Measuring Complexity of Network and Service Management Components

Ognjen Prnjat, Lionel Sacks
University College London, Torrington Place, London WC1E 7JE, England, UK
email: {oprnjat | lsacks}@ee.ucl.ac.uk

Abstract

Software metrics are currently used in the industry mainly for cost and effort estimation, while some research suggested their use as fault indicators. There are very few empirical studies in software measurement, especially in the realm of object-oriented metrics. There is no record of management system assessment using software metrics. In this paper we discuss the use of the established object-oriented metrics as complexity/coupling and thus risk indicators early in the management system development lifecycle. Moreover, we subject a service management component designed in UML to the metric assessment, and present a detailed analysis of these measurements. The service management component was re-engineered in the context European Commission-sponsored ACTS research project (FlowThru). The measurement results indicate that the highest level of complexity, and thus also risk, is exhibited at the interconnection points between stand-alone components. Finally, the results imply a strong ordinal relationship between the metrics.

Keywords: Service management components, object-oriented metrics, UML.

1 INTRODUCTION

Telecoms network and service management systems are in their essence complex distributed software systems with many interdependencies. Designing, implementing and deploying such systems holds finite risk in terms of the impact of their stability on the overall stability of the underlying managed networks [Prnj99a][Prnj00]. Ability to highlight and remove potential risk areas in the management system's operation early in the development lifecycle would thus be greatly beneficial. In this paper, we discuss an approach to assessing complexity and coupling of the system classes using the well-established object-oriented metrics, with the aim of pin-pointing potential risk areas early in the design. We select a set of seven established object-oriented metrics, and describe how these can be deployed early in the development lifecycle. Moreover, we illustrate our approach with the case study of the ACTS project FlowThru subscription management component. We present our complexity/coupling measurements in detail. Considering the shortfall of the empirical metric studies, and no indication of previous management system assessment using metrics, we perceive the case study as a general contribution to the field. First, we introduce the current state of the art in software metrics. Then, we present our candidate early-lifecycle metric suite. Next, we present the FlowThru management system, with the focus on the subscription management component. Finally, we present the results of the assessment of the FlowThru subscription management component with our metric suite, and discuss the outcome and implications.

2 METRICS: BACKGROUND

Software measurement is a branch of software science dealing with measurement of internal and external attributes of software. Internal attributes are measured only in terms of the entity under observation, and they are measured directly, *i.e.*, independently [Fent94]. External attributes are measured in terms of how the entity relates to its environment, and they are measured indirectly - *i.e.*, measures of other attributes must exist so as to obtain the measure of an external attribute.

Software measurement is rarely applied in the industry: only 1-2 % of software organisations use metrics in the development process [Your96]. Applications in the industry focus on cost, productivity and effort estimation [Well94]. A set of metrics distinct from these process-oriented estimation metrics focuses on measuring the internal structure of software. These aim to capture the complexity of software modules and their dependencies. A number of pre-object-oriented complexity measures exist [Shep93]. With evolution of object-oriented (OO) design, a number of new complexity metrics emerged. The old metrics are not applicable to the OO paradigm, where the data and algorithms are bound together in a class, and a software program is a number of collaborating objects. OO structural complexity metrics are presumed to be collectable early in the development [Chid98] [Kami99], from analysis and design documents developed through a notation such as OMT [Rumb91] or Unified Modelling Language [UML]: moreover, some tools performing metrics collection have recently been developed [MetricsOne]. A number of OO metrics exist [Hend96]; in the following we list the most important ones.

Inheritance complexity is measured using Depth of Inheritance Tree (DIT) [Chid94] and Number of Children (NOC) [Chid94] metrics. Complexity of the inter-class relationships can be measured using the number of relationships [Li93] metric. Stand-alone class complexity is assessed using the Weighted Methods per Class metric (WMC) [Chid94], and the interface complexity metric [Hend96]. Relationship between classes can also be measured using the Coupling Between Objects (CBO) [Chid94], Message-Passing Coupling (MPC) [Li93][Lore94] and Response For a Class (RFC) [Chid94] metrics. Whitmire complexity metric [Whit97] quantifies the overall relationship complexity, including associations, aggregations, inheritance and message passing. The Lack of Cohesion of Methods (LCOM) [Chid94] measures the amount of cohesion in a class. The DIT, NOC, CBO, RFC, WMC and LCOM are collectively known as CK (Chidamber-Kemerer) metrics.

The CK metrics were suggested for prediction of external process attributes: productivity, re-work and design effort [Chid98]; testing effort and reuse [Chid94]; and maintenance effort [Li93]. In these studies, it was indicated that the metrics are effective for the assessment of these economic variables. As such, these metrics are considered as a managerial tool aiding project managers in effort allocation and planning. In [Chid94] these metrics were suggested to identify the design flaws and areas of re-design: however, no details were given. Another family of studies [Bria98][Kami99][Basi96] dealt with another aspect of metrics application: their relationship with fault-proneness. In [Basi96] CK metrics were shown to be better in predicting fault-proneness then other existing metrics. The metrics counts were related through a model to the binary value of fault-proneness: the class was detected during testing as either with a fault, or not. Measurements were performed on final code; the faults were recorded during testing.

