form of equation(1) is as equation(2). Especially, the jumping possibility

makes the matrix A the positive stochastic matrix. In this case, the probability vector $p_u(t)$ will converge to the eigenvector of matrix A.

The conventional computation of probabilities with matrix form is expressed as:

$$P(t+1) = A \cdot P(t) \tag{2}$$

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-Inproblem.

3.3. HITS

In this section we review HITS algorithm [12] 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. The original equation of probability computation is defined as:

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

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. If A is the authority vector and H is the hub vector, the equation(3) can be rewritten as :

$$\begin{cases} A(t+1) = W \cdot H(t) \\ H(t+1) = W^T \cdot A(t+1) \end{cases} \tag{4}$$

where W is the adjacent matrix of graph, so the matrix W is not the stochastic matrix and the vectors A and H will not converge to the main eigenvector of matrix W and W'. It has been proved that as t tends to infinity, the direction of vector A will converge to the direction of the main eigenvector of the matrix ww', and the the direction of vector H will converge to the direction of the main eigenvector of the matrix $w'w$. Normalization of vectors A and H are required after each iteration.

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, and further, it takes the relation of the two properties which captures more complex structure information of dependency graph than pagerank algorithm.

3.4. SimCorr

In this section, 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.

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 implement different functions of a concern.

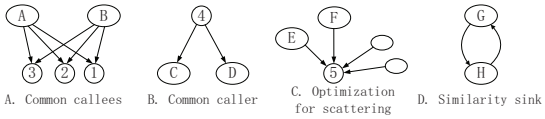

Figure 4. SimCorr Correlation

Inspired by SimRank [13], 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.

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

The program elements calling many common or similar elements will get higher similarity scores. But some elements in the graph exist a large scatter degree, and this scatter feature affects the accuracy of similarity computation results. So we need optimize the computation of similarity to reduce side effect caused by the scatter between elements. For example in Figure (4-C), the ellipses like *E* and *F* stand for the different modules in the software system; the node 5 means a frequency called program element, so node 5 has a high scatter degree. According to the theory of similarity computation, the references from different modules *E* and *F* to crosscutting node *5* significantly increase similarity score of different modules unexpectedly. To eliminate the side effect of crosscutting elements, an optimization is taken through dividing the similarity scores of their successors by their in-degrees which we could see from the Equation (5). Finally, it is quite easy to prove the convergency of the optimized equation which is similar to the proof for SimRank.

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), if the subgraph exists in program dependency graph, their similarity score will finally become extremely high. We adopt the approach in Algorithm (1) to solve this problem. The main idea about this algorithm is that after each similarity iterative computation, it will check whether the graph exists the Similarity Sink, and if SS exists, the similarity score will be dropped.

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.

The PAF is based on the API of Eclipse C/C++ Development Tooling(CDT) because the source codes of DATE framework is written in C language. The CDT project based on the Eclipse platform provides a fully functional C and C++ Integrated Development Environment. We extract the program elements and call relationships of the C projects. The program elements consist of functions, global variables, the variables of complex types (e.g., typedef, union, struct, enum) which don't include the

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 ith 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

macro information. The call relationships could be the function-to-function or the function-to-variable. The algorithms for program analysis is written in Java language, which are Fan-In, PageRank, HITS, SimCorr, LAAM, MSAM, etc. The PAF for DATE could also analysis the projects written in C language.

DATE has 24 modules, 343 files. The raw data about elements and relationships extracted from DAQ include the elements of C library, but we filter these elements and relationships and mainly focus on the functions and elements provided by DAQ. The version information of the main softwares and platform is listed as follows:

(i) Version of Java is above 1.6.

(ii) Version of Eclipse is 3.6.2.

(iii) 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

Top 10 variables with high fan-in value are shown in Table (2). From the table, we could know

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

that rcShm is the global variable, many elements are the complex variables (e.g., eventHeaderStruct, rorc_pci_dev_t, equipmentHeaderStruct, eventDescriptorStruct, rorcHandle_t).

4.3. The Similarity between elements

In this section, we give experiment to illustrate the effectiveness of our SimCorr approach to group similar elements. Proper definition of correlation between nodes is the primary challenge for analyzing the relationships between elements. SimCorr is used to calculate the similarity between elements through their features and behaviors in context. We give three examples to see the top similar elements of three functions.

In Table (3), the similar elements to function - /DAQ/infoLogger/libInfo.c/infoLog(constchar,constchar,constchar,) are listed and the results are ranked according to the similarity value. From the table, we could see that 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,)

In Table (4), there are top 10 similar elements to function - /DAQ/rorc/rorc_lib.c/rorcClose(rorcDescriptor_t,). 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.

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

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,)

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 which means the name of modules have this function.

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

There are main three situations for these kind of functions which could be summarized as follows:

(i) 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.

