POLITECNICO DI MILANO

Corso di Laurea specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

An environment of continuous integration & software metrics for a DB programming language

Relatore: Prof. Giuseppe Pozzi

Corelatore: Doct. Massimo Rosin

Tesi di Laurea di:

Xu Shaojie, matricola 73808

2010/2011

Ringraziamenti

All'inizio del presente lavoro desidero ringraziare ed esprimere la mia riconoscenza nei confronti di tutte le persone che, in modi diversi, mi sono state vicine ed hanno permesso e incoraggiato sia i miei studi che la realizzazione e stesura di questa tesi.

Desidero innanzitutto ringraziare il Professore Giuseppe Pozzi per la continua disponibilità e prontezza durante tutte le fasi del mio lavoro ed avermi prontamente aiutato nei momenti di difficoltà

Ringrazio anche il dottor Massimo Rosin per aver contribuito alla mia formazione e crescita nel campo professionale. Non scorderò la sua continua disponibilità e pazienza che mi ha dedicato durante l'intera esperienza di stage nella azienda Reply Technology, gli incoraggiamenti, i consigli e al tempo stesso gli apprezzamenti per quanto realizzato. Senza i suoi aiuti, il mio lavoro non avrebbe potuto raggiungere il livello che ha raggiunto.

Inoltre ringrazio sinceramente il miei colleghi Matteo Galli e Mauro Lucchini per avermi fornito materiale utile per la realizzazione della mia tesi e per essere sempre stati disponibili a dare consigli preziosi e dirimere i miei dubbi durante la stesura di questo lavoro.

Ringrazio anche l'azienda Reply S.p.A per avermi offerto la possibilità di svolgere uno stage su nuove tematiche, ambienti e strumenti all'avanguardia.

Rivolgo un ringraziamento speciale ai miei genitori, per avermi dato la possibilità di studiare, per aver sostenuto le mie scelte e per i consigli importanti che mi hanno sempre dato.

Abstract

Continuous integration (CI), a term used as one of the practices of XP (Extreme Programming), has been widely recognized and employed by many software companies. There is no denying there are benefits resulting from the introduction of tools and practices of continuous integration for the software development. Nowadays, an enterprise which carries on any activity associated with code generation does not use repository is rare. Software metrics, those qualitative and quantitative measurements on purpose of improving the software product and the process has also been applied for year.

Software metrics, which is the measurement of the software product quality and process, has been becoming increasingly important to software development. It emphasize that how good the product is designed (quality of design) and how well the software product conforms to that specification. To have more accurate schedule and cost estimates, better quality products, and higher productivity. It is highly advised to employ software metric to the development process as well as the product.

Given that software development is becoming more and more extremely complex, especially for Oracle PL/SQL, in the current software market, there are few well-defined, reliable measures of either the process or the product to guide and evaluate development. Furthermore, only employing a simple way of monitoring the software development process or product is far away from satisfaction. Setting up a multi-environment of checking, building, controlling and monitoring for a software development is going to be a tendency.

My thesis investigates on the practice of continuous integration, which is a technique of agile software development methods, and its impact on software quality in terms of both developing process and the product. Meanwhile, to ensure a good analysis and evolution of software quality, it's inevitable to investigate the software metrics, which deals with the measurement of the software product and the process by which it is developed. At last, a real case will be presented. It is an implementation of two plug-ins both in the server side (SONAR) and client site (Eclipse) aiming to do the analysis of Oracle database programming language PL/SQL.

Sommario

La presente tesi riguarda la pratica della integrazione continua(continuous integration), che è una tecnica di sviluppo software che impatta sensibilmente sulla qualità del software sia in termini di processo di sviluppo che di prodotto finale.

Per garantire una buona analisi ed il conseguente sviluppo del software, è necessario utilizzare alcune metriche del software, con l'obiettivo di misurare il prodotto software ed il processo attraverso il quale esso si sviluppa.

