Statistical Stream Metrics for Software Quality

Meena Sharma
Associate Professor
IET, Devi Ahilya University
Indore, India

Dr. Rajeev G Vishwakarma
Professor
SVITS, Rajiv Gandhi Technical University
Indore, India

ABSTRACT
The Statistical stream metrics developed by us are of unique type (as compared to the existing available metrics) and we propose these metrics as the solution towards software quality. Probably the managers feel they are a bit "techie." We expect that this concise research of the measures has shown that they are practical and pragmatic techniques of assuring quality. The foundation of statistical stream metrics is based upon the principle of FanIn & FanOut or component coupling. Most of the systems consist of components and it is the software performance that these components actually do. The way components are linked or associated together pretty much effect the complexity of a software product. If a component has to do a number of separate tasks it is said to be lacking in "cohesion." Also, systems are highly coupled, if the components within the system communicate data extensively with other components. Systems theory approach talks about that the components which are highly coupled and are less cohesive. These sorts of components with more coupling and less cohesion may be less reliable and difficult to maintain than those components that are loosely coupled and highly cohesive.

General Terms
Software Quality

Keywords
Quality, Metrics, Statistical Stream, Geometric Progression

1. INTRODUCTION
The set of metrics we have developed extends the structural metrics and are based on coupling of the software components. It is structured and placed as it generally falls under the category of metrics with stream of statistical information. At the conceptual level our statistical stream metrics (SSM) are not difficult to understand; it is when we come to apply them that the fun can start. It is good to have practical and realistic approach to achieve results.

The basis of SSM is founded upon the following working premise. Software products are made up of components and the software functionality is produced by the interaction between the components. The way they are fitted together that influence the complexity of a system. If a component has to do several discrete tasks it is said to be deficient in "cohesion." Systems are highly coupled, if the components within the system give and accept data with other components. Systems theory approach talks about that the components that are having more of coupled among each other fall short of cohesion and thus these components have poor reliability. The components with low coupling are more trusted and easy to maintain.

2. STATISTICAL STREAM METRICS FORMULATION
Let us now consider the definition of the terms cohesion, coupling and component. Cohesion can be defined as the intra components complexity that is calls inside the component itself. Coupling is the degree of connection among one component and others in the same system. A component is any element known by decomposing a (software) system into its elemental parts. The systems approach maps to software systems particularly easily as most engineers use currently. At least recognizable with, top-down design techniques that create hierarchical arrangements of system components are popular. Rapid engineering is one of the approaches picking up very fast subject to the fact that we should produce proper documentation, and maintainability. Here again, SSM can be used. The amount of cohesion and coupling among components is analyzed and represented as the statistical stream metrics. The methodology for the construction of the model can reasonably range from the simple to the complex. We propose to begin with basic illustration of SSM to show the simplest concepts, how to gain data using the metrics and how to utilize that data. We will then expand the basic model.

SSM are piloted to the components of a system design. Fig.1 shows an instance of organization of the components. We observe for component A and we can define three measures.

The first measure is "FanIn." This is basically a count of the number of components which can call, or pass control, to component A. The second is "FanOut." This is the count of the number of components which are called by component A. The third measure is derived from the first two by using the following formula. We will call this computation the statistical stream index of module A, abbreviated to SS(A). Let us describe our methodology in detail as follows.

We propose the statistical metrics assessment formula based on geometrical progression. Let us write all the geometrical progression for n terms as,

ar0, ar1, ar2 , ar3 ……. ar,n-2 , arn-1

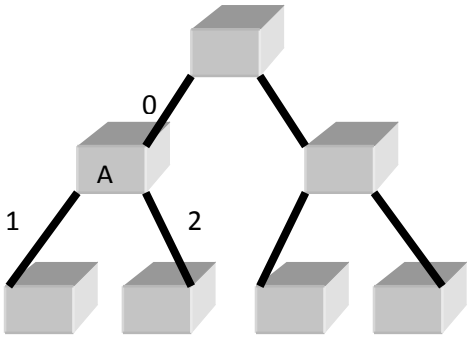

Fig 1: Components & Complexity

We propose the statistical metrics assessment formula based on geometrical progression. Let us write all the geometrical progression for *n* terms as,

$$ar^0, ar^1, ar^2, ar^3 \ldots\ldots\ldots ar^{n-2}, ar^{n-1} \quad (1)$$

where $r \neq 0$, is the common ratio and '*a'* is a scale factor, equal to the sequence's start value.