There are only a few reported studies dealing with empirical OO measurements [Chid98] [Chid94][Li93][Bria98][Basi96][Kirs99]. In all these, CK metrics were collected directly from the code. Apart from one of the three studies in [Chid98], the only other reported study where it was attempted to collect the metrics from the analysis and design documents is [Cart96]. Here, however, most of the metrics proved to be difficult to collect from the design documents without having access to the implementation, with the exception of DIT and NOC. In [Kami99], use of metrics was suggested early in the development lifecycle; however, the source code of a mail system (141 classes) was used for metrics collection.

3 CANDIDATE METRIC SUITE

Measuring complexity and coupling early in the telecoms system development lifecycle can be seen as of high importance.

Since early years of software engineering [Cons79], to the modern days of OO software [Bern93] an axiom was established: good internal structure implies good external attributes of software. Good software should have low coupling between classes. Coupling is a measure of the degree of dependence between classes: "two classes are coupled if there is evidence that methods defined in one class use methods or instance variables defined in another" [Chid94]. Second, the stand-alone classes of good software should have high cohesion and low internal complexity. Cohesion is the extent to which the class is geared towards performing a coherent task [Cons79]. Internal class complexity could be concerned with either class internal structure (complexity of its control flow) or the complexity of the class as seen from the outside: complexity of its interface.

By locating and removing/redesigning points in the design which are highly complex, the telecoms software designer would avoid likely causes of software failure; and would minimise the likely fault propagation by reducing the coupling of the modules/classes. In this context, metrics can be seen as risk indicators early in the development lifecycle [Prnj99b].

Our early-lifecycle metrics suite [Prnj99b] consists of seven distinct OO metrics: Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Message-Passing Coupling (MPC), Response For a Class (RFC), interface complexity metric, and Whitmire complexity metric. All are class-level metrics. DIT [Chid94] is depth of inheritance tree: deeper trees constitute greater design complexity, and the deeper a class is in the inheritance hierarchy, more methods it inherits and more complex it is. NOC [Chid94] is defined as the number of immediate sub-classes subordinated to a class in the class hierarchy: classes with high NOC are more complex - they effect more classes. CBO [Chid94] is a count of a number of other classes that a class is coupled to. If a method in class A uses methods or instance variables in class B, then A is coupled to B. CBO is independent of the number of references that A makes to B. High coupling makes a class highly dependent on other classes and thus more vulnerable to error propagation and less reliable. MPC [Li93] is, in contrast to CBO, dependent on the number of references that class A makes to class B. MPC is defined as the number of send statements in a class. Large MPC implies large dependency on other classes: classes with high MPC have higher coupling and pose more risk to system operation. RFC [Chid94] is a set of all methods that can be invoked in a response to a message received by an object of a class, *i.e.*, the number of methods potentially available to the class. Large RFC indicates large complexity: tracing of dependencies becomes difficult, and coupling paths more intricate. The interface complexity [Hend96] assesses the stand-alone complexity of the class. Interface can be specified as a set of services: queries and commands. Interface complexity is the sum of weighted commands and queries: weight factor is number of arguments required for the query/command. The larger the interface size, the more difficult it is to select and correctly use the service provided by the class. Whitmire complexity [Whit97] assesses total class coupling within the design. It is a four-dimensional metric: dimensions are sets of inheritance, association, aggregation and message passing arrows related to the particular class. The magnitudes in each dimension are given by the cardinality of the relevant set of arrows.

This set of OO metrics can be calculated from analysis and design documents. DIT and NOC metrics can be calculated early in the lifecycle, considering the UML class diagrams depicting the inheritance hierarchy. The CBO, MPC, RFC and Whitmire complexity can be calculated once the interrelationships between classes are identified: UML class diagrams depicting associations and aggregations must be available, as well as the collaboration diagrams illustrating the message exchange between the collaborating objects. Interface complexity metric can be calculated once the stand-alone class interface has been specified, including the full set of parameters. These metrics were chosen as a representative set for the assessment of the analysis/design complexity of a system. We believe these metrics effectively capture the complexity of the design, tackling both the stand-alone class complexity, as well as different forms of inter-class coupling, ranging from inheritance coupling, through general relationship coupling such as association and aggregation, to message-passing oriented coupling which reflects the amount of interaction on the detailed level. Also, this metric set is representative because it includes the key metrics suggested in research. We omitted the stand-alone class cohesion measures (LCOM [Chid94]): those depend on the low-level class internal detail, available only through code-level information [Biem98]. These measures are thus not useful when considering the analysis and design information, which we are proposing to measure.

In the following, we present the FlowThru management system, with the focus on the subscription management component, which was used as a case study for complexity and coupling measurements.

