2002

Investigation of an Oracle application's high-level design

Scott W. Schield
The University of Montana

Investigation of an Oracle Application's High-Level Design

by

Scott W. Schield

B.A. Psychology, The University of Montana, 1997

presented in partial fulfillment of requirements

for the degree of

Master of Science

Computer Science Department

The University of Montana

August 2002

Approved by:

Joel Henry

Chairperson

[illegible]

Dean, Graduate School

8/1/02

Date

Schield, Scott W. M.S., August 2002 Computer Science

Investigation of an Oracle Application's High-Level Design

Director: Joel Henry [illegible]

This project investigates the high-level design of an Oracle application. In order to complete this task, the high-level design needed to be documented. After the development of the high-level design, the Fan-In and Fan-Out for each module was recorded. The Fan-In and Fan-Out is a metric that measures both the complexity and coupling of a software application. The values obtained from the Fan-In and Fan-Out were compared against reported defects. It is generally thought that complexity and coupling of a module is associated with the amount of defects found in that module (i.e., as complexity and coupling increase then defects will increase). The results of this project did not identify a clear association between complexity, coupling and defects. However, other factors are likely to increase the effort needed to maintain this system. These factors include duplicate code, database updates across modules, and the absence of coding standards. These issues were uncovered during code review, but were not included in this investigation.

Table of Contents

List of Tables

List of Illustrations

hical User

Oracle is the largest vendor of database management systems. Oracle can be used on both MS Windows and UNIX operating systems. Some of the benefits of using Oracle are[1]:

1. Large quantities of data can be stored and updated.
2. PL/SQL can be used to retrieve information from tables.
3. Allows access to other software that is written in a different language (C/C++).
4. Data can be shared with other users.

Besides the database, Oracle also provides a development environment known as Oracle Designer xi. Oracle Designer xi features a toolset that allows a developer to model, generate and capture the requirements and design of a client/server application. Oracle Designer 6i will also allow the developer to asses the impact of changing the database design or application. The toolset allows the software development to take place directly inside Designer. Some of the benefits of using Oracle Designer xi are[2]:

1. Central repository for all developers.

2. Accurate analysis of system requirements.
3. Auto-generate databases and module definitions.
4. Design capture and development.

Background

As mentioned before, the Oracle application is a large-scale, commercial, Graphical User Interface (GUI) based software system. The system is being used by a Montana State Government Agency. Due to the nature of the application, both the Montana Government Agency and the software development company cannot be identified. For the rest of the paper the Montana Government Agency is referred to as Agency and the software development company is referred to as Development Company.

Typically, an outside company develops large-scale, commercial applications. Outside companies that feature large-scale, application development are IBM and Unisys. A company, like IBM, is contacted by the Agency and asked to submit a bid for the specified development. The company that submits the lowest bid for development is usually chosen to carryout the application development.

Due to the cost of developing software and the funds available to the Agency, the Agency decided to go with the lowest bid. The contract was awarded to the Development Company in 1997. The Development Company delivered the software application to the Agency in December of 1999.

The Development Company met all the requirements, including delivery time, according to the contract set-forth by the Agency. However, the software application featured a large amount of defects. The contractual requirements concerning the quality of the software application were vague. As one should understand, the Agency and the Development Company interpreted the vague quality requirements differently. The agreement reached between the Agency and the Development Company consisted of the Development Company providing more personnel beyond the contractual requirements in order to stabilize the software system.

The Development Company provided the extra software engineers through the summer of 2000. Since the summer of 2000, the Agency is still striving to maintain stability with the software system.

Project Goals

The goal of the project was to analyze the software design and relate the findings to the current defects within the software system. By performing this type of analysis, it would help to pinpoint areas of high-risk. These high-risk areas are parts of the software system that may be prone to higher amounts of defects and possibly affect the maintenance of the system.

CHAPTER 2: Approach & Solution

As described before, the goal of the project was to analyze the software design and relate the findings to the current defects within the software system. The design of the project was limited to a high-level system design. This type of design is acceptable when trying to understand the complexity of the whole software system without regards to specific functions, data types and data structures. One important measure that can be obtained from a high-level design is the amount of coupling between modules and the complexity of modules.

Project Plan

1. Establish a direct connection with the Agency's server. This would allow work to be performed from the University of Montana and not on-site in Helena, Montana.
2. Establish privileges that would allow access to the software application code.
3. Become familiar with Oracle Designer 6i. This was necessary, as the Agency's development environment is Oracle Designer 6i.
4. Manually record the 'call to' and 'called by' for each module. This would allow the representation of the high-level design to be formed.
5. Develop the high-level design in Microsoft Visio 2000.
6. Apply coupling and complexity metrics to the high-level design. This would show modules that would be considered to have a high-level of coupling and complexity.
7. Group and count the defects for each module analyzed.

8. Perform analysis on the coupling and complexity measures against the defect count.

The following chapters will discuss the development environment, overview of the software development process, how the project was carried out and results of the project.

CHAPTER 3: Development Environment

As discussed earlier, Oracle provides a toolset know as Designer 6i. The Agency performs all their development inside Designer 6i. The initial design done for this project was a high-level design of the Oracle Forms based inside Designer 6i.

"Oracle Forms is a 4GL Rapid Application Development (RAD) environment. Forms Builder is used to create applications to enter, access, change, or delete data from an Oracle database[3]." The software engineers create modules that support the business rules set-forth in the requirements. The modules allow the end-user to enter, update, and query information in an Oracle database and communicate with other modules via PL/SQL commands. Modules feature code that is similar to code found in other programming languages, such as data validations, calculations, error checking and algorithms to carry out the business rules.

Oracle Modules inherently, by design, allow for built-in error checking. Even though the user is working directly with database information, the user's workspace is local. No data is written to the Oracle database until the user tells it to write to the database. This allows the user to make mistakes without jeopardizing the information in the database. However, it is worth noting that once the user commits to inserting information into the database there is no way to go back.

This system under study features business rules that could be grouped together in very distinct areas. Some of these areas were named: Accounting, Registration and Returns Processing. Each distinct area had Oracle modules that carried out the business rules. The number of Oracle modules found in each distinct area varied in size from 1 to 58. There were 19 distinct areas which accounted for 213 Oracle modules.

CHAPTER 4: Software Development Process

The software development process is more than just coding and delivery of the software. The process is a well thought-out approach that when followed correctly will lead to a well-developed product. This Chapter will present an overview of the development process and the specific approach used by the developing company.

Process Overview

There are many different process models for developing software. A software process model guides the software engineers through the entire development of the application. The software process model encompasses requirement analysis, design, implementation, testing and delivery. Some of the software process models used in development include: Waterfall Model, Prototyping Model, and the Spiral Model. These are just a few of many software process models used to guide software engineers. Each software process model has positive and negative benefits, but the key is to choose the process model that best fits the area of development.