The n-th term of a geometric progression with initial value 'a' and common ratio 'r' is given by

$$a_n = ar^{n-1}$$

Our statistical stream approach is based on the mathematical fact of Geometrical Progression as described above; that is, the general form can be represented as the geometric progression.

In our case we have taken:

$$a = (FanIn(A) * FanOut(A))^{n-1} \quad (2)$$

where n is a fixed constant value for a particular series for all the terms

$$r = -1/((FanIn(A) * FanOut(A)) \quad (3)$$

n= the number of components involved in *FanIn* and *FanOut*

(4)

Now we take an example. The Fig.1 shows a typical collaboration of the software components. Let us write the value for 'n' as per equation stated in (4).

n=3 (in our case we see in Fig.1 that the component A has 3 links attached to it; 0, 1 & 2 respectively) (5)

So for the value of n=3 equation (1) becomes

$$ar^0, ar^1, ar^2$$

Let us compute the value of 'a' for n=3

$$a = [(FanIn(A) * FanOut(A))^{n-1}]$$

$$= [(FanIn(A) * FanOut(A))^{3-1}]$$

$$= [(FanIn(A) * FanOut(A))^2$$

We can now represent three terms as below

$$a_1 = a*r^0$$

$$= [(FanIn(A) * FanOut(A))^2] r^0$$

$$= [(FanIn(A) * FanOut(A))^2]*1$$

$$= [(FanIn(A) * FanOut(A))^2 \quad (6)$$

$$a_2 = a*r^1$$

$$= [(FanIn(A) * FanOut(A))^2] * r^1$$

$$= [(FanIn(A) * FanOut(A))^2] * [-1/((FanIn(A)* FanOut(A))]$$

$$= - [(FanIn(A) * FanOut(A))] \quad (7)$$

$$a_3 = a*r^2$$

$$= [(FanIn(A) * FanOut(A))^2] * r^2$$

$$= [(FanIn(A) * FanOut(A))^2] * [-1/((FanIn(A) * FanOut(A))]^2$$

$$= 1 \quad (8)$$

Arranging up the values of a_1, a_2 and a_3 from (6), (7) and (8) we get the expression for statistical metrics index as

$$SS(A) = a_1 + a_2 + a_3$$

That is

$$(FanIn(A) * FanOut(A))^2 - (FanIn(A) * FanOut(A)) + 1$$

The value of $FanIn(A) * FanOut(A)$ can be treated as a constant *K* so we may further write statistical stream index as:

$$SS(A) = K^2 - K + 1;$$

In Fig.1 we find that the value of *K* is 1*2=2, so the complexity will be as:

$$SS(A) = 2^2 - 2 + 1 = 3;$$

Similarly if n=4, then the statistical stream index can be derived using the following expression as;

$$SS(A) = K^3 - K^2 + K - 1; \quad (9)$$

We should make a note that there is power component. This power containment indicates the nonlinearity of the complexity. The hypothesis is that if some component A is additionally complex than other component B then component B is much more complex rather than just a little bit more complex than component A. Given the supposition that we could raise to a power three or four or anything we want but on the standard that the simpler the model the better, then two is a good enough choice. From our point of view rising to, two makes it easier, as we will see, to pick out the potential bad guys. That is a good enough reason and we will leave it to the practitioners to concern about the finer detail.

3. POWER COMPONENT

We should make a note that there is power component. This power containment indicates the nonlinearity of the complexity. The hypothesis is that if some component A is additionally complex than other component B then component B is much more complex rather than just a little bit more complex than component A. Given the supposition that we could raise to a power three or four or anything we want but on the standard that the simpler the model the better, then two is a good enough choice. From our point of view rising to, two makes it easier, as we will see, to pick out the potential bad guys. That is a good enough reason and we will leave it to the practitioners to concern about the finer detail.

4. GUIDELINES FOR SSM

Statistical stream metrics can be piloted for any functional decomposition of a software system. We may consider structure charts, data flow diagrams and software development life cycle block diagrams. Apparently we may have to adapt terminology to suit the notation being used. For example, in a data flow diagram we do not have "calls;" instead we have data flows between processes. The principle is the same. One of the easiest applications we have come across is to use statistical stream metrics on the hierarchical directory structure as used in configuration amd change management system this is a fine case of the synergy that can sometimes be found to function within software engineering.

Statistical stream metrics provide a useful purpose right from high level design to the way down to low-level design when we can start to use A&D metrics discussed in previous section. Given the functional hierarchy we see that there is one additional attribute possessed by each component, that is its level in the hierarchy. The following is a step-by-step guide to deriving these most simple of SSM.