4 FLOWTHRU MANAGEMENT SYSTEM

The ACTS project FlowThru focused on the reuse and integration of a number of components developed by the other ACTS projects - PROSPECT, REFORM, and VITAL. The aim was to demonstrate integrated multi-domain service and network management using a number of reused components, demonstrate integration technology at work, and specify development guidelines for reusable management components [Lew99b].

Development methodology, focused on building the reusable management components, and building systems from reusable components, was specified [Lew99a]. The reuse was incorporated in the lifecycle through specification of reusable components not only on the basis of their design and software, but also by including component's analysis model. Thus, flexibility is achieved since the component is more self-contained and as such does not have to be a part of any distinct framework. The reuse model is that of a façade [Jaco92]. Means of mapping between the façade and the ODP [ODP] viewpoint models are specified. The notation used is UML [UML].

Components specified in this manner provide a basis for developing a management system satisfying the target business process requirements. FlowThru identified three distinct trial business systems, based on the TeleManagement Forum's Telecom Operations Map process areas [NMF-TOM]: fulfilment, assurance, and accounting business systems. The components constituting these systems, in the FlowThru scenario, form the management system responsible for provision and maintenance of the ATM connectivity services.

The fulfilment business system aims at provision of the services. It consists of subscription, configuration, and network planning management components. The subscription component is responsible for introduction of new services available to the customer, and withdrawal of these services. It also enables both the service provider administrators and the customers to manage the end-user access to the service capabilities. Configuration management deals with network provisioning: it performs the configuration of network elements according to customer demands. Network planning component performs Virtual Path (VP) and route planning, considering the anticipated network traffic and customer demands. The assurance business system deals with the in-service problems that are encountered: it focuses on fault management, Service Level Agreement (SLA) violations, and the like. A number of components are encompassed, including service level accounting, TINA trouble ticketing, ATM accounting, and subscription management components. The accounting business system focuses on the accounting processes for the connectivity provider and the third party service provider. The TINA-based components include access session, service session, subscription, accounting, ATM accounting and connection management components.

The focus of our work is the subscription management component, which was the subject of software measurement presented here. Design of the component is based on the subscription model specified in the TINA service architecture [TINA-SA]; and was further refined, implemented and reused in the ACTS project PROSPECT. This component is located in the service provider's domain, and its basic role is to manage the subscription aspects of the service-level interactions between the customer and the provider. It is accessed by both the service provider's and the customer's administrators. This component manages the view of the services offered to the user: i.e., the definition and the list of available services. Secondly, it manages the subscribers' profile: it deals with the creation and deletion of subscribers, with their details, and the details of their network sites and user groups. Finally, it manages the process of customer's subscription to the services offered: creation and deletion of subscriptions, management of subscription details, and authorisation of the end-users access to the services. Thus, the component has the full knowledge of the classes of service provided, and the SLAs and service records that are part of the service offered. Customers can create a new subscription contract, they can modify and existing contract, modify a Service Usage Group (SUG) associated with an existing contract, and cancel an existing contract.

The analysis-level modelling of the component was conducted using the FlowThru methodology [Lew99a]. Since this component was already implemented, and its design already specified, the analysis model was developed post-facto, and from scratch. However, the existing component design was not followed in detail; rather, the generic requirements on this component were considered when building the analysis model.

The scenarios of the component use were modelled through the UML use case diagrams, depicting the interactions between the actors and the component. These were complemented with the UML class diagrams depicting the interrelationships between the analysis-level object classes: boundary objects (handling the communication between the component and the outside world), control objects (performing the use case specific behaviour) and entity objects (representing the information within the component) [Jaco92]. The consolidated analysis diagram is shown in Figure 1. This diagram does not depict the interactions between the Provider Administrator (PA) and the Customer Administrator (CA) with the component.

The analysis-level control and boundary objects correspond to the ODP computational objects (COs) and their interfaces, respectively. The main COs used to manage the services offered, the customer's subscriptions to these services, the customer profiles, and the service usage groups, are the Service Manager, Subscription Manager, Customer Account Manager and the Service Usage Group (SUG) Manager, respectively. The most complex interactions within the component were modelled using the UML collaboration and sequence diagrams.