At the system level, the developers begin to develop a system view for the intended application. This is done by establishing requirements for all system elements and is a critical step since this application was interacting with a database. At this level, the requirements gathered are at the system level and some top-level design and analysis is performed.

The analysis of software requirements is one of the most essential parts of the development procedure. To understand the system, it is up to the software engineer to understand the software domain, the intended function, behavior, performance and user interface. The requirements gathered for both the system and software are written into a Requirements Document. Before the actual coding starts, the software engineers and customer must agree on the content of the Requirements Document.

Developing the software design is the next stage of the development process. During the design phase, the software engineers begin to develop and build a representation of the software based off the requirements. The design is assessed for quality before the actual coding begins. It is important to note that the design is maintained throughout the software development process, and provides the software engineers with a visualization of the software structure.

After the design is completed and is determined to be of good quality the software engineers begin the coding phase. The code for the software system is based off the software requirements and design. The software engineers are now going from the software representation to implementation of the software design.

Once the coding of the system is completed, the next phase is to test the software. The different types of software testing that can be performed are a massive topic and are only touched on briefly in this paper. The overall goal of testing is to verify that the software is meeting the requirements for the system. Any errors found during testing are fixed before the system is delivered.

Once the software has been delivered to the customer, the developing company will provide support for an extended period. Support is given for installation, maintenance and fixing any defects that may arise during that period. Once the negotiated support time has elapsed, it is up to the customer to provide or contract future support of the software.

Generally, software development involves the five stages listed above. Each software development model will approach how the five stages are carried out in a different manner. For example, the Prototype Model will complete the initial design, coding and testing very rapidly. The prototype design is then presented to the customer for feedback. The software engineers then take the comments and change the software to fit the customer's needs. The Development Company that produced this software followed the Waterfall model of software development.

Specific Process

As stated before this Oracle application was designed by following the Waterfall model of development. The Waterfall model can be thought of as a linear and sequential method of software development[4]. The development begins at the system level and sequentially moves to analysis, design, coding, testing and support.

There are many development models that are judged better than the Waterfall model, but this model is the oldest and mostly widely used software development model. However, there are a few problems when following this model of development[4].

1. The model implements sequential development. This means before moving onto a new phase that current phase must be

completed. This rarely happens in 'real-world' software development. By its inherent design, any changes during software development are harder to incorporate.

2. For the Waterfall model to work successfully, all the requirements must be known before the actual development takes place. Most customers do not know the entire requirement for their desired system. Any changes to the requirements after development has started are hard to incorporate.
3. The final fallback of using the Waterfall model concerns prototyping and a pre-beta version of the system. Because the development phase is linear and sequential, it is not possible to show the customer a working model of the system. The customer does not have the ability to see the product until it is almost delivered. Any mistakes made during the development concerning requirements and design are difficult to correct.

The Development Company also chose to have another company develop the majority of the software. This is known as outsourcing in the Software Industry. Outsourcing, to a third party, allows the development to take place at a lower cost while usually maintaining integrity within the system. Typically, outsourcing involves hiring a smaller company to carry out the development. This smaller company usually has less over-head and this leads to cheaper development costs. Today, a large majority of the outsourcing is granted to 'off-shore' developers. A large amount of 'off-shore' developers are based in India. The price to develop software in India is very low when compared against outsourcing the development to a stateside company. It is important to note that 'off-shore' developers generally meet or exceed standard software development practices.

Pressman[4] reports that outsourcing is either a strategic or a tactical decision. The strategic decision involves business managers determining if a significant or all the development can be performed outside the company. The tactical decision involves a project manager determining if a reasonable amount or all the development can be subcontracted. Regardless of the reason, outsourcing the software development process has both positive and negative effects.

The most significant positive benefit is a reduction in cost. When the development process is outsourced, the cost to support the development (software engineers and facilities) can drastically be reduced. The Development Company planned to outsource the development from the beginning, and was able to submit a much lower bid to the Agency. The main deciding factor of the Agency awarding the bid to the Development Company was their ability to deliver a product based on the Agency's requirements for such a lost cost. The majority of the other software development companies submitting bids to the Agency were not able to achieve this level of development for the price.

A negative effect with outsourcing is the lack of control over the development process. Without good, concrete software development methods in place, a company who sub-contracts, is risking its integrity. A few steps can be taken to avoid the pitfalls associated with outsourcing. It is necessary that the development company stay in close contact with the outsourcing company. This can be achieved by having an on-site software manager working closely with the outsourcing team. If feasible, the customer should also perform these same actions. This will help ensure that the outsourcing development company is meeting the requirements of the system and following the best software engineering practices. The Agency and the Development Company appear to have struggled to do this on a consistent basis. This resulted in a loss of control over the development process. The loss of control over the development process was evident when the application was delivered to the Agency.

CHAPTER 5: Software Design

As stated previously, one of the steps in software development involves developing a design of the system. This chapter will address why software design is important, what type of design was provided to the Agency, what was done to develop a current design and what was done with the current design.

Why Software Design?

Software is prevalent in every phase of our life. From our homes to our offices, we are surrounded with products that are run by software (e.g., cars, traffic lights, microwaves, etc...). One may ask how so many complex systems actually work. The products that work usually have been well designed.

Everyday on our way to work we drive on roads, cross bridges, and enter buildings at our place of employment. It would be unrealistic to think that a civil engineer just showed up on the site without any design and started building the bridge. It is natural to think that the finished bridge would be lacking in quality. Would it be safe to drive over? Highly doubtful even if the civil engineer was to achieve such a task. Would it be possible for a civil engineer to build a walking bridge over a small creek without any design? It is realistic to admit that this would be feasible and should be of high quality when completed. If a team of software engineers sat down at Microsoft and decided to code the latest version of Microsoft Office without a design, then would you buy it? If a student decided to code a program that printed "Hello World" without a design, then what would the quality be concerning that program? Software engineers are no different from the civil engineers. When the complexity of a system increases the importance of developing, a design increases.

The Agency's requested system should have started with the Development Company engineering a design document. A software design encompasses four areas of a program[4]. These areas included data structures, software architecture, interface representations, and procedural/algorithmic detail.

The software design translates the software requirements into a visual representation of the software. The design, allows the over-all quality to be assessed before the actual coding of the application takes place. Changes to the design can be made to improve quality. The design is carried through the entire development process and is typically delivered with the final system.

Delivered Design

This Oracle application was delivered to end user with little design documentation. As discussed earlier, a design document is vital for maintaining a software application. Being that this was an Oracle application, the Agency should have received a database design and a design of the PL/SQL application logic. At the time the project started, there was no evidence that any design documentation currently existed.

This system delivered in December 1999, has undergone numerous changes through the summer of 2002. During this time, the Agency's software engineers continued to make these changes in the system without fully understanding the impact that those changes would have on other areas of the system.

Current Design Development

The design of the application was limited to a high-level system design. This type of design is acceptable when trying to understand the complexity of the whole software system without regards to specific functions, data types and data structures. What follows are the steps that were taken to develop the high-level design.