(ii) Different functions with the same name appear in different modules and files, such as dumpBuffer in readList and monitoring modules.

(iii) the same function with different names, such as print_usage and usage.

The number of these kind of functions should be reduced as less as possible. Many optimization could be done according to the situation.

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. Note, from the crosscutting definitions, one concern can exhibit more than one crosscutting level.

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

We summarize below the DAQ aspects with their functionality and crosscutting natures. Figure (5) shows the crosscutting of the DAQ aspects.

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.

DateRecording(D): This concern is the high-level recording library which provides an abstract access layer to the low-level recording library. The high-level recording library can handle only DATE raw events and uses an approach similar to the one implemented in the low-level recording library. Basically, a set of channels is handled all together and many events can be sent simultaneously and asynchronously to any of the open channels. The library then handles the relations with the guest Operating System to queue and perform in parallel all the outstanding transfers. In the DAQ system, the code crosscuts through 2 modules(recordingLib, readout). In readout module, only Recorder.c adopts the DateRecording crosscutting concern. In recordingLib module, datenetperfldc.c, datenetperf.c and dateRec.c use the DateRecording crosscutting concern.

EventBuilder(E): This module includes all the decoding routines used by the event builder to dump and print its internal data structures, for error, debugging and normal reporting. In the DAQ system, the code crosscuts through 4 modules (eventBuilder, runControl, hltAgent, db).

ClientInterfaceOfMonitor(C): This is the module that contains all the monitor clients' interface calls. In the DAQ system, the code crosscuts through 2 modules(monitoring, recordingLib).