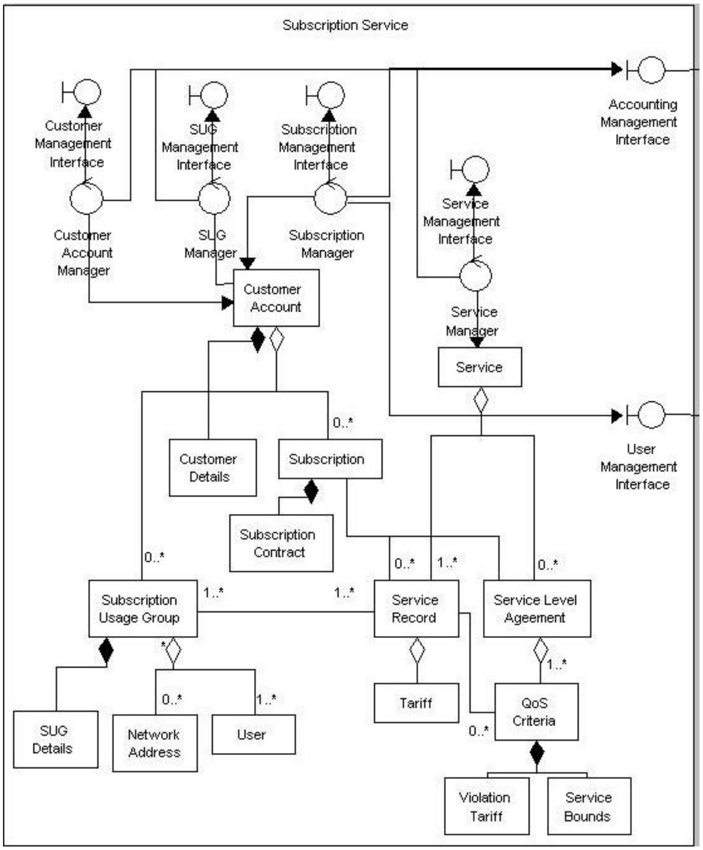


Figure 1 - The consolidated analysis class diagram [Flow-http]

The design and implementation of the component already existed prior to component's use in the FlowThru scenario. Thus the design-level model of the component was re-documented using UML. The design model is comprised of the "static" and the "computational" models. The static model is a set of UML class diagrams representing in more detail the entity objects from the analysis model, and depicting their relationships. These entity objects are referred to as t-type objects. The computational model shows the core functional units - computational objects (mapped from the analysis model control objects), represented as packages which export the interface (i-type) objects (which in turn are mapped from the analysis model boundary objects). The entity objects are linked to functional units that manage and use them. Both analysis and design level entity objects effectively correspond to ODP information objects. While most of the analysis to design mappings were reported [Lew99a] to be one-to-one, in some cases the one-to-many (when an object is decomposed in the design phase) and many-to-one (when two or more objects in the analysis were identified, having similar functionality) mappings also occurred.

5 METRIC ASSESSMENT: ANALYSIS AND DESIGN MODELS

The analysis model was developed *post-facto*, considering the generic requirements for the component. The analysis model is based on [Jaco92] framework, and is represented by a set of 24 UML diagrams [Flow-http]. The assessment of the FlowThru analysis model involved 4 metrics out of the 7 proposed in the suite. There is no inheritance in the FlowThru analysis model, and thus the inheritance metrics, DIT and NOC, are not applicable. The interface complexity metric is also not applicable, since although the interface methods were defined, the parameters were not. The FlowThru analysis model consists of 28 classes. The data collection was performed manually.

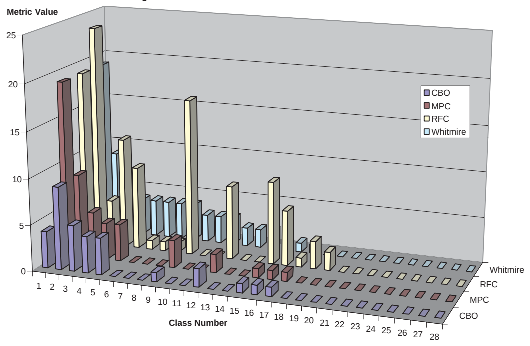


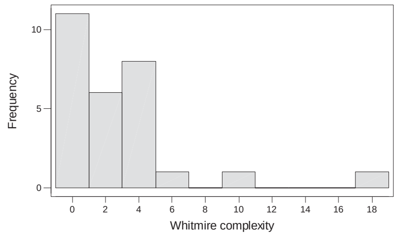
Figure 2 - Metrics distributions per class (FlowThru analysis)

Figure 2 graphically depicts the metrics distributions per class. On average, the most complex class is the Provider Administrator (PA) interface - the boundary/interface class of the provider management application, which controls the subscription management component. Next, the four control/computational object classes follow: the Subscription Manager, Service Manager, Customer Account Manager and the SUG Manager. The boundary/interface object classes are next on the complexity scale, the Subscription Management Interface exhibiting particularly high complexity. Finally, the purely information (entity) object classes representing the key data entities follow.

As an illustration, for Whitmire complexity metric, we present the basic summary statistics (mean, median, standard deviation, minimum, maximum) (Table 1), the histogram of the distribution of metric values (Figure 3, left), and the boxplot of metrics values (Figure 3, right). The boxplot depicts the centre and variation of the data set, and marks the outliers. It is constructed from the three summary statistics: *median* (value *m* for which half the values of data set are smaller then *m* and half are bigger), the *upper fourth* (value *u* which is the median of values larger than *m*), and *lower fourth* (value *l* which is the median of values smaller than *m*). Values *m*, *u* and *l* split the data into quarters. The box length is d = u - l, and upper tail value is u + 1.5 d. The outliers are marked with a star. Thus, the boxplot shows the skewness of data, by the position of the median in the box, and by the length of the tail. Here, median is off-set of the centre and the tail lengths are unequal (left being non-existent); the data set is strongly skewed to the left. This metric identifies the PA interface as the most complex, followed by the four main control/computational classes - Subscription Manager standing out.