1) Log into network

The first step was to log onto the Agency's network. This involved establishing a connection with the Agency's server and providing a username, password and logon domain.

2) Log into local desktop environment

Novell software controls the login for the local desktop environment. It was necessary to provide a username, password, and working server name to access the local desktop environment. The desktop environment is identical to the desktop found if a software engineer was working directly from inside the Agency's offices. The password was changed every three weeks.

3) Log into Oracle Designer 6i

Oracle Designer 6i holds all the source code for this Oracle based application. Again, to obtain access it is necessary to provide a valid username and password combination. Along with the username and password, it is also necessary to enter a valid database connection string. The database connection string is the database related to the Oracle application.

It was necessary to choose a work area when accessing Designer 6i. Choosing a work area is similar to choosing where a developer wishes to perform changes. It would be similar to choosing where you want to save a file. However, not to get confused, this is not saving. For this project, a separate work area was provided. This allowed a separate working space to perform the duties of this project. Once the work area is chosen, then the Designer 6i interface is presented.

4) Access Oracle Designer 6i Editor

The Designer 6i interface is a standard Windows GUI. The interface presents a wide range of choices for the user. For this project, the choice was Oracle Design Editor. The Design Editor allows a user to explore all the code written for that particular Oracle Application. The interface of the Design Editor is similar to the Microsoft Windows Explorer window. The folders are presented in a tree-like structure and can be expanded to show their contents.

When Design Editor appears the software engineers is presented with four tabs.

1. Server Module
2. Module Applications
3. Database Administration
4. Distribution and Replication

For this project, I was interested in the 'module tab'. This 'module tab' provides all the code for the modules in the application.

5) Obtaining the calls

To represent the high-level design, it was only necessary to obtain the 'called to' and 'called by' for each module. The easiest way to obtain the 'called to' and 'called by' for each module is to perform an individual design of each module. However, none of the individual designs could be saved. Furthermore, Designer 6i contains many design tools that aid the software engineer. However, there were privilege issues involved to use these tools. In order to access some of these design tools, and save the module diagrams the user must be granted the privileges to checkout the modules out of the Agency's source control. Due to the sensitivity and secrecy of this application, it was not possible to checkout the pertinent information. It was possible to view the information, but not edit or save any added information.

The following steps were taken to record the 'called to' and 'called by' for each module found in Design Editor:

1. A list of modules was recorded in MS Excel. Consultation was carried out with an Agency software engineer to ensure only applicable modules were recorded in the list.
2. A new module diagram can be viewed in the following manner:
    a. File -> New -> Module Diagram. A drop-down box appears

    b. Select 'APP_PTDB' form the drop-down box and click 'OK'. A system folder, Windows-like Explorer box appears.
    c. Inside the Explorer box, expand the 'APP_PTAP' folder. A list of modules appears.
    d. Select a module and click 'OK'. A single module diagram appears and shows the 'called to' and 'called by' for each module.
3. When the diagram was presented the following information was recorded in an Excel spreadsheet:
    a. The module name.
    b. Whom the module calls ('called-to').
    c. Who calls that module ('called-by').

   The Agency's application featured many distinct areas of business rules. The spreadsheet was designed to delineate the different areas (e.g., Accounting, Registration, etc…) of the Oracle application. This spreadsheet could then be used to develop the design on a local computer.
4. The process was then repeated for each of the 213 modules. This was necessary because Designer 6i does not allow a software engineer to mass generate each module diagram.

6) Implementing the design

After, all the 'called to' and the 'called by' for each module were recorded the next phase of the project took place. This phase of the project involved diagramming the high-level design. The high-level design provides a structural framework for a more detailed design. In other words, the high-level design is the foundation for a more detailed design. It is important to note that the goal of this project was not to develop a detailed design. The detailed design needs to be completed by software engineers who work with the system on a daily basis.

As with any business application, one of the most important areas of processing is accounting. For the Agency, the Accounting areas were the most important and contained very complex business rules. The Accounting area was the starting point for the high-level design.

The high-level design was developed in Microsoft Visio 2000. Visio 2000 was chosen because the design can easily be modified and can be exported to HTML pages for easy viewing by all users[5]. The design was carried out for each of the distinct areas. After the design of a distinct area was completed then the design was reworked. Reworking a design increases readability and facilitates understanding of the system when viewed by other software engineers. When all 213 modules were incorporated into the design, the system was reworked three times to increase readability and understanding.

CHAPTER 6: Fan-Out & Fan-In

After the design was completed, it was necessary to interpret the diagram to gain a better understanding of the design. This was done by measuring the Fan-Out and Fan-In for each module in the high-level design. Both Fan-In and Fan-Out measure the complexity and coupling of a system.

Fan-Out

Fan-Out provides a measure of the number of modules that are directly controlled by another module (Figure 1). Two modules are considered coupled if methods declared in one module call methods or access attributes defined in another module. This information was readily available by examining the number of ‘called to’ for each module.

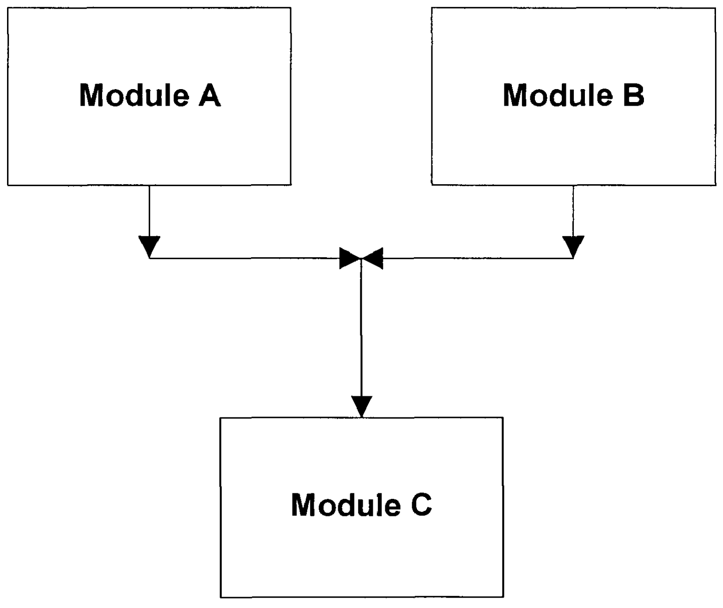

Figure 1: Fan-Out

Modules that have a high Fan-Out value (greater than 5) are considered to be highly coupled (e.g., highly interconnected)[6]. It is important to note that this is a very loose implementation of calculating the Fan-Out and all results obtained from this metric are general in nature. However, the benefit is that it pinpoints possible modules that may have high coupling. Software Engineers could then take this information and perform more in-depth design of these areas in order to formulate results that are more concrete. The Fan-Out results for each module are presented in Appendix A.