1. The level at which the component is in the design hierarchy should be noted.
2. *FanIn* is the number of message calls received by the component. On the level 1, we may have a single topmost component. This component may not have any *FanIn*, so we may assign a *FanIn* value of 1 to this component.
3. For every component, count the number of calls from that component. For components that do not communicate or call the other components, we can award a value of one to that *FanOut*.
4. The *SS* index value has to be calculated for every component by the use the above formula.
5. The sum of the *SS* values for all components contained by every level is calculated. This can be called as the LEVEL SUM.
6. The sum of the *SS* values for the total system design is calculated. We will call this the SYSTEM SUM.

In the analysis phase we can continue as further,

7. Each level may be checked on the basis of *SS* values along with *FanIn* and *FanOut* values.
8. The LEVEL SUM values at every stage are designed using a histogram or line plot

If our systems are superior to the ones we have seen then it will clearly get longer but remember that once done it is very easy to keep up to date. We should be able to automate the calculations depending on the environmental conditions.

Having got the data, we now need to do something with it. We must realize that, for statistical stream metrics, there are no absolute values of good or bad. Statistical stream metrics are comparative indicators. This means that value for our system may be higher than for a system we have but this does not mean that our system is worse. If we receive a high value of metrics then it does not necessarily mean that the code or component will not be or less reliable and will be difficult to maintain. It will be less reliable and less maintainable than its counterparts. In most systems, less reliable and less maintainable implies that it is most probably going to cost large amounts of money to fix and enhance. Most probably it could even be a terrifying component.

5. HANDLING FANIN & FANOUT

Statistical A bad component is the one that causes the system administrator nightmares. Because he or she knows that if that component is modified or edited, it causes the whole system to crash. Then it will take weeks to put it back in place because Fred designed it and Fred was weird. Fred also left five years ago! So the strength of statistical stream metrics is not in the numbers themselves but in how we use the data.

If we find more than 25% of the components with high value of the *FanIn*, *FanOut* and *SS* then we should rework on the values. If this collection is more or less than the 25% direct then do not worry about it. Also this should be noted that rather than getting fussy about 25% figure we should concentrate on high metric values.

High *FanIn* values indicate that there is less cohesion in the module. It may well be that we have not split out the functions to a great degree. Essentially, these components are called recurrently because they are doing more than one job. High levels of *FanOut* also point out a lack of cohesion or missed levels of abstraction. It is found that it is better to prefer *FanOut* rather than *FanIn*. But we should not disgrace *FanIn* [Mitchell,,2005].

High *SS* values means that there are highly coupled components. We need to look at these components for the *FanIn* and *FanOut* values to check out to reduce the complexity level. Sometimes we may hit a "traffic center." We have a potential problem area. If it is large component, it may be very much error-prone. If the complication cannot be reduced then at least we have to make sure that the component is tested thoroughly.

When we observe the LEVEL SUM plot; we should be able to see the values with a rather smooth curve illustrating controlled growth in data flow across the levels. If the values increase suddenly across levels is an indication of missed level of abstraction contained by the general design. It is found that where the design has less ten levels, we can consider count of components simply at each level that works fine.

The final item of data we have is the SYSTEM SUM value. This gives us an overall complexity rating for the design in terms of statistical stream metrics. There are a number of alternative design proposals [Kharb,2008]. Statistical stream metrics give users the opportunity to increase confidence in the choice they eventually make by quantifying aspects of complexity.

It should be noted that in order to use it for an organization the model will have to be tailored to the design mechanism of the organization. The basic difference involving the simple and the sophisticated statistical stream models lies in the definition of *FanIn* and *FanOut*.

For a component *A*, let:

a = Count of the quantity of components that call A.

b = Check the upper hierarchy components and see that how many of the upper hierarchy components are connected to the component A down in the hierarchy to see that how many parameters does component A receives.

c = Check the lower hierarchy components and see that how many of the lower hierarchy components are connected to the component A up in the hierarchy to pass parameters to the component A.

d = Quantity of data elements read by component A.

Then:

FanIn(A) = a + b + c + d,

Also let:

e = Count of the quantity of components called by A.

f = Check the upper hierarchy components and see that how many of the upper hierarchy components are connected to the component A down in the hierarchy to see how many lower level parameters are passed from the component A.

g = Check the lower hierarchy components and see that how many of the lower hierarchy components are connected to the component A up in the hierarchy to see how many upper level components receive parameters from the component A.

h = Quantity of data elements written to by A.