Mean	2.536
Median	1.5
Standard deviation	3.892
Minimum	0
Maximum	19

Table 1 - Whitmire complexity statistics (FlowThru analysis)



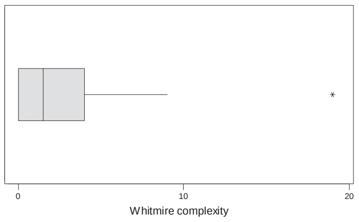


Figure 3 - Whitmire complexity histogram (left) and boxplot (right) (FlowThru analysis)

All four metrics used to assess the analysis-level FlowThru management component indicate that the two most complex classes are the PA interface - the interface from the provider management application to the subscription management component, and the Subscription Manager - the main control/computational class in the component. The highest complexity thus seems to be is exhibited at the interfaces between stand-alone components.

The design model was simply reverse-documented using UML on the basis of the existing implementation. It consists of 108 UML diagrams [Flow-http]. The assessment of the design model involved 4 metrics out of the 7 proposed in the suite. There is no inheritance in the FlowThru design model, and thus the inheritance metrics, DIT and NOC, are not applicable. The MPC metric is also not applicable, since collaboration diagrams illustrating the message exchange between the collaborating objects are not included in the design documents. The design model consists of 103 classes. The data collection was performed manually.

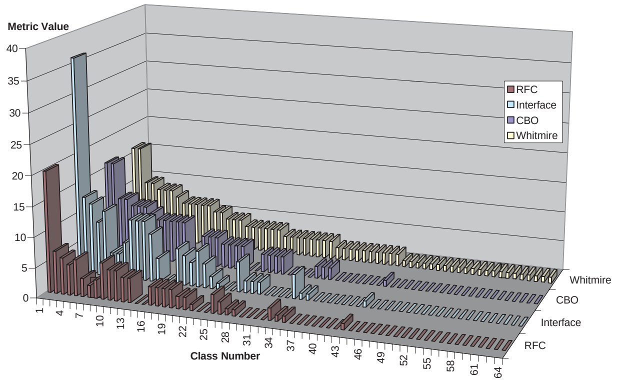


Figure 4 - Metrics distributions per class (FlowThru design)

Figure 4 graphically depicts the metrics distributions per class. Classes with all the metric values of 0 are not included in the diagram. On average, the most complex class is the i_DB_sag, the main Database interface class. However, for the purpose of this discussion we ignore all the Database (DB) interface classes, and concentrate on the classes directly relevant for the realisation of the design target functionality.

All four metrics used to assess the FlowThru management component design indicate that the most complex class is the i_subscrnInfoQuery, the interface class derived from the analysis-level Subscription Management Interface, as the most complex one. All of the metrics next point out the other interface classes derived from analysis level Subscription Management Interface, followed by the interface classes derived from the SUG Management Interface and the Service Management Interface analysis classes. The purely information (t-type) object classes representing the key data entities exhibit much lower complexity counts.

Considering this result, we can note that the complexity of the design-level packages, derived from the analysis-level control/computational classes, is hidden behind the interfaces they export. This was not the case in the analysis model, where the interface complexities were trailing after their corresponding control/computational class counterparts. However, the complexity measurements do propagate evenly through the development phases. In the design, the interface classes derived from the analysis Subscription Management Interface are the most complex. This is analogous to the analysis model situation, where the most complex class, apart from the PA interface which is not described in the design, was the Subscription Manager. Similarly, the most complex design-level information object class (or t-type class in the FlowThru design terminology) is the t_AssignGroupSelection, derived from the most complex information object class in the analysis: the Subscription Usage Group.

The most complex class, i_subscrnInfoQuery, is the outside interface to the main control/computational class in the component. Thus, the complexity measurements of the FlowThru design again re-iterate the point raised in the FlowThru analysis study: the highest complexity is exhibited at the interfaces between the stand-alone components.

6 DISCUSSION

Metrics data exhibits distinctive features, including non-normal distributions and multicollinearity. Thus, we conducted a statistical analysis of measurements, so as to trace interesting trends in data. This is followed by a more general discussion of the experiment.

The typical metrics distribution (for all metrics) is strongly left-skewed, with a few outliers distinctly standing out. Since the distributions are non-normal, the basics statistics such as mean and the use of parametric statistical methods do not accurately capture the distribution features and the interrelationships between the distributions. In the case of non-normal distributions the use of robust statistics (such as median and ranks) and nonparametric statistical methods is advocated [Schn92][Fent91]. Assumptions for the use of nonparametric statistical methods are much less restrictive then for the parametric methods (which usually assume normal distributions, equality of variances across samples, *etc.*). However, the nonparametric methods are as rigorous, and allow the analysis of order relations [Schn92].