Fan-In

When consulting with Dr. Joel Henry it was decided that, another metric should be applied to the high-level design. The goal of applying a new metric was to understand the overall complexity of the system. The new metric consisted of measuring the Fan-In for each module. The Fan-In measures how many modules are calling a particular module. For example, if module A was called by modules B and C. Then module A would have a Fan-In of two. That is considered a Fan-In depth of one (Fan-In (1)) for module A. It was also taken into account the Fan-In depth to a level of three for any particular module.

Example

1. Module A called by Module B and Module C.
2. Module B called by Module D.
3. Module C called by Module E.
4. Module D called by Module F.
5. Module E called by nothing.

6. Equation

    a. Total Fan-In = Fan-In (1) + Fan-In(2) + Fan-In(3)

    b. A Total Fan-In = A Fan-In (1) + A Fan-In(2) + A Fan-In(3)

    c. A Total Fan-In = 2 + 2 + 1

    d. Module A Total Fan-In = 5

At this point further clarification is need for Fan-In. The easiest way to conceptualize the idea of Fan-In is to consider a path to that particular module. Given that Module A had a total Fan-In of 5, then there are five unique paths to Module A up to a level of three. Consider the following:

1. Module A Fan-In(1) = 2. Two paths can be taken to reach Module A directly.
2. Module A Fan-In(2) = 2. Two paths can be taken to reach Module A at one level of indirection.
3. Module A Fan-In(3) = 1. One path can be taken to reach Module A at two levels of indirection.

Recording the Fan-In for each of the 213 modules was a bit more difficult then recording the Fan-Out. Calculating the Fan-In for each module was carried out in the following way:

1. First, evaluate the value of the 'called by' for each module.
2. If the value was equal to zero, then zero was recorded for the Fan-In of that module.

3. If the value was greater than zero, then a new Visio 2000 diagram was started for that particular module if it was not easy to deduce the total Fan-In for that module.
4. The diagram was then expanded up to a depth level of three and the total Fan-In was recorded.

CHAPTER 7: Defects

After the Fan-In and Fan-Out were recorded, the next step involved analyzing the defects found in the system.

The Agency burned the defects onto a cd and the defects were logged into an Excel spreadsheet. The defects in the spreadsheet were defects logged between 10/14/1999 and 09/26/2001. Before defects could be compared against the Fan-In and Fan-Out results, it was necessary to standardize the recording process. This required ensuring that all module names were in a consistent format (e.g., PTAC001F, PTCM001F, etc...). This was done to increase the readability of the document and to ensure correct filtering of the spreadsheet when looking for a particular module.

Once the defects were in a standard format, it was then possible to track the defects for each module. The defects were recorded in the following manner:

1. Filter the defects and obtain a count for each module in the high-level design.
2. Record the critical defects for each module

There were 1851 defects for the modules in the high-level design. Out of those 1851 defects, there were 610 defects that were categorized as critical defects by the Agency.

Once the defects were recorded, it was then possible to perform some statistical analysis on the defect counts versus Fan-In and Fan-Out.

CHAPTER 8: Results

The final step in the project was to determine if there was a relationship between the defects reported by the Agency versus the Fan-In and Fan-Out results obtained from the high-level design. It was thought that as the coupling and complexity increased for a module, then the defects would increase for that particular module. This would be consistent with what is found in a Software Development process[4]. Typically, a module that is highly coupled would produce more defects due to the interconnectedness. To obtain the relationship between defects versus Fan-In and Fan-Out a correlation measure was applied to the data.

Coupling and Complexity Breakdown

As stated previously, a Fan-Out of five is considered to have high coupling. Since the Fan-In went to a depth level of three, then any module having a Fan-In greater or equal to 15 would be considered complex. The modules meeting these criteria are presented in Table 1 (Fan-Out) and Table 2 (Fan-In). The findings show that the system has few modules that would be considered to be highly coupled and complex.

Module	Fan-Out
PTAC002F	7
PTCM021F	8
PTCM022F	10
PTCM025F	7
PTCM029F	8
PTCM002F	7
PTCM003F	12
PTRG009F	7
PTRP004F	5
CM_MENU	6

Table 1: Modules withHigh Coupling (Fan-Out)

Module	Fan-In
PTAC026F	15
PTCM067F	16
PTRG015F	16
PTFC101L	19
PTCM007F	21
PTCM002F	23
PTCM004F	23
PTCGLIB1	25
PTCM032F	25
PTER101L	25
PTCM016F	27
PTAC007F	28
PTCM017F	31
PTCM025F	31
PTCM029F	31
PTCM021F	32
PTRG010F	32
PTRG005F	33
PTCM003F	41
PTCM022F	48
PTAC027F	50
PTCM018F	50
PTCM010F	58
PTCM005F	73
PTCM102L	74
PTAC101L	85
PTCM101L	189
PTGN081L	266

Table 2: Modules with High Complexity (Fan-In)

The Fan-Out and Fan-In was then broken down for each distinct area. The breakdown consisted of summing the total Fan-Out and Fan-In for each distinct area and dividing it by the total number of modules in that distinct area. This breakdown would show if modules in a distinct area were highly coupled and complex (Table 3). The results show that very few areas present modules that were highly coupled and complex.

Module	Fan-Out Mean	Fan-In Mean
PTAC	1.33	5.89
PTAT	0	0
PTCG	1	25
PTCM	2.9	27.41
PTDI	0	0
PTER	1.28	3.57
PTFC	0.61	0
PTGN	0.2	45.2
PTOG	0	0
PTNP	0	0
PTPA	0	0
PTRG	1.16	7.16
PTRP	1.19	1.81
PTSC	0.67	0
PTSF	0	0
PTSM	0.32	0.05
PTTK	0.5	0
ACCOUNTING	0	1
CM_MENU	6	1

Table 3: Distinct Area Mean

Why a Correlation?

The values for each module can be represented in X and Y pairs. The defects are always the Y value while Fan-In and Fan-Out are always the respective X value. The project investigators were interested in knowing whether defects versus Fan-In and Fan-Out were related, and if so, then in what capacity or strength of association. One key in choosing a correlation is evaluating if there are any independent variables in the data. "An independent variable is the variable that is controlled or manipulated by an experimenter so that its effect on a dependent variable can be determined[7]." If this project presented an independent variable, then the choice would have been a regression measure. Listed below are three reasons why a correlation measure was chosen[7]:

1. There is no independent variable.
2. The values of X or Y were not pre-selected and thus both X and Y may vary freely.
3. Interested in assessing the strength association between X and Y and possibly predicting either variable from a knowledge of the other.

Types of Correlations

The degree of association between two variables is represented by a correlation coefficient. The correlation coefficient can range in values between -1.0 to +1.0. A value of +1.0 denotes a perfect positive relationship between two variables. This is represented with all the data points falling on a straight line such that high values of X are paired with high values of Y and low values of X are paired with low values of Y (Figure 2).