Then:

FanOut(A) = e + f + g + h

The derivation, analysis and interpretation remain the same, even if there are changes to the basic definitions. We must say that our advice to any organization starting to apply statistical stream metrics would be to build up confidence by using the simpler form. If these work for our organization then leave it at that. Further advisable is that if simple form does not work out then only we should go for the sophisticated or complex form.

We are encouraged by the fact that there have been a number of experimental validations done by us, of statistical stream metrics that seem to support the claims made for them. These results have been encouraging. Statistical stream metrics have are easy to be used and provides number of benefits so far as design and complexity are concerned.

6. DECOMPOSING HIERARCHY

In this section we will show that splitting the Fans we can reduce the complexity further. The designer should attempt for the development software structure with lower *FanOut* in the higher levels of hierarchy and more *FanIn* in the bottom levels of the hierarchy. A few examples of common modules which result in high *FanIn* include input & output modules, [Bucchiarone,2006].

Decomposing can be used to resolve the problem of unnecessary *FanOut.* An intermediary module can be used to factor out modules with strong cohesion and loose coupling. Consider when there are many lower level modules as shown in the Fig.2. As per the guidelines we have already discussed, it is always assumed that *FanIn* is not less than 1 in any case. So if there is no *FanIn*, we assign a value of 1.

Let us now compute the complexity of the module *A.*

$SS\ (A) = 4^4 - 4^3 + 4^2 - 4 + 1;$

$SS\ (A) = 256 - 64 + 16 - 4 + 1;$

$SS\ (A) = 205;$

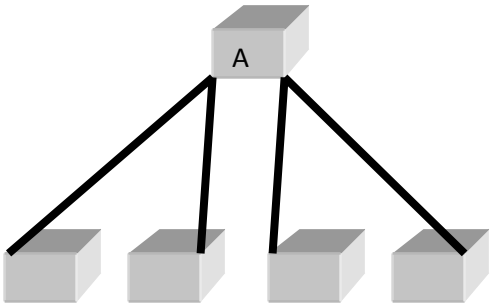

Fig 2: Several FANOUTs

This is pretty high. So in order to reduce it we can decompose the modular hierarchy as below shown in the Fig.3. We introduce an intermediate level and incorporate a module called *B.* Now the number of the lower level modules are shared or linked by two modules *A* and *B.* As a result complexity or coupling among the modules is decomposed. We find that in stead of four lower level modules coupled to a single high level module; now two lower level modules are linked to top level module and one intermediate module.

In the example (Fig.3 on the next column), FanOut is reduced by creating a module B to reduce the number of modules invoked directly by module A. In this example we see that without decomposing or splitting (using our statistical stream method) the FanOut from module A the complexity obtained is 205.

Outcome: Let us compute the index for the example shown in Fig.3, decomposing the FanOuts. Now we discuss the decomposed components. After decomposing (using our statistical stream method) component A in to the components A and B we get the following index. We find that there will be 2 components attached to A and 2 components will be attached to B.

SS (*A*) = (1*3)2-(1*3) +1

SS (*A*) = 32-3+1

SS (*A*) = 9-3+1

SS (*A*) = 7

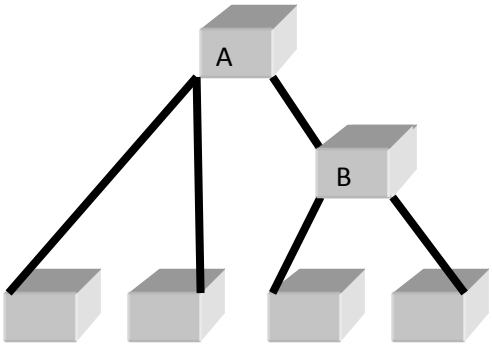

Fig 3: Decomposing FANOUTs

Similarly the SS index for the component B will be also 7. So the total complexity will be 7+7 = 14. So the total complexity will be the sum of the two that is; 7+ 7= 14. This is pretty low as compared to the original FanOut index arrangement without decomposing. The original we computed was 83. This is true other way round, also for the FanIns. If we split FanIns then also we will find that the complexity index is reduced.

This method is better than structured, cyclomatic complexity and essential complexity method as in this method it is easy to understand the rules and assess the complexity empirically and numerically. Further this method is more important as it focuses on component technology and better suitable for component based software development. So we may understand that this approach will be useful for object oriented paradigm. Finally as most of the software development is based on components and objects so it is very helpful in the regard that during design we can compute the complexity of components. Likewise it further adds to understand the collaboration among components based on the computed complexity so that we can also understand the complexity or risk involved in a particular implementation or programming environment.