La tesi considera inoltre un caso reale riguardante la realizzazione di due plug-in: il primo e' lato server (SONAR), il secondo e' lato client (Eclipse), con l'obiettivo di effettuare l'analisi dei comandi nel linguaggio di programmazione PL / SQL del DBMS Oracle.

Table of Content

Ringraziamenti 2

Abstract 3

Sommario 4

List of Figures 9

List of Tables 13

1. Introduction 14

1.1 The Topic 14

1.2 Why is it relevant 14

1.2.1 The need of continuous integration 15

1.2.2 The need of software metrics 19

1.2.3 Customer requirement analysis 20

1.3 Paper outline 21

2. Continuous Integration 23

2.1 Introduction 23

2.2 Typical practices of CI 24

2.3 CI in classical software process 27

2.3.1 "Traditional" process problems 27

2.3.2 Problems solved through continuous integration 28

2.4 Continuous Integration in the real world 28

2.5 Chapter Summary 30

3. Software Quality Metrics ... 31

3.1 Introduction ... 31

3.2 Software quality ... 31

3.2.1 Software quality definitions ... 32

3.2.2 Quality classifications ... 34

3.2.3 Quality models ... 35

3.2.4 Quality factors ... 38

3.3 Software metrics ... 41

3.3.1 Software metrics definition ... 42

3.3.2 Metrics classification ... 42

3.4 Software metrics selected for the PL/SQL ... 44

3.4.1 Product Metrics ... 44

3.4.2 Metrics chosen for PL/SQL ... 50

3.5 Software quality control tools ... 53

3.6 Chapter summary ... 56

4. Metrics Plug-in design and implementation on the Client-side: Eclipse ... 57

4.1 Eclipse Plug-in Infrastructure ... 57

4.1.1 Eclipse platform architecture ... 57

4.1.2 Plug-in Structure ... 59

4.2 PL/SQL plug-in development ... 64

4.2.1 Plug-in interface implementation with SWT/JFace ... 64

4.2.2 Plug-in logic implementation ... 68

4.3 Chapter Summary 72

5. Metrics Plug-in design and implementation on the Server-side: Sonar 73

5.1 Introduction of SonarSource 73

5.2 The structure of SONAR 75

5.3 Maven 77

5.3.1 Maven Introduction 77

5.3.2 Maven POM.xml 78

5.3.3 Using of Maven 80

5.3.4 Using of Sonar based on Maven 83

5.4 Sonar Web Application & ROR 84

5.4.1 Sonar Plug-in 84

5.4.2 ROR (Ruby on Rails): the future of web application framework 87

5.4.3 MVC Structure: the web design pattern 90

5.5 Sonar Plug-in for programming language Pl/SQL 93

5.5.1 Getting started 94

5.5.2 Code the plug-in in Client side (Java) 94

5.5.3 Code the plug-in in Server Side (Ruby on Rails) 99

5.5.4 Installation 104

5.6 Chapter Summary 105

6. Experimental Results 106

6.1 Plug-in test results and analysis on Eclipse 106

6.2 Plug-in test results and analysis on Sonar 111

6.3 Chapter summary 116

7. Conclusion 117

7.1 Thesis contribution 117

7.2 Lessons learnt 118

7.3 Future research directions 119

8. Bibliography 120

List of Figures

Figure 2-1 Continuous Integration practice 25

Figure 2-2 Continuous Integration Schemas... 26

Figure 3-1 A control flow graph of a simple program 46

Figure 3-2 Checkstyle Eclipse Plug-in 54

Figure 3-3 Eclipse show view of Checkstyle plug-in....................................... 55