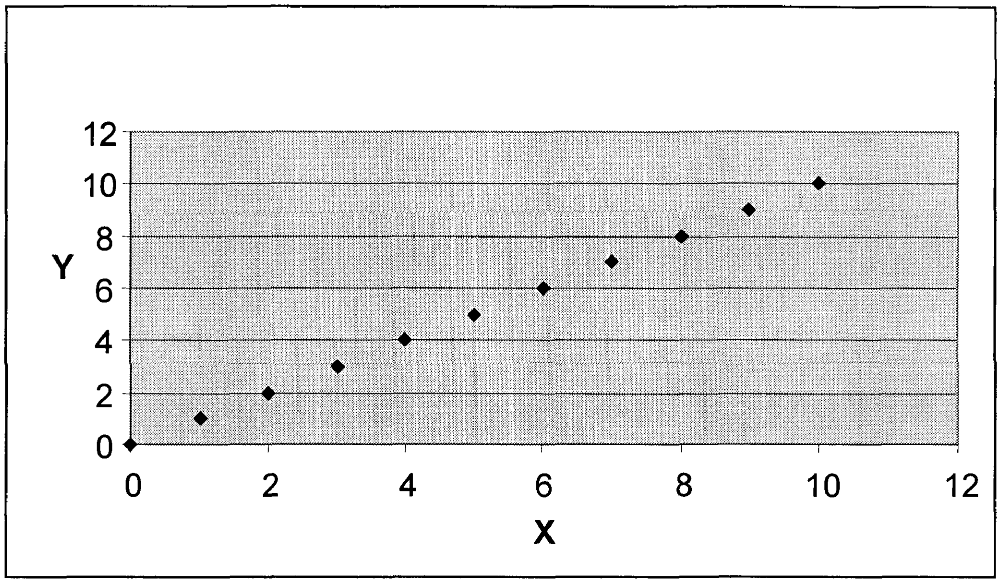

Figure 2: +1

A value of -1.0 denotes a perfect negative relationship between two variables. This is represented with all the data points falling on a straight line such that low values of X are paired with high values of Y and high values of X are paired with low values of Y (Figure 3).

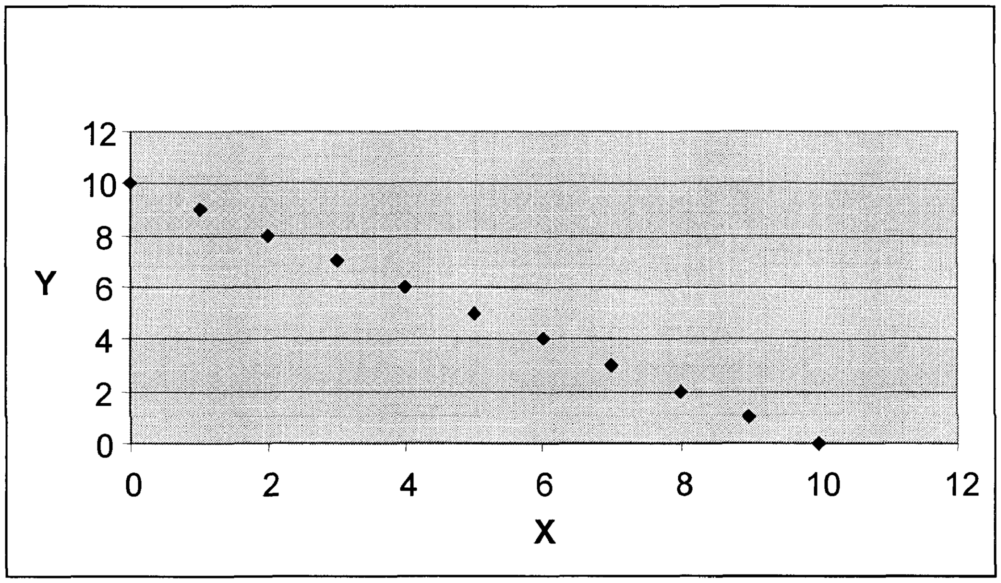

Figure 3: -1

If the value is zero, then there is no linear association between the variables. This can be represented with all the data points falling on a horizontal line or in a circular fashion (Figure 4).

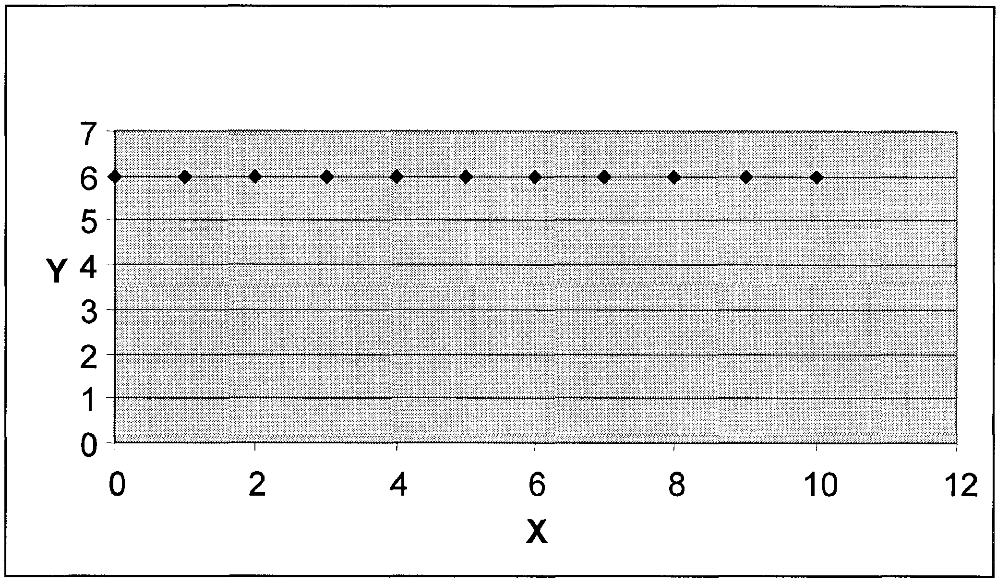

Figure 4: 0

The correlation used for this data was the Pearson Product-Moment Correlation Coefficient. The correlation coefficient produced is appropriate for describing the linear relationship between two quantitative variables.

Fan-Out Correlation

Appendix A show the module Fan-Out count and the amount of defects associated with each of those modules. Before finding a correlation coefficient, it was necessary to produce a scatter plot. The scatter plot would show if some linear relationship exists between Fan-Out and defects. The scatter plot is presented below.