7. COMPARING RESULTS

If we compare these results with the structure metrics complexity calculation done by Kafura then we find that statistical stream metrics is better approach [Henry,1981], [Kafura,1985]. Also mathematically geometric progression expresses the nonlinear nature of the coupling among components. So making use of this index is appreciable and anticipated. Let us first compute the complexity index as per the empirical formulae proposed by Henry and Kafura. In Fig.1, the complexity for the component A will be as follows.

Complexity (Kafura)

=(FanIn(A) * FanOut(A)) 2

= (1*2) 2

= 9

After this approach was found inconclusive, unrealistic and unreasonable the modifications were done. This approach was okay with the lower number of components but failed to give the correct results when the development was done with high number of components. Later on Henry and Selig [Henry,1999] suggested that apart from intra component connectivity there should be an element of internal component complexity. The new formula proposed was as follows.

Complexity(Henry) = Ci * (FanIn(A) * FanOut(A)) 2

Where, Ci is the internal (cyclomatic) complexity of the code. This value can be computed by McCabe formulae [McCabe,1976]. Suppose that the internal flowgraph of a code fragment is as shown in the Fig.4. The cyclomatic complexity will be;

Ci = e - n + 2;

Ci = 8 – 7 + 2;

Ci = 3

So finally the complexity becomes as;

Complexity(Henry) = Ci * (FanIn(A) * FanOut(A)) 2

Complexity(Henry) = 3 * 9

Complexity(Henry) = 27

Still it was found that this value is not acceptable as the real complexity is different. Our statistical stream metrics produce the realistic value due to considering nonlinear nature of the component collaboration. Complexity increases

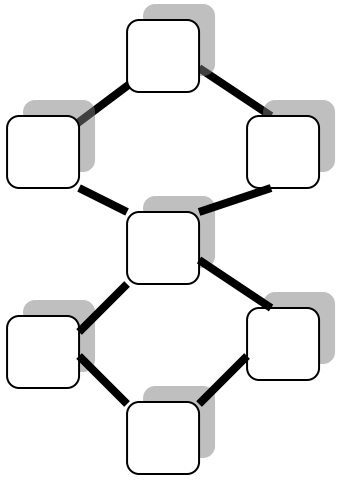

Fig 4: Flowgraph of Component

Also computing the value of the complexity using both internal and external complexity at the same time is not practical and thus not justified. The basic reason is that program code design and the component design are entirely different aspects of the software development. The cyclomatic complexity helps to work upon and thus reducing the complexity during the code (syntax) design for a particular code fragment where as structure metrics help to find out the complexity related to the calls given or received by the whole deliverable components. The cyclomatic complexity is useful for programmers and the structure metrics is helpful to architects, managers and deployment and maintenance team. In our statistical we have considered the coupling exclusively and results are more logical and trusted. The statistical stream metrics approach work fine with less number and high number of components as well.

8. COCLUSION

The metrics and the associated techniques in this paper do not give a total solution to the problem of complexity but they provide a start, and one that does not cost much to spend. To summarize this chapter, it is based on the assumption that avoidable complexity costs our industry a great deal of money. We can save money if we design hierarchy and complexity with a word of caution and then we can deliver good product to our testers and also to end users. This way we can save lots of money also. The techniques involving the metrics provide the opportunity to the organizations in order to improve quality and process including design.

If we observe the code carefully we conclude that there is redundancy of code or code fragments that may go up to 30%. That is an awful lot of code to be there when it does not need to be! To summarize, A&D metrics are found well established, well proven foundation for techniques to manage complexity and, hence, reliability and maintainability. Like them or not, we have found that they work. These metrics do open some other interesting options. The availability of the tools that support metrics is good. These are easily found through the web. A number of such tools also facilitate the automatic generation of unit test cases. This can offer a significant saving on testing effort.

Statistical stream metrics developed by us are of unique type (as compared to the existing available metrics) but, there exist some unwillingness in the industry to make use of statistical stream metrics. Probably the managers feel they are a bit "techie." Most probably they feel that they are not yet ready to use complicated techniques like statistical stream metrics. We expect that this concise research of the measures has shown that they are practical and pragmatic techniques of assuring quality [Sharma,2009] .