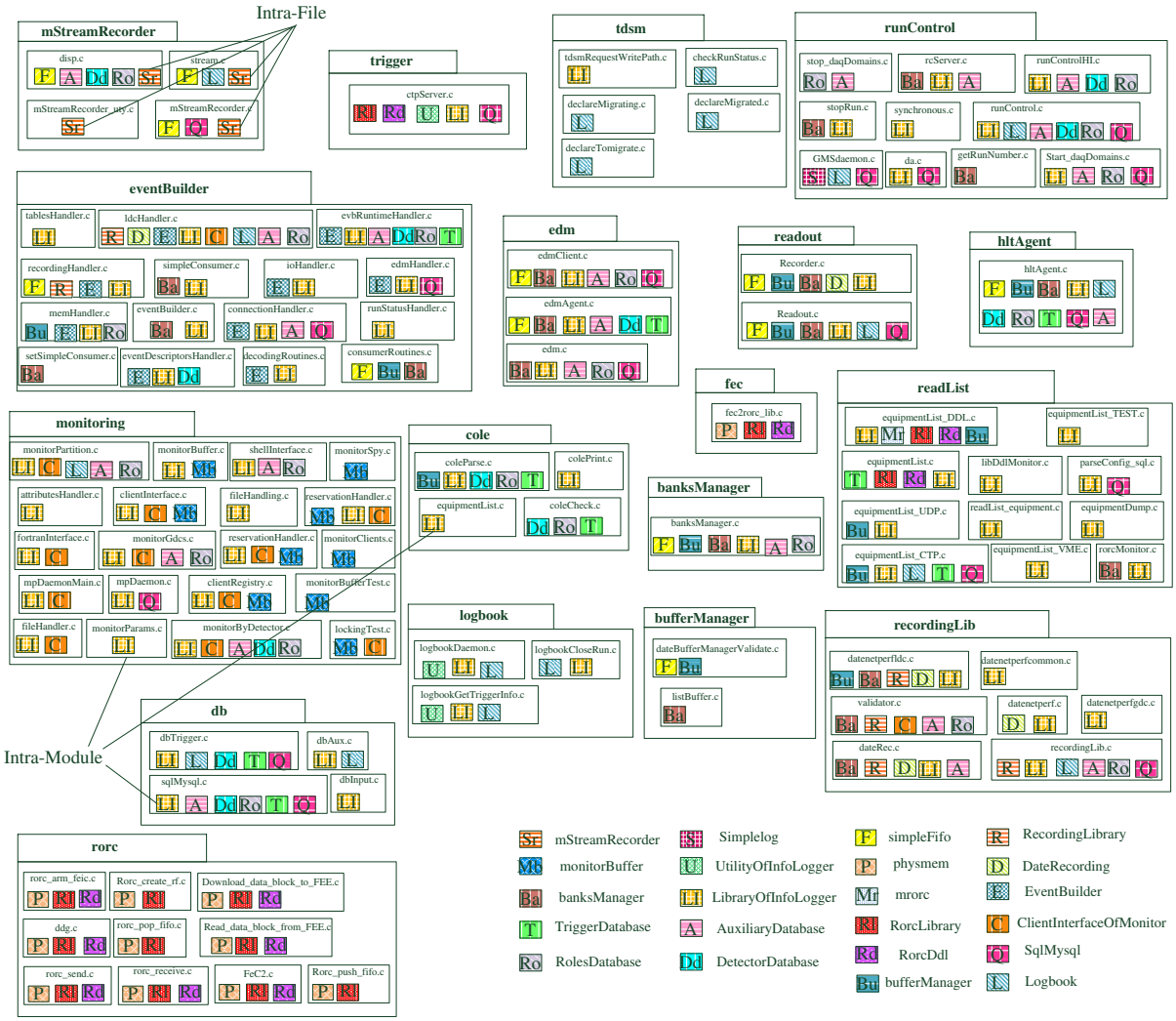

Figure 5. Simple Crosscutting Concerns in DAQ

SqlMysql(Q): This module provides DATE SQL database routines which are implemented for MySQL. In the DAQ system, the code crosscuts through 4 modules(readList, readout, db, trigger).

Logbook(L): This module is designed for DAQ logbook daemon, which could process collecting and publishing SOR/EOR events per detector/partition, collect data from DCS DIM services and publish them in logbook at SOR, etc. In the DAQ system, the code crosscuts through 11 modules (Logbook, readList, db, monitoring, recordingLib, mStreamRecorder, tdsm, readout, runControl, eventBuilder, hltAgent).

simpleFifo(F): The simpleFifo package implements a simpleFifo entity using a shared memory block provided by the caller. This block is partitioned into a control block and a data block. The simpleFifo entity can then be used to exchange blocks of arbitrary size in a first-in first-out fashion. In the DAQ system, the code crosscuts through 8 modules (eventBuilder, simpleFifo, mStreamRecorder, readout, hltAgent, edm, bufferManager, banksManager).

Physmem(P): This concern provides the access to the memory device. The physmem package gives the opportunity to set aside from Linux a region of physical memory and use it as a bank. This is necessary for devices such as the DDL, which write directly into memory through the PCI bus. In the DAQ system, the code crosscuts through 3 modules(fec, Rorc, Physmem).

Mrorc(M): This concern provides library of ALICE mRorc. In the DAQ system, the code crosscuts through 2 modules (readList, mrorc).

RorcLibrary(Rl): This concern provides Library routines for ALICE RORC programs. The RORC (Read-Out Receiver Card) which is the interface between the DDL (Detector Data Link) and the LDC. In the DAQ system, the code crosscuts through 4 modules(readList, fec, rorc, trigger).

RorcDdl(Rd): This concern provides DDL related functions for ALICE RORC test. In the DAQ system, the code crosscuts through 4 modules(readList, trigger, rorc, fec).

BufferManager(Bu): The DATE bufferManager package provides the support for allocation and deallocation of memory coming from a common buffer via a lightweight protocol. This concern provides three sets of entries: one set for the producer process (who allocates blocks), one set for the consumer processes (who deallocate blocks) and one set common between all classes of processes. In the DAQ system, the code crosscuts through 8 modules(hltAgent, eventBuilder, readList, readout, banksManager, recordingLib, bufferManager, cole).

SimpleLog(S): This concern defines functions to log messages. Each message has a log level. Output format is standardized (includes timestamp and message severity). Messages are printed to stdout, or a file. Debugging messages can be discarded. Fatal messages cause exit. In the DAQ system, the code crosscuts through 3 modules (infoLogger, runControl, logbook).

UtilityOfInfoLogger(U): This file include 4 different concerns. In the DAQ system, the code crosscuts through 3 modules(infoLogger, trigger, logbook).

- Memory allocation and string duplication with integrated error handling.
- C implementation of a FIFO class (storing pointers to void).
- C implementation of a line buffer class.
- C implementation of a thread safe FIFO with wait / sync, storing pointers to void.

LibraryOfInfoLogger(LI): This concern provides Client side C-library for logging messages to infoLoggerReader. In the DAQ system, the code crosscuts through 15 modules(infoLogger, hltAgent, banksManager, eventBuilder, recordingLib, cole, edm, runControl, monitoring, tdsm, readList, readout, db, trigger, logbook).

AuxiliaryDatabase(A): This concern provides the access to the memory device. This concern provides auxiliary routines for the DATE database package. In the DAQ system, the code crosscuts through 8 modules(edm, db, runControl, eventBuilder, banksManager, monitoring, recordingLib, mStreamRecorder).

DetectorDatatbase(Dd): This Module to implement DB access to the detectors database. In the DAQ system, the code crosscuts through 8 modules(Db, hltAgent, cole, mStreamRecorder, runControl, edm, eventBuilder, monitoring).

banksManager(Ba): DATE banks manager module is used for loading, sizing, mapping and initialization of the memory banks (and the entities included therein) associate to a running DATE system. All DATE actors running on the same DATE role will map to the same memory bank(s). This module defines the API to map to the banks and to initialize the entities therein. We also define the entities - as available on a running DATE system - in terms of virtual address, offsets and sizes. In the DAQ system, the code crosscuts through 9 modules(recordingLib, banksManager, runControl, eventBuilder, hltAgent, edm, readList, readout, bufferManager).