The strong relationship between the whole set of metrics is a problem referred to as *multicollinearity*. Thus, we concentrated on investigating the associations between the metrics. First, we investigated the linear association between the metric values, by calculating r, the *linear (Pearson) correlation coefficient* between each pair of metrics. The linear correlation coefficient is a descriptive measure of the linear (straight-line) relationship between two variables. This is the typical approach to testing for metrics interrelationships, commonly reported in literature [Chid98] [Li93]. The linear correlation coefficients for FlowThru analysis and design measurements are high for each pair of metrics within one isolated experiment. The majority of correlation coefficients are higher than 0.8. Similar result was reported in [Chid98], where the correlation between the metrics was mostly higher than 0.8. These results should indicate that the regression equations linking each pair of metrics are highly suitable for making predictions of one metric on the basis of the other (*i.e.*, one metric should be a good linear predictor of the other). Then, the *coefficient of determination*, or r^2 (where r is the linear correlation coefficient), would give the quantitative measure (percentage) of the amount of variation of one metric (*response variable*) explained by the variations in the other metric (*predictor variable*).

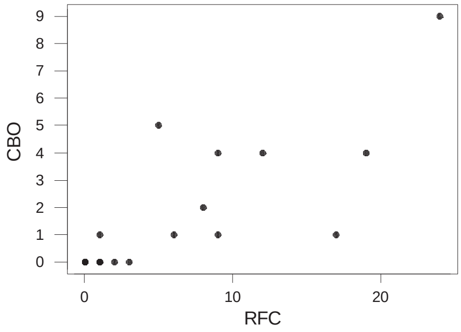


Figure 5 - Scatter plot of CBO versus RFC (FlowThru analysis)

However, by examining the scatter plots we concluded that the data points are very weakly scattered about a straight line: one of the scatter plots is shown in Figure 5 (CBO plotted against RFC). Once this is the case, we cannot make definitive statements concerning the usefulness of one metric as a linear predictor of the other. The assumption for both the linear correlation coefficient and for finding the regression equation is that the data points are clearly scattered about the straight line [Weis99]. Also, regression is sensitive to the presence of outliers, which appear to be a distinctive feature (legitimate data points) of metrics distributions (as discussed above) and as such can not be justifiably removed. Moreover, we can not use regression inferences because the two basic conditions supporting the assumptions for regression inferences are not met. The first is that the plot of the residuals against the values of the predictor variable (*residual plot*) should fall in a horizontal band centred and symmetric about the x-axis. The second is that the normal probability plot of the residuals should be linear. *Residual* is the difference between the observed and predicted value of the response variable. The plot of the FlowThru analysis CBO residuals against the RFC values is shown in Figure 6, left. The residuals do not fall in a horizontal band. The normal probability plot of the residuals is shown in Figure 6, right: the plot is not straightforwardly linear. Thus, we conclude that the assumptions for regression inferences are violated.

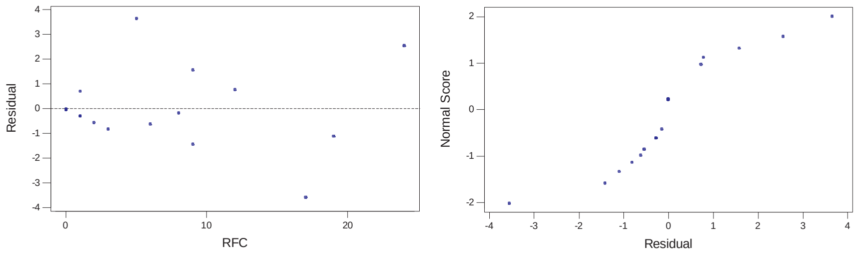


Figure 6 - Residuals versus RFC (response is CBO) (left) and the normal probability plot (right)

Considering the weak linearity of scatter plots, and the violations of the regression inferences assumptions, we conclude that we cannot claim that any pair of metrics can be used to determine the regression equation (linking the two metrics) from which meaningful predictions can be made. We can say that there is enough statistical evidence to doubt the possibility that the magnitude ordering of one metric directly implies the linear magnitude ordering of the other, and that the intervals between the two values of one metric are proportional to the intervals between the values of the other metric, for any pair of adjacent classes that these measurements refer to.

An alternative approach to determine the relationship between the metrics is through the use of nonparametric statistical methods. The nonparametric methods can be used to avoid rigorous assumptions such as linearity. The nonparametric method equivalent to linear correlation is the calculation of the *rank (Spearman's) correlation coefficient* between paired values (from same classes) of the two metrics. This procedure effectively lowers the metrics scale from interval to ordinal, avoiding the magnitude-related relationships between metrics [Schn92]. In this procedure, we use ranks of metrics rather than the metrics values themselves. This loosens up assumptions about data relationships (linearity), while still giving a valid measure - ranking of classes according to the metrics value. The rank correlation coefficients for FlowThru analysis and design measurements are shown in Table 2. All rank correlation coefficients are significant at the 95% confidence level.