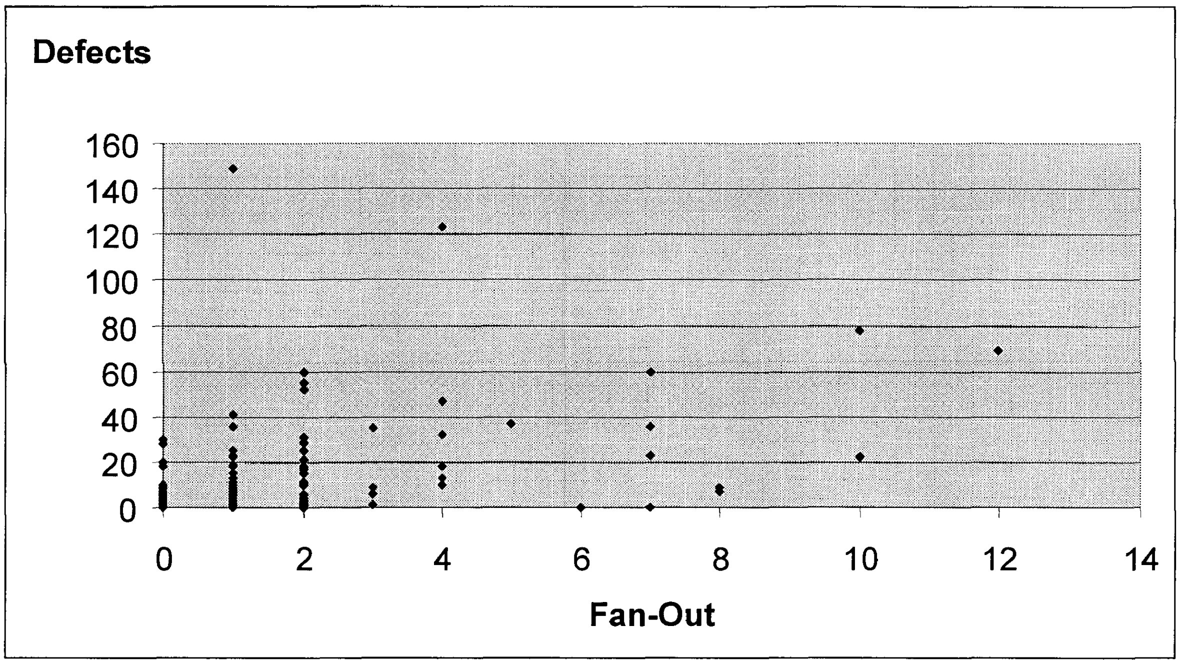

Figure 5: Scatter Plot

From the scatter plot, it seemed that there might be a positive correlation between Fan-Out and defects. The next step was to find the correlation coefficient between these two variables.

The formula for the Pearson Product-Moment Correlation Coefficient is presented below[7]:

$$r = \frac{\sum_{i=1}^{n} XiYi - \frac{\left(\sum_{i=1}^{n} Xi\right)\left(\sum_{i=1}^{n} Yi\right)}{n}}{\sqrt{\left[\sum_{i=1}^{n} X^2 i - \frac{\left(\sum_{i=1}^{n} Xi\right)^2}{n}\right]\left[\sum_{i=1}^{n} Y^2 i - \frac{\left(\sum_{i=1}^{n} Yi\right)^2}{n}\right]}}$$

Equation 1

The values for the formula are listed below.

$$X = Fan - Out$$
$$Y = Defects$$
$$\sum Xi = 219$$
$$\sum Yi = 1851$$
$$\sum XiYi = 5412$$
$$\sum X^2 i = 1001$$
$$\sum Y^2 i = 85743$$

The computation of the correlation coefficient for Fan-Out and defects is listed below.

$$r = \frac{5412 - \frac{(219)(1851)}{213}}{\sqrt{\left[1001 - \frac{(219)^2}{213}\right]\left[85743 - \frac{(1851)^2}{213}\right]}}$$

$$r = \frac{3508.859}{\sqrt{[775.831][69657.549]}}$$

$$r = 0.4773$$

Fan-In Correlation

Appendix A show the module Fan-In count and the amount of defects associated with each of those modules. Before finding a correlation coefficient, it was necessary to produce a scatter plot. The scatter plot would show if some linear relationship exists between Fan-In and defects. The scatter plot is presented below.

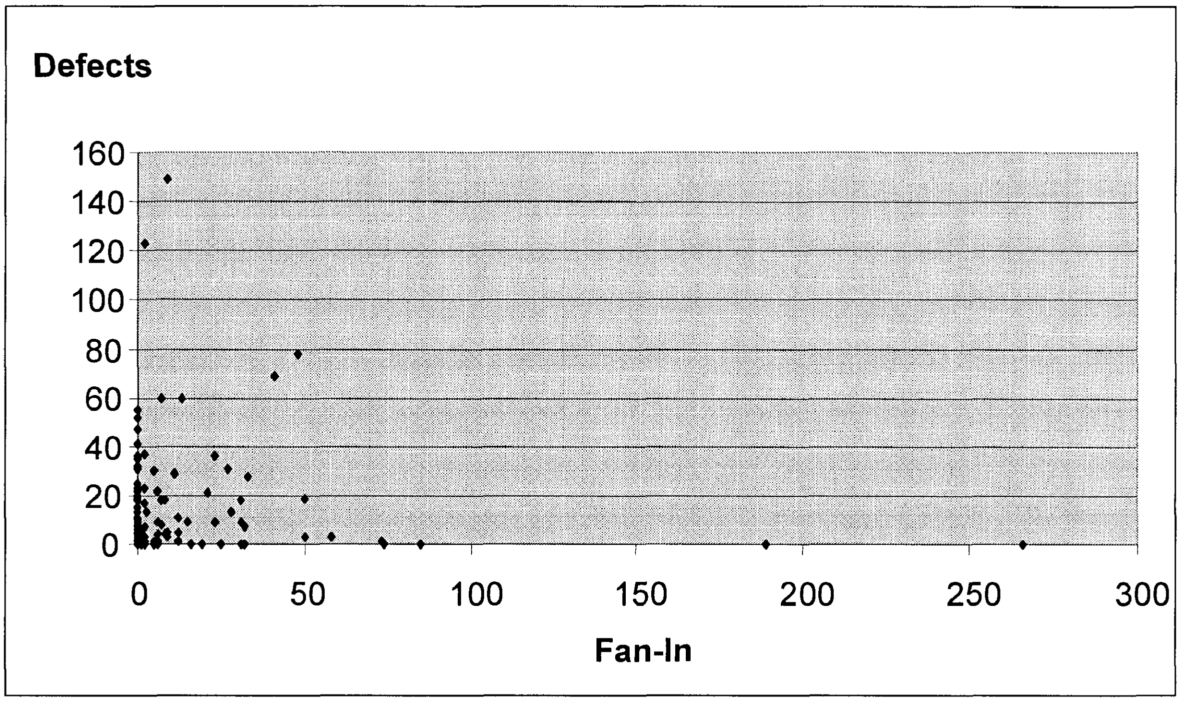

Figure 6: Scatter Plot

From the scatter plot, it appears that there might be a positive correlation between Fan-Out and defects. The next step was to find the correlation coefficient between these two variables.