Figure 3-4 ClearSQL 56

Figure 4-1 Eclipse platform architecture 58

Figure 4-2 Eclipse plug-in structures... 59

Figure 4-3 Manifest.mf file 60

Figure 4-4 Plugin.xml 62

Figure 4-5 Two actions of PL/SQL plug-in 63

Figure 4-6 structure and relationship of SWT... 64

Figure 4-7 Two tables to show the analysis result 66

Figure 4-8 Menu Oracle 66

Figure 4-9 Menu File 67

Figure 4-10 Directory dialogue to select a project 68

Figure 4-11 Analysis with default value zeros 71

Figure 5-1 Axes of Sonar 73

Figure 5-2 The homepage of sonar 74

Figure 5-3 The homepage of a specific project 74

Figure 5-4 Drilldown to see the code 76

Figure 5-5 Steps of doing a code analysis in sonar 77

Figure 5-6 Standard directory of maven project 82

Figure 5-7 Maven command to launch an analysis 83

Figure 5-8 Successful Build 84

Figure 5-9 MVC schema 91

Figure 5-10 Schematic views of Ruby and the Rails framework [19] 93

Figure 5-11 Successful creating a Maven project 94

Figure 5-12 Plug-in-Classes 95

Figure 5-13 Dependencies of sonar plug-in 95

Figure 5-14 Oracle Pl/SQL Plug-in 97

Figure 5-15 list of all the extension points.. 98

Figure 5-16 PL/SQL Dashboard Widget... 98

Figure 5-17 Sonar web application standard directory.....................................100

Figure 5-18 Url_plsql_drilldown class 103

Figure 5-19 NCSS measurement details 103

Figure 5-20 Added method plsql_details 104

Figure 6-1 Launch new analyses 106

Figure 6-2 Oracle Project Selections 107

Figure 6-3 Analyses results 107

Figure 6-4 Launch another new analyses...108

Figure 6-5HTML report 109

Figure 6-6 Comparison of two projects 110

Figure 6-7 Launch a Sonar analysis 111

Figure 6-8 Successful analysis 112

Figure 6-9 A project only has PL/SQL 112

Figure 6-10 A project has both JAVA and PL/SQL 113

Figure 6-11 NCSS details..114

Fgiure 6-12 V(G) < 10 details 115

List of Tables

Table 5-1 Project builds lifecycle 81

Table 5-2 Sonar Plug-ins.. 87

Table 5-3 A summary of the default Sonar Rails directory structure 101

1. Introduction

1.1 The Topic

Software Quality activities are conducted throughout the project life cycle to provide objective insight into the maturity and quality of the software processes and associated work products.

Up to now, there are plenty amount of solutions for the assurance of the software quality. What we are going to address is the software quality improvement and assurance through the CI (continuous integration) as well as the specific software metrics investigation.

1.2 Why is it relevant

The fundamental nature of software development is changing dramatically, and quality professionals must change with it. Therefore, modern software processes have changed to classical and traditional models into processes which are evolutionary, iterative, and incremental. And continuous integration (CI) as one solution of agile way of software development, implements continuous processes of applying quality control - small pieces of effort, applied frequently. Improving the software quality, reducing the developing time is the main purpose and practice of continuous integration. It replaces the traditional practice of applying quality control after completing all development.

Moreover, to obtain a better control of software quality both in the development process and the work product, we need to have several software source oriented metrics to measure the software or the source code quantitatively. As for our project, the target software source is the Oracle database programming language – PL/SQL.