	CBO	**MPC**	**RFC**	**Whit.**
CBO	1			
MPC	0.993	1		
RFC	0.846	0.859	1	
Whit.	0.614	0.626	0.626	1

	CBO	**RFC**	**Whit.**	**Inter.**
CBO	1			
RFC	0.994	1		
Whit.	0.749	0.742	1	
Inter.	0.993	0.998	0.741	1

Table 2 - Metrics rank correlation coefficients: FlowThru analysis and design

The metrics rank correlation coefficients are high. This indicates that there is a strong ranking relationship between the metrics. Thus we can say that there is enough statistical evidence to say that *each metric could be useful in predicting the ranks of the other metrics* (more or less, depending on the value of the rank correlation coefficient). Exceptionally high (>0.95) rank coefficient is exhibited between CBO and MPC, and between RFC and interface complexity. This indicates that any combination of: CBO or MPC; RFC or interface complexity; and Whitmire complexity metrics would form a set of three complementary metrics.

We consider the FlowThru metrics experiments to be an empirical research contribution in their own right. As discussed before, there are few reported studies dealing with the practical system evaluation using the OO metrics. In the majority of reported studies, the metrics (CK) were collected from code, despite the fact that they were originally envisaged [Chid94], and later advertised [Kami99], as earlier lifecycle measures. The one of the only two reported studies involving the design documents [Cart96] reported problems with metrics collection - only DIT and NOC were collected successfully. Moreover, metrics were never applied to network and service management systems.

Our experiments are the first to assess a management system using metrics. Although the system assessed originates from a research project, we consider it as representative since a number of professional organisations participated in the design. The system was chosen due to the public availability of design documentation, which is not the case in the industrial organisations.

The experiments demonstrated that the metrics collection from the analysis and design documents is possible. However, not all of the metrics can be collected early in the development lifecycle: the analysis-level documentation allows collection of Whitmire complexity, CBO, MPC, RFC, NOC and DIT; while the interface complexity can be collected only at the detailed design stage, since it requires full interface elaboration. It is also generally believed that the ability of metric collection to some extent depends on the development framework and the notation used. However, we demonstrated that through thorough use of established diagrammatic techniques such as UML metrics collection becomes easy (note that some tools such as MetricsOne [MetricsOne] have been recently developed to ease the collection of metrics from design documents).

A number of general observations can be made concerning the results of our case studies. First, the system did not contain any inheritance. This might be due to either the fact that the development team was not accustomed to the OO design philosophy, or it could be a reflection of the size of the system, which can be considered as small-scale. However, low inheritance measures (DIT and NOC) were also reported in a number of earlier studies [Chid98] [Cart96] [Basi96].

Second observation is that the CBO counts appear distinctly low, as compared to the MPC, RFC and Whitmire complexity counts, which also depict the coupling between objects. This is mainly due to the fact that these other three coupling measures actually include the amount of collaboration between classes, while the CBO accounts for the number of collaborating classes. However, the CBO counts are still much lower than in previous studies reported in the literature, which can be due either to the size of our system, or the fact that the designers, using the design heuristics, aimed at minimising the number of collaborating classes.

Next, all the metrics have a typical non-normal distribution: strongly left-skewed, with a few outliers distinctly standing out. Also, metrics are highly correlated in terms or ranking of the classes. Particularly high rank correlations are exhibited between CBO and MPC; and RFC and interface complexity.

The final observation is that the majority of classes have distinctly low metric counts. In case of FlowThru analysis 53% of the classes have the average complexity between 0 and 2, while in the FlowThru design this number rises to 76%.

All of the metrics used thus identify a subset of classes as distinct from others in terms of complexity. These classes are most usually the main computational object classes in the system, performing a manger-control task. Also, the highly complex classes can be the most important information object classes in the system. The most complex classes singled out in our case studies were the classes operating at the interfaces between the stand-alone management components. These most complex classes were singled out by all of the metrics making up our metrics suite. These measurements indicate that the highest risk for the systems' integral operation is exhibited at the major interconnection points.

The metric suite in these experiments was used simply as an analysis tool for assessment, *i.e.* diagnostics of the most complex classes. These classes are then labelled as the classes with highest risk. The re-design was not applied during development, because the system was assessed *post-facto*.

7 CONCLUSION

In this paper, we have suggested the use of established object-oriented software metrics for assessment of the management system design early in the development lifecycle. The complexity and coupling measurements would give an early indication of the potential risk areas in the system design. This information would be valuable in the context of modern, highly distributed network and service management systems, whose correct functioning is of paramount importance for the operations and maintenance of the underlying managed telecoms network.

We used seven existing software measures to form a metric suite that yields the complexity and coupling measurements of the system classes. We demonstrated the usefulness and applicability of this approach through a case study of the FlowThru subscription management component: both analysis and design models. Apart from being one of the rare empirical case studies of the object-oriented measurement, this experiment is the first one to assess a telecoms management system using the object-oriented metrics. The case study empirically demonstrated that the highest complexity, and thus also risk, for the management systems' operation is exhibited at the interconnection points between the stand-alone components. Finally, the experiment assessed the nature of the interrelationship between the individual metrics within the metrics suite, uncovering a strong ordinal relationship between the metrics.