The values for the formula are listed below.

$$X = Fan - In$$
$$Y = Defects$$
$$\sum Xi = 1581$$
$$\sum Yi = 1851$$
$$\sum XiYi = 17056$$
$$\sum X^2 i = 150403$$
$$\sum Y^2 i = 85743$$

The computation of the correlation coefficient for Fan-Out and defects is listed below.

$$r = \frac{17056 - \frac{(1581)(1851)}{213}}{\sqrt{\left[150403 - \frac{(1581)^2}{213}\right]\left[85743 - \frac{(1851)^2}{213}\right]}}$$

$$r = \frac{3316.887}{\sqrt{[138667.971][69657.549]}}$$

$$r = 0.033$$

Interpretation of r

The correlation coefficient obtained from the Pearson Product-Moment gives the nature and strength of the linear association between two variables. However, there are two other statistics, both functions of *r*, that present a better intuitive sense for the strength association represented by *r*. These statistics are the coefficient of determination (r^2), and the coefficient of nondetermination (k^2)[7].

The coefficient of determination will give the proportion of variance of one variable that is explained by the variance of the other variable. The coefficient of nondetermination will give the proportion of variance that is not explained by the variance of the other variable. The calculations for the respective statistics are shown below.

$$r^2 = (0.4773)^2$$
$$r^2 = .2278$$

Fan-Out Coefficient of Determination

$$k^2 = 1 - (0.4773)^2$$
$$k^2 = .7722$$

Fan-Out Coefficient of Nondetermination

$$r^2 = (0.033)^2$$
$$r^2 = 0.001$$

Fan-In Coefficient of Determination

$$k^2 = 1 - (0.033)^2$$
$$k^2 = 0.999$$

Fan-In Coefficient of Nondetermination

1) Fan-Out r

The results obtained from the coefficient of determination and nondetermination is interpreted in the following manner.

1. 22.78% of the variance of the defect scores can be explained by the linear relationship between this variable and the corresponding Fan-Out scores.
2. 77.22% of the variance of the defect scores is not explained by the linear relationship with the Fan-Out scores.

The linear relationship between Fan-Out and defects does not account for much of the variance in the defect scores (22.78%), but instead 77.22% of the variance is not accounted.

1) Fan-In r

The results obtained from the coefficient of determination and nondetermination is interpreted in the following manner.

1. 0.10% of the variance of the defect scores can be explained by the linear relationship between this variable and the corresponding Fan-In scores.
2. 99.9% of the variance of the defect scores is not explained by the linear relationship with the Fan-In scores.

The linear relationship between Fan-In and defects accounts for minimal variance in the defect scores (0.10%), but instead 99.9% of the variance is not accounted.

Final Analysis

The correlation coefficient obtained for Fan-Out showed a positive association ($r = 0.4773$) with the number of defects per module. However, the strength of the association should be classified as weak for this particular project. The correlation coefficient for Fan-In was much worse than Fan-Out. It should be classified that a minimal association ($r = 0.033$) can be found between the Fan-In and defect values for this particular project.

The results were a little unexpected due to the problems that the Agency has had with this system. The results from the coefficient of determination and nondetermination show there are other variables that need to be included to reduce the percentage of unaccounted variance in the defect values. Including other types of variables may show a strong positive relationship between Fan-In, Fan-Out and defects.

Many different variables could account for the variance of the scores. Measuring the Source Lines of Code (SLOC) for each module would give a better understanding concerning module complexity. Typically, a module that has a higher value of SLOC is more complex when compared to another module having a lower value of SLOC. Other variables that would account for the variance of the scores include:

1. Number and type of module data structures.
2. Number of function points for each module.
3. Number and type of database access for each module.
4. Number of global variables used by each module.

The results obtained from the coupling and complexity breakdown are better understood when it is taken into account that other variables affect the defects found in the system. When looking at the system average for both Fan-In and Fan-Out it shows that most modules are not highly coupled or complex. Three distinct areas were considered highly complex. These areas are PTCG, PTCM and PTGN. However, both PTCG and PTGN contain less than six modules in those distinct areas while PTCM contains 29 modules. Looking at the results in this fashion, one should be able to conclude that PTCM is more complex than PTCG and PTGN.

CHAPTER 9: Summary

This project investigated the high-level design of an Oracle application. Analysis on the coupling of modules was performed and compared against defects found in the application. The Oracle application being investigated was a large-scale, commercial, Graphical User Interface (GUI) based software system.

By looking at the high-level design and the amount of defects within the system, one can conclude that the system is complex. Increased complexity will usually lead to an increase in maintenance tasks. The Agency is currently experiencing this problem in bring the system up to a stable level. However, parts of the high-level design failed to shown any direct association between coupling, complexities and defects.

As stated previously, many factors can affect the amount of defects found in a system. It is apparent that to thoroughly associate the complexity of the system with defects, then a more detailed design must be completed. This would allow the items mentioned in the previous Chapter to be measured. More, importantly it would allow the full nature of the system (database) to be encompassed in forming associations with the defect count.

The Fan-In and Fan-Out counts discovered a few areas that may be highly coupled and complex. However, there was no clear association between these counts and defects. The modules analyzed in this system account for 1,851 defects. A system with this many defects tends to be highly coupled and complex. A more detailed design at the module level and accounting for other factors should adequately show that the system is highly coupled and complex.

Finally, the importance of development control during the software process cannot be underestimated. Outsourcing allows some development control to be given up. Both the Agency and the Development Company, failed to assert measures that allow for some control and monitoring of the development process. This lack of control allowed the 3rd party company to follow unacceptable software development practices. This was evident in the testing results of the system, lack of any delivered design, Agency code reviews and finally the amount of defects found in the delivered system.

APPENDIX A: Fan-Out, Fan-In & Defects

Table 4: Fan-Out, Fan-In & Defects

Module	Fan-In	Fan-Out	Defects
PTAC001F	0	2	55
PTAC001R	0	0	3
PTAC002F	13	7	60
PTAC003F	7	2	60
PTAC004F	0	2	25
PTAC005F	0	2	52
PTAC005R	0	0	6
PTAC005R_SD	0	0	0
PTAC006F	0	2	10
PTAC006R_SD	0	0	0
PTAC007F	28	1	13
PTAC008F	0	1	15
PTAC009F	0	1	9
PTAC010F	0	2	31
PTAC011F	0	4	10
PTAC012F	0	2	6
PTAC013F	0	1	6
PTAC014F	0	1	4
PTAC017F	0	1	2
PTAC018F	0	1	8
PTAC019F	0	1	7
PTAC020F	0	1	11
PTAC021F	0	1	22
PTAC022F	0	1	36
PTAC022F_NEW	0	0	0
PTAC023F	0	2	15
PTAC024F	12	1	5
PTAC025F	0	1	2
PTAC026F	15	3	9
PTAC027F	50	1	3
PTAC028F	0	1	41
PTAC029F	0	1	18
PTAC031F	6	0	4
PTAC032F	0	1	2
PTAC033F	6	1	0
PTAC034F	6	2	1
PTAC035F	2	1	0
PTAC041R	0	0	3
PTAC101L	85	1	0
PTAT001F	0	0	0
PTAT002F	0	0	1
PTCM001F	2	1	7
PTCM002F	23	7	36
PTCM003F	41	12	69