The goal of both Continuous Integration and software metrics are both to

improve the quality of software produce and its corresponding process. Nevertheless, they have distinctive behavior of improving software quality. By combining them together properly, it is possible to obtain a better performance than only using them separately. For Continuous Integration, it increases the quality of software through the assurance of the code being free of bugs and buildable by frequently, automatically, continuously checking every change of code submitted into repository. As long as there is any error or break of building, all the tasks need to stop and the priority is to fix that problem before any proceed. Instead, for software metrics, or to be specific in our project, software code metrics, it guarantees the quality of the software by analyzing the code at times and let the team members be aware of the status of current software development status. For example, in last build, the number of views are 50, and the number of cyclomatic complexity V(G) which are larger than 10 (research reveals that a piece of code with cyclomatic complexity greater than 10 is not easily testable, and so it is not easily maintainable than that less than 10, it is highly suggested to limit program with cyclomatic complexity less than 10, the lower the better) is 40. And those greater than 10 is zero which is good status. However, after next building, the number of views are 60 (program increased) and the number of program with cyclomatic complexity becomes 3 instead of 0 which is not advisable. Actions need to be taken to fix the 3 pieces of code so as to make their value of V(G) less than 10. Thus the good testability and maintainability are guaranteed. And the details about this will be explained in the following of my thesis.

Next, let us have an inside view of the importance of applying continuous integration and software metrics in modern software development, and the meaningfulness of employing both of them. After that I will present why it is necessary to develop our project based on the analysis of customer requirements.

1.2.1 The need of continuous integration

Normally, improving software quality, increasing productivity and reducing risk are the purposes of using Continuous Integration (CI) as a best development practice for any organization that produces software.

Nowadays, Continuous integration has been heralded as a best practice of software development. It is widely acknowledged that software built frequently with the automated regression test suits run after each build, results in higher developer productivity and shorter release cycles. As we all know that the earlier we detect the bugs, the less cost to fix them. Normally, if a bug is detected in the end, or just before the releasing period, huge amount of resource and capital would be wasted. It is really quite easy to see why this is generally true: building your software and running your unit test libraries shortly after code changes are committed to the SCM repository will allow your developers to produce higher quality code quicker as they will be able to quickly fix any integration build errors or software defects they introduced shortly after committing their changes while the changes are still fresh in their memory. Every time after the build of the source, if any of the tests fail, all effort is directed at fixing the problem in order to obtain a working version of the system. As Martin Fowler states:"The whole point of Continuous Integration is to provide rapid feedback." It is really much less expensive to troubleshoot and fix a bug within minutes or hours of checking in the code than within days or weeks when lots of other, inter-related, code changes are committed to the repository. Failing builds, of course, are the main impediment to such rapid feedback if they are not fixed timely. Continuous integration has therefore been called the "heartbeat of software". If it stops, you can't ship. [1]

The key of continuous integration is: [41]

- Fully automated integration
  What is integration? Making different modules work together is integration. Of course all the modules have to work together and they must be integrated. And there are two ways of integration, manual integration and automated integration. It is the fully automated integration that makes CI so popular in software development application.
  If it is manual integration, you need to do the build, test, deployment and so forth manually. What's worse is that it is impossible to do all these work in short time, normally a few hours or days which would lead to let check-ins. As we have discussed above, the less you check you code, check the status, the more the bugs would appear in the end. The bugs are delayed to be detected owing to the postponed, time-consuming, non-automated integration. If developers are running build scripts manually (either from their machines, or on a server somewhere), that

doesn't quality as CI. The act of checking-in the code will be a trigger of the CI build. This feature makes the possibility of realization of higher frequency, shorter duration (a few hours or even less) of continuous integration.
The automation feature of the continuous integration is guaranteed by several tools:

- Checkout or update of source code by a version control tool
- Compilation of project source code by a build tool
- Execution of source code checking by a static analysis tool
- Unit test execution by a unit test harness or runner

Automation of integration makes the software development task in a way that is deterministic and repeatable so that it can be executed during process. The purpose is to minimize the interaction of human so as to remove the human error and have the tasks being run in background.

- Compilation of latest source code
  To emphasize again: The earlier you can detect problems, the easier it is to resolve them. As long as the code is being checked in, we have to ensure that the code is run able, functional, and free of bugs and complying with the specification.