The foundation of statistical stream metrics is based upon the principle of FanIn & FanOut or component coupling. Most of the systems consist of components and it is the software performance that these components actually do. The way components are linked or associated together pretty much effect the complexity of a software product. If a component has to do a number of separate tasks it is said to be lacking in "cohesion." Also, systems are highly coupled, if the components within the system communicate data extensively with other components. Systems theory approach talks about that the components which are highly coupled and are less cohesive. These sorts of components with more coupling and less cohesion may be less reliable and difficult to maintain than those components that are loosely coupled and highly cohesive.

9. REFERENCES

[1] Bucchiarone A, Polini A, Pelliccione P, Tivoli M, "Towards an Architectural Approach for the Dynamic and Automatic Composition of Software Components," *Workshop On Role of Software Architecture For Testing And Analysis*, Portland, Maine, USA, July 17-20, 2006, pp 12-21

[2] Dantsin E, Eiter T, Gottlob G, Voronkov A, "Complexity and expressive power of logic programming," *ACM Computing Surveys (CSUR)* , Vol. 33, No. 3 , September 2001, pp. 374-425

[3] Henry S M, and Kafura D, "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering*, Vol. SE-7, 1981, pp. 510–518.

[4] Henry S M, Selig C, "Predicting Source-Code Complexity at the Design Stage," *IEEE Software*, *IEEE Computer Society*, March 1990, pp. 36–44.

[5] Kafura D, Canning J, "A Validation of Software Metrics Using Many Metrics and Two Resources," *The 8th International Conference on Software Engineering*, London, UK, August 28-30, 1985, IEEE, pp. 378 – 385.

[6] Kharb L, Singh R, "Complexity Metrics for Component-Oriented Software Systems," *ACM SIGSOFT Software Engineering Notes*, Vol. 33, No. 2, March 2008, Article No. 4

[7] Knuth D E, **"**Computer programming as an art," *Communications of the ACM*, Vol. 17, No. 12, December 1974, Pages: 667 - 673

[8] Lyu M R, "Software Reliability Engineering: A Roadmap," *Future of Software Engineering*, Minneapolis, MN, USA, May 23 - 25, 2007, pp.153-170

[9] Mathias K S, James H. C, Hendrix D,. Barowski L A, "The role of software measures and metrics in studies of program comprehension," *The 37th annual Southeast regional conference*, Mobile, AL, USA, April 15-18, 1999, Article No.: 13

[10] McCabe T J, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. 2, No. 4, December 1976, pp. 308–320.

[11] McCabe T J, Butler C W, "Design Complexity Measurement And Testing**,"** *Communications of the ACM*, Vol. 32, No. 12, December 1989, pp. 1415 - 1425

[12] Mitchell A, Power J F, "Using Object-Level Run-Time Metrics To Study Coupling Between Objects," *ACM Symposium on Applied computing 2005*, Santa Fe, New Mexico, USA, March 13-17, 2005, pp. 1456 - 1462

[13] Morasca S, "Refining The Axiomatic Definition of Internal Software Attributes," *Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, October 9-10, 2008, pp. 188-197

[14] Nagappan N, Murphy B, Basili V, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," 30th International Conference on Software Engineering, Leipzig, Germany, May 10-18, 2008, pp. 521-530

[15] Olender K M, Osterweil L J, "Interprocedural static analysis of sequencing constraints," *Transactions on Software Engineering and Methodology* (TOSEM) , Vol.1, No. 1, January 1992, pp. 21 - 52

[16] Pandey R K, "Managing Software Design Complexity: Facade Vs Role-Based Design," *SIGSOFT Software Engineering Notes*, Vol. 34, No. 1, January 2009, pp. 1-4

[17] Rensink A, Zimakova M, "Towards Model Structuring Based on Flow Diagram Decomposition," *The 1st Workshop on Behaviour Modelling in Model-Driven Architecture*, Enschede, The Netherlands, Article No.5

[18] Sharma M, Chandwani M, "Statistical Stream Metrics for Improving Quality of Analysis & Design," communicated.

[19] Stein C, Etzkorn L, Utley L, " Computing Software Metrics From Design Documents," *The 42nd Annual Southeast Regional Conference*, Huntsville, Alabama, USA, April 2004, pp. 146 – 151

[20] Virani S, Etzkorn L, Gholston S, Farrington P, Utley D, Fortune J, "Investigation of Domain Effects on Software," *The 47th Annual Southeast Regional Conference*, Clemson, South Carolina, USA, March 19-21, 2009, Article No.: 37

[21] Yourdon E, Constantine L, "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*," Prentice-Hall, Inc., February 1979