Table 4: Fan-Out, Fan-In & Defects

Module	Fan-In	Fan-Out	Defects
PTCM004F	23	1	9
PTCM005F	73	3	1
PTCM006F	2	2	17
PTCM007F	21	2	21
PTCM010F	58	2	3
PTCM012F	0	2	1
PTCM013F	0	1	1
PTCM016F	27	2	31
PTCM017F	31	4	18
PTCM018F	50	1	19
PTCM019F	0	4	32
PTCM020F	1	0	5
PTCM021F	32	8	7
PTCM022F	48	10	78
PTCM025F	31	7	0
PTCM029F	31	8	9
PTCM032F	25	2	0
PTCM035F	0	0	0
PTCM036F	0	0	0
PTCM037F	0	0	0
PTCM038F	0	2	0
PTCM061F	0	0	0
PTCM062F	0	0	0
PTCM067F	16	1	0
PTCM101L	189	1	0
PTCM102L	71	1	0
PTER001F	1	2	2
PTER002F	3	1	13
PTER003F	0	0	4
PTER004F	0	4	13
PTER005F	0	0	0
PTER007F	12	2	11
PTER008F	0	1	4
PTER010F	0	2	4
PTER011F	9	2	3
PTER012F	0	1	10
PTER013F	0	1	3
PTER014F	0	1	1
PTER035R	0	0	2
PTER101L	25	1	0
PTFC001F	1	3	6
PTFC002F	2	0	3
PTFC003F	0	0	6
PTFC005F	2	0	1

Table 4: Fan-Out, Fan-In & Defects (continued)

Module	Fan-In	Fan-Out	Defects
PTFC006F	9	1	149
PTFC007F	11	2	29
PTFC008F	0	1	3
PTFC009F	0	0	0
PTFC010F	0	0	0
PTFC011F	0	0	0
PTFC012F	0	0	0
PTFC101L	19	0	0
PTFC102L	0	1	0
PTNP001F	0	0	20
PTOG001F	0	0	6
PTOG002F	0	0	0
PTOG003F	0	0	1
PTOG004F	0	0	1
PTOG005F	0	0	1
PTOG006F	0	0	1
PTOG007F	0	0	3
PTOG008F	0	0	0
PTOG009F	0	0	3
PTOG010F	0	0	0
PTOG011F	0	0	1
PTOG012F	0	0	0
PTOG013F	0	0	0
PTOG014F	0	0	2
PTOG015F	0	0	2
PTOG016F	0	0	0
PTOG017F	0	0	0
PTOG018F	0	0	0
PTOG019F	0	0	1
PTOG020F	0	0	2
PTOG021F	0	0	7
PTOG022F	0	0	2
PTOG023F	0	0	6
PTOG024F	0	0	2
PTOG025F	0	0	3
PTOG026F	0	0	1
PTOG027F	0	0	2
PTOG028F	0	0	1
PTOG029F	0	0	2
PTOG030F	0	0	4
PTOG031F	0	0	4
PTOG032F	0	0	10
PTOG044F	0	0	1
PTOG045F	0	0	1

Table 4: Fan-Out, Fan-In & Defects (continued)

Module	Fan-In	Fan-Out	Defects
PTOG046F	0	0	1
PTOG099F	0	0	1
PTOG100F	0	0	1
PTOG138F	0	0	0
PTOG139F	0	0	0
PTOG140F	0	0	2
PTOG500F	0	0	2
PTOG600F	0	0	0
PTRG001F	0	3	35
PTRG001R	0	0	1
PTRG002F	6	10	22
PTRG002R	0	0	4
PTRG003F	5	0	1
PTRG003R	0	0	0
PTRG004F	8	0	4
PTRG004R	0	0	0
PTRG005F	33	0	28
PTRG006F	8	0	18
PTRG008F	9	1	5
PTRG009F	2	7	23
PTRG010B	0	0	3
PTRG010F	32	1	0
PTRG011B	0	0	0
PTRG011F	5	0	0
PTRG012F	12	0	1
PTRG014F	0	0	8
PTRG015F	16		0
PTRP001F	0	2	18
PTRP002F	2	4	123
PTRP003F	6	1	9
PTRP004F	2	5	37
PTRP005F	0	1	25
PTRP006F	0	1	19
PTRP007F	0	0	3
PTRP008F	0	1	5
PTRP009F	5	0	30
PTRP010F	0	4	47
PTRP011F	7	0	18
PTRP012F	7	0	8
PTRP013F	0	0	6
PTRP016F	0	0	0
PTRP021R	0	0	4
PTRP028F	0	0	0
PTSC003F	0	1	2

Table 4: Fan-Out, Fan-In & Defects (continued)

Module	Fan-In	Fan-Out	Defects
PTSC004F	0	1	4
PTSC005F	0	0	0
PTSM001F	0	0	2
PTSM002F	0	1	1
PTSM003F	0	1	2
PTSM004F	1	1	0
PTSM005F	0	1	0
PTSM007F	0	1	5
PTSM011F	0	1	0
PTSM012F	0	0	0
PTSM013F	0	0	0
PTSM014F	0	0	1
PTSM016F	0	0	1
PTSM017F	0	0	0
PTSM018F	0	0	1
PTSM019F	0	0	0
PTSM020F	0	0	2
PTSM021F	0	0	8
PTSM023F	0	0	1
PTSM024F	0	0	0
PTSM034F	0	0	1
PTTK001F	0	1	23
PTTK002F	0	0	9
ACCOUNTING	1	0	0
CM_MENU	1	6	0
PTCGLIB1	25	1	0
PTDIS01F	0	0	0
PTDIS02F	0	0	0
PTDIS03F	0	0	0
PTDS003F	0	0	0
PTGN001F	0	0	0
PTGN002M	0	1	0
PTGN003F	0	0	0
PTGN004F	0	0	0
PTGN081L	226	0	0
PTPASSWD	0	0	0
PTSF002F	0	0	0

BIBLIOGRAPHY

[1] Oracle Database at DESY, http://www.desy.de/asg/oracle/homepage.html

[2] Oracle Corporation, Oracle Designer 6i Overview: An Oracle Technical White Paper, October 2000

[3] Oracle Corporation, Building Oracle Forms Applications Using Designer 6i: An Oracle Technical White Paper, October 2000

[4] Pressman R, Software Engineering- A Practitioner's Approach, Fifth Edition, McGraw-Hill Higher Education, 2001

[5] Microsoft Visio 2000 Help Documentation

[6] McConnell S, Code Complete: A Practical Handbook of Software Construction, Microsoft Press, 1993

[7] Kirk R, Statistics: An Introduction, Third Edition, The Dryden Press, 1990