- From source control
  Typically developers download source code from the SCM repository into their local work area and do their coding using an IDE tool such as Eclipse or VS.NET. Let's assume that there are 10 developers doing the development in parallel and there is no rule of committing the code into SCM repository. They build and unit test their changes in the local work areas. It is possible to have the situation that the committed code might conflict with the code that has already committed to the repository. Under this situation, it would be the best practice for the developer to resolve the conflicts in the local work area build and test the changes and then synchronize the code with the repository if and only if the changes cause no merger conflict. [42]

As we mentioned at the beginning of this section, the three purpose of continuous integration is to improve quality, increase productivity, and reduce risk.

It is obvious that the quality of software product and its corresponding process would be improved by employing CI from what we have stated above. All the practices are all in aim of improving the quality. But what's the connection of CI with the higher productivity? Is there any direct impact of CI on the software development productivity? I say yes.

For one thing, using a fully automated build process, definitely, should have a rather positive effect on the increasing of productivity. With a fully automation of integration, developer are spending less time doing the buildings. More efforts are exerted to the generation of code to fulfill the functions. For another, by strictly employing CI, we have the advantage that all the code in the SCM repository is buildable and free of bugs. There is no more situation of full of bugs before the releasing period. The release lifecycle is reduced. Thirdly, the satisfaction and confidence of being successfully is obtained to the team by executing the successful building each time thanks to the CI. So team members are more motivated to work on that project. This is an indirect of positive impact on the productivity. Last but not least, checking a project with the team members spread all over the world with different time zone would be a big problem. Unfortunately, this case is quite normal currently like a team member in the US checked in the latest code on Friday nightly, shortly before leaving work, and fixed all the bugs. On Sunday, another team member in China check out the code and resume to work based on the code of the team member from USA.

Any time a developer checks in their code without properly checking whether the committed code is buildable and free of bugs. That would be a huge problem for continuous integration. It might probability break the development process. Assuming what the Chinese developer in the morning found that the code committed the day before by the USA team member is unbuildable and full of bugs? For him, either fixes the problems by himself or wake up the team in the USA. Both of them can be a disaster to the whole team.

Regarding reducing risk, a few reasons stated above can be applied here either. Fully automation of integration is aiming at reducing or eliminating the human error during the building process. After each CI process, if there is any problem occurs, the responsible person would be noticed. This person would stop the work at hand and turn to fix the problem before any advance

which ensures the whole project is on-track.

Taking all the facts above into consideration, we are naturally going the conclusion that by using the CI, a team would have more time on producing good, commercial-quality code and spend less time manually running low-value tasks. A team is able to achieve value to the business market. All of that clearly and deeply explains the CI important.

1.2.2 The need of software metrics

Software metrics are measures of the attributes of software products and processes. They are increasingly playing a central role in the planning and control of software development projects.

Software Metrics can be divided into product metrics and process metrics. Product metrics measure the software at a certain development phase. Product metrics may measure the complexity, the size of the final program (either source or object code), or the number of pages of documentation produced. Process metrics, are measures of the software development process, such as overall development time, type of methodology used, or the average level of experience of the programming staff. [5]

But why on earth do we need the software metrics to measure the software process and product?

There is one sentence saying: “Developing software is more of a business or a process than an art form… a business or process needs to be managed through the use of various control functions.” There is another saying: “The key to successful risk management is in the ability to measure. In order to be successful a rigorous and well-thought path to managing these [risk] issues must be continuously developed. At the heart of it all is the notion of having key business measures. Industry gurus have told us for years that we need to measure what we manage.” These are some key points I took from the book measuring the Software Process: A practical guide to functional measurements by David Garmus and David Herron.

There is no denying that project productivity is increased as the quality increase. In order to increase quality and productivity, weaknesses must be identified in the methods being currently used and actions taken to strengthen these areas of software development process. So, if we really want to know whether or not we are improving our effectiveness and efficiency then we should really spend some time understanding why measures are important, and define critical software metrics. And it is the exact content will be covered in my thesis.