8 REFERENCES

[Basi96] V. R. Basili, L. C. Briand, W. L. Melo, “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transactions on Software Engineering, Vol. 22, pp. 751-761, 1996.

[Bern93] E. V. Bernard, “Essays on Object-Oriented Software Engineering”, Prentice-Hall, 1993.

[Biem98] J. M. Bieman, B. Kang, "Measuring Design-Level Cohesion", IEEE Transactions on Software Engineering, Vol. 24, No. 2, pp. 111-124, February 1996.

[Bria98] L. C. Briand, J. Daly, V. Porter, J. Wust, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", Proceedings of the Fifth International Software Metrics Symposium, pp. 246-257, 1998.

[Cart96] M. Cartwright, M. Shepperd, “An Empirical Investigation of Object-Oriented Software in Industry”, Technical Report TR96/01, Bournemouth University, 1996.

[Chid94] S. R. Chidamber, C. F. Kemerer, “A Metrics Suite for Object-Oriented Design”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.

[Chid98] S. R. Chidamber, D. P. Darcy, C. F. Kemerer, “Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis”, IEEE Transactions on Software Engineering, Vol. 24, No. 8, August 1998.

[Cons79] L. Constantine, E. Yourdon, “Structured Design”, Prentice-Hall, 1979.

[Fent91] N. E. Fenton, “Software Metrics - A Rigorous Approach”, Chapman and Hall, 1991.

[Fent94] N. E. Fenton, “Software Measurement: A Necessary Scientific Basis”, IEEE Transactions on Software Engineering, Vol. 20, No. 3, March 1994.

[Hend96] B. Hendson-Sellers, “Object-Oriented Metrics, Measures of Complexity”, Prentice-Hall, 1996.

[Jaco92] I. Jacobson, “Object-Oriented Software Engineering - A Use-Case Driven Approach”, Addison-Wesley, 1992.

[Kami99] T. Kamiya, S. Kusumoto, K. Inoue, "Prediction of Fault-proneness at Early Phase in Object-Oriented Development", Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '99), pp. 253-258, 1999.

[Kirs99] C. Kirsopp, M. J. Shepperd, S. Webster, "An Empirical Study Into the Use of Measurement to Support OO Design", Proceedings of the 6th IEEE International Metrics Symposium, IEEE Computer Society, 1999.

[Lew99a] D. Lewis, C. Malbon, A. DaCruz, "Modelling Management Components for Reuse using UML", Proceedings of the 6th International Conference on Intelligence in Services and Networks, Springer-Verlag, Berlin, 1999.

[Lew99b] D. Lewis, "A Software Development Methodology for Service Management", Published in Telecom'99 Forum.

[Li93] W. Li, S. Henry, “Object-Oriented Metrics that Predict Maintainability”, Journal of Systems and Software, Vol. 23, pp. 111-122, 1993.

[Lore94] M. Lorenz, J. Kidd, “Object-Oriented Software Metrics”, Prentice-Hall, 1994.

[MetricsOne] http://www.numbersix.com/Extras/MetricsOne.asp

[NMF-TOM] Network Management Form, "Telecoms Operations Map", NMF GB910, Stable Draft 0.2b, April 1998.

[ODP] ITU Draft Recommendation X.901-X.904, "Basic Reference Model of Open Distributed Processing"; Part 1: "Overview and Guide to Use", 1995; Part 2: "Foundations", 1995; Part 3: "Architecture", 1995; Part 4: "Architectural Semantics", 1995.

[Prnj99a] O. Prnjat, L. Sacks, "Integrity Methodology for Interoperable Environments", IEEE Communications, Special Issue on Network Interoperability, Vol. 37, No. 5, pp. 126-139, May 1999.

[Prnj99b] O. Prnjat, L. Sacks, "Telecoms System Design Complexity and Risk Reduction Based on System Metrics", Proceedings of the 10th European Workshop on Dependable Computing (EWDC10), May 1999.

[Prnj00] O. Prnjat, L. Sacks, "High Integrity Inter-Domain Management", in A. Galis (Ed.), "Multi-Domain Communication Management Systems", CRC Press - USA, ISBN: 084930587X, June 2000.

[Rumb91] J. Rumbaugh et. al., “Object-Oriented Modelling and Design”, Prentice-Hall, 1991.

[Schn92] N. F. Schneidewind, "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, pp. 410-422, May 1992.

[Shep93] M. Shepperd, “Software Engineering Metrics, Vol. 1”, McGraw-Hill, 1993.

[UML] Rational Software Corporation, Unified Modelling Language, http://www.rational.com/

[Weis99] N. A. Weiss, "Introductory Statistics", Addison-Wesley, 1999.

[Well94] E. F. Weller, "Using Metrics to Manage Software Projects", IEEE Computer, Vol. 27, No. 9, pp. 27-33, 1994.

[Whit97] S. A. Whitmire, “Object-Oriented Design Measurement”, John Wiley and Sons, 1997.

[Your96] E. Yourdon, "Rise and Resurrection of the American Programmer", Prentice-Hall, 1996.