TriggerDatabase(T): This Module to implement DB access to the trigger database. In the DAQ system, the code crosscuts through 5 modules (hltAgent, db, cole, edm, eventBuilder).

RolesDatabase(Ro): This Module to implement DB access to the DATE roles database. In the DAQ system, the code crosscuts through 10 modules(db, edm, runControl, monitoring, eventBuilder, hltAgent, cole, banksManager, mStreamRecorder, recordingLib).

mStreamRecorder(Sr): The ”Multi-Stream recorder” library provides common functions for disp and stream, the dispatcher internals, the stream internals independent of ROOT/CASTOR. In a standalone

mode just prints to stdout. In the DAQ system, the code crosscuts through 4 files inside mStreamRecorder module(mStreamRecorder.c, disp.c, mStreamRecorder_uty.c, stream.c).

monitorBuffer(Mb): This is Monitor buffer handler module. In the DAQ system, the code crosscuts through 9 files inside monitorBuffer module (monitorSpy.c, reservationHandler.c, clientRegistry.c, monitorBuffer.c, eventHandler.c, clientInterface.c, monitorBufferTest.c, monitorClients.c, lockingTest.c).

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

The basic work in Program Analysis Framework is finished. From the PAF framework, we could obtain the raw data of the C projects, get the fan-in value and similar elements set for each program, rank the program elements with different algorithms. 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. The research work on taking advantage of graph clustering to group program elements related to one concern into a cluster is our current plan. 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

In this paper, we analyzed DAQ system in significant detail, searching for opportunities to use aspect oriented techniques in them. The contributions of this paper are that 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, and the SimCorr is adopted to measure the similarity between the program elements.

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

7. Acknowledgments

The ALICE experiment needs a huge work from the hardware to the software. We would like to thank all members of ALICE cooperation for providing efforts on developing, maintaining, debugging and deploying the whole system.

8. Reference

[1] V. Altini, T. Anticic, F. Carena, W. Carena, S. Chapeland, V. C. Barroso, F. Costa, E. Dnes, R. Divi, U. Fuchs, T. Kiss, I. M. andF. Roukoutakis, K. Schossmaier, C. Sos, P. V. Vyvre, and B. von Haller for the ALICE collaboration, "Commissioning and first experience of the alice data acquisition system," in *Proceeding of the 16th IEEE-NPSS Real Time Conference (RT '09)*, 2009.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of European Conference on Object-Oriented Programming(ECOOP)*, 1997.

[3] O.Ormandjieva, M. Kassab, and C. Constantinides, "Measurement of cohesion and coupling in oo analysis model based on crosscutting concerns," in *Proceedings of the 15th International Workshop on Software Measurement (IWSM2005)*, 2005.

[4] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho, "Do crosscutting concerns cause defects?" *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING(TSE)*, vol. 34, pp. 497–515, 2008.

[5] E. Figueiredo, A. Garcia, M. Maia, G. Ferreira, C. Nunes, and J. Whittle, "On the impact of crosscutting concern projection on code measurement," in *Proceedings of the 10th International Conference on Aspect-Oriented Software Development(AOSD 2011)*, 2011.

[6] M. Marin, A. van Deursen, and L. Moonen, "Identifying aspects using fan-in analysis," in *Proceedings of the 11th Working Conference on Reverse Engineering*. Washington DC, USA: IEEE Computer Society, 2004, pp. 132–141.

[7] C. Zhang and H.-A. Jacobsen, "Efficiently mining crosscutting concerns through random walks," in *Proceedings of the 6th international conference on Aspect-oriented software development*, 2007, pp. 226–238.

[8] D. Zhang, Y. Guo, and X. Chen, "Automated aspect recommendation through clustering-based fan-in analysis," in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering(ASE '08)*, 2008.

[9] J. Huang, Y. Lu, and J. Yang, "Aspect mining using link analysis," in *Proceedings of the Fifth International Conference on Frontier of Computer Science and Technology(FCST 2010)*, 2010.

[10] M. Eaddy, A. Aho, and G. C. Murphy, "Identifying, assigning, and quantifying crosscutting concerns," in *Workshop on Assessment of Contemporary Modularization Techniques (ACOM)*, 2007.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Tech. Rep., 1998.

[12] J. Kleinberg, "Authoritative sources in a hyperlinked environment," in *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[13] G. Jeh and J. Widom, "Simrank: A measure of structural-context similarity," in *Proceeding of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2002)*, 2002.