Given that software development is extremely complex, especially for Oracle PL/SQL, in the current software market, there are few well-defined, reliable measures of either the process or the product to guide and evaluate development. Thus, accurate and effective estimating, planning, and control are nearly impossible to achieve. This is the goal of software metrics-the identification and measurement of the essential parameters that affect software development. [5]

1.2.3 Customer requirement analysis

As an Oracle partner, Software Factory Reply Technology is especially committed in working with Oracle products and tools. Among these, the database is probably the most used and stressed: near all applications need at least one database where to store data and vital information. The database is also a development environment: Oracle RDBMS, just as other databases, provides a specific language for scripting development, the Pl/SQL language. Developers can write Pl/SQL code to manipulate data, often with higher performance and ease with respect to Java or other development environments. In this sense, Pl/SQL code is an important part of the source code for the average software project in our Factory, just as well as Java.

Thus, it becomes of the utmost importance to guarantee at least the same quality we can obtain for the Java part of our work.

Sonar and Checkstyle were the leading tools for Java code review and software metrics at the time the Factory needed specific tools for such tasks, but nothing seemed to satisfy the same needs for Pl/SQL. Quest Software's

TOAD had a module, called CodeXpert, which was able to perform some measurement about complexity on Pl/SQL code, but had some flaws: first release was somehow buggy, reports were giving results only if the code was showing measures over a certain threshold (not customizable) and, ultimately, it was a client-side tool, with expensive license costs.

Therefore, having no other alternative, we decided to develop a Pl/SQL software metrics tool of our own. The tool was written in Java and was integrated in a pluggable architecture which also hosted a Checkstyle implementation, in order to collect measures for projects based on both Java and Pl/SQL.

As soon as Sonar becomes a mature platform, we switched from Checkstyle to Sonar, but since Sonar had no Pl/SQL module, we did not abandon our software metrics tool. Also, we used the same tool at some Customers' sites, in order to provide quality measurements on applications under maintenance, both ours and third-parties, so we needed to continue supporting it.

Quest Software today offers a Sonar commercial plug-in for Pl/SQL metrics. Such plug-in is not free of charge and, moreover, it requires TOAD as server-side metrics engine, thus requiring the use of a TOAD instance specifically for this task, and one additional TOAD license to be purchased. That's why we continued the effort of development on our tool, and we decided to provide plug-ins for Sonar (server side) and Eclipse (client side), which are core parts in software life cycle within our Factory.

1.3 Paper outline

The paper will be organized as follows:

Chapter 2 discusses the Continuous Integration as well as the agile software development and software life cycle. After that Continuous Integration practices in real world will be explained.

In chapter 3, software quality and its models and classification will be

introduced firstly. And then software code metrics will be covered generally, the definition of various software metrics and its application to improve the quality. At last I will explain the selected software metrics applied to our project in real practice in our project.

In chapter 4 we focus on explaining the design and implementation of both the PL/SQL core analyzer and the plug-in to be installed in Eclipse. The PL/SQL analyzer is the core analyzer of both for Eclipse and Sonar. It is mostly developed by Reply Technology senior consultant Massimo Rosin and I will give a brief introduction on it. After that I will explain in details about the Eclipse structure especially the architecture of Eclipse plug-in based on which Eclipse to be extended. To develop the plug-in of Eclipse, SWT/JFace is very important and it will be explained as well. The related work done by us will be presented in the end of this chapter.

In chapter 5 the server side Sonar plug-in will be shown. Firstly, the Sonar itself will be introduced briefly as well as Maven, based on which Sonar software code analysis is run. Secondly, Sonar plug-in research and development will be discussed. How the plug-in is organized and how to develop the plug-in from the Eclipse side to the server service side? I will explain these in detail. To develop the plug-in in the server service side, it is essential to have a good command of ROR (Ruby on Rails), which is deem as the future of agile web application development.

Chapter 6 you will see the tests and experimental results, and then analysis will be done based on the result.

Chapter 7 summarizes the contribution of this thesis, the lesson learnt, and outlines of the further research directions.

2. Continuous Integration

2.1 Introduction

Continuous Integration is emerged in the Extreme Programming (XP) community, and XP advocates Martin Fowler and Kent Beck first wrote about continuous integration around 1999. [25]

But what is exactly Continuous Integration? What is the difference between the other XP practices? If in a software company, members or developers of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day, and we can say that this company is using continuous integration, one the best practice of XP programming.

Every time, when the developer submits the code into the repository, that part of code would be verified by an automated build (including test) to detect integration errors as quickly as possible. The results of the build and testing are reported to the entire team. This tightens team-wide communication about the quality of the committed changes. Continuous integration leads to significant reduction of integration problems and enable to speed up the software development cycle.

The goal of continuous integration is to know the global effects of local changes as soon as possible. Integration bugs are hard to track down because they originate from the interaction between (changes to) different subsystems, so they are hard to test for on a subsystem level. Postponing integration makes things even worse: the interaction between changes increases very fast making integration bugs exponentially harder to find. It is therefore important that integration builds are executed quickly enough. [2]

In multiple ways, an agile developing team would benefit from such a system. For one thing, the system can provide an immediate feedback to all parties shortly after a problem is occurred. For another, valuable deployment artifacts would be created and different audiences would be reported within

the agile team.

It has many advantages: [25]

- When unit tests fail or a bug emerges, developers might revert the codebase back to a bug-free state, without wasting time debugging
- Developers detect and fix integration problems continuously - avoiding last-minute chaos at release dates, (when everyone tries to check in their slightly incompatible versions).
- early warning of broken/incompatible code
- early warning of conflicting changes
- immediate unit testing of all changes constant availability of a "current" build for testing, demo, or release purposes
- immediate feedback to developers on the quality, functionality, or system-wide impact of code they are writing
- frequent code check-in pushes developers to create modular, less complex code
- Metrics generated from automated testing and CI (such as metrics for code coverage, code complexity, and features complete) focus developers on developing functional, quality code, and help develop momentum in a team.

2.2 Typical practices of CI

A typical CI (figure 2.1) practice starts with the compilation of the application. If there is any problem, which will fail the CI cycle and the system; the whole team would be notified and to fix that. The CI server will recompile and go on executing as long as the problem is fixed. After this

stage, the CI system is ensured that the database will be up to date. Nevertheless, the CI cycle would not be failed. It depends on the server configuration unit test. And then in the next two steps the CI server will package and deploy the application. After all functional tests successfully pass; the CI system will generate a report of the integration cycle and notify them to a certain set of developers.

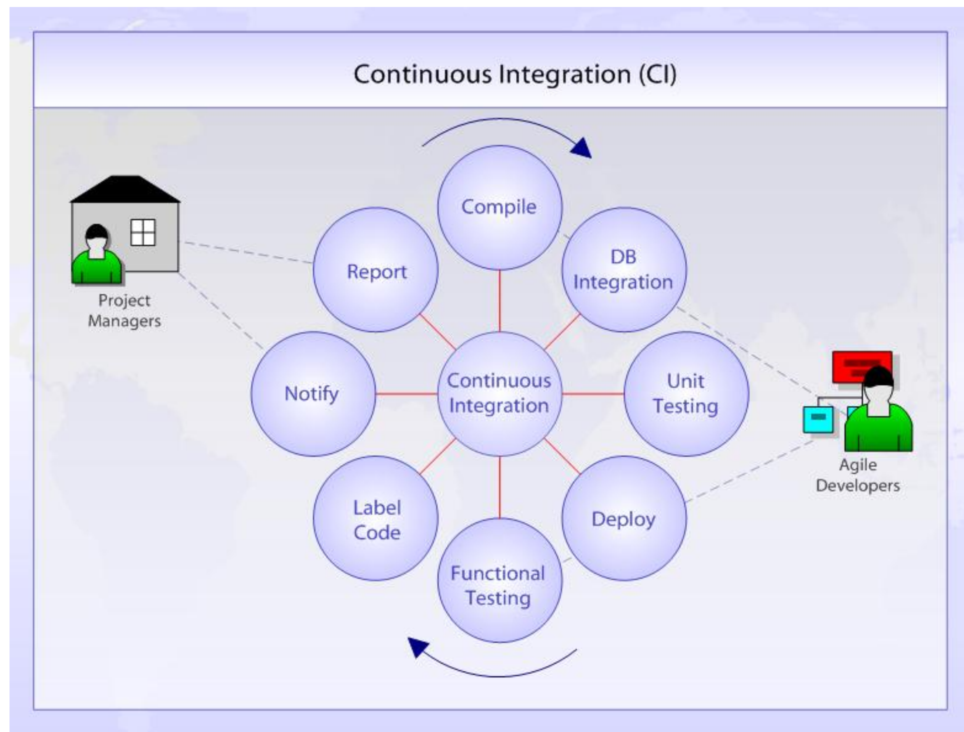


Figure 2-1 Continuous Integration practice

To make each CI stage more clear, we list the following steps which are typical in a CI system:

1. Building the software on the local PC and then runs unit tests.

2. Checking the source into a repository as long as the submitted code passes all the unit tests.

3. CI Server detects the changes in the repository and gets a copy of the code

to its local system.

4. The CI Server builds and runs unit tests.

5. The results of the build and unit tests are automatically posted to a web site, such as the Hudson web server, from where all team members can be informed the current state of the software.

A classic workflow of a CI system is as the following figure shows:

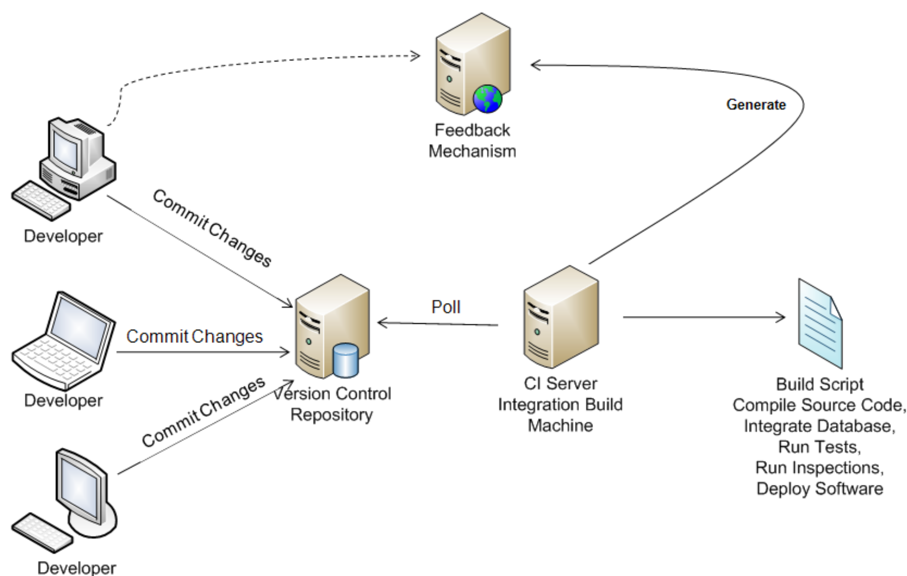


Figure 2-2 Continuous Integration Schemas

Apart from the above minimum steps that you do under your CI system, other things that can be done include:

- Produce developer documentation of the system. This can be in the form of help files, UML diagrams, and more.
- Run code metrics on the source that indicate code coverage, adherence to standards, amount of duplicated code, cyclomatic complexity, NCSS and

so forth.

- Produce an install set by calling programs such as Install Shield.
- Call external, sophisticated testing applications to do functional testing
- Burn a CD or DVD that contains the release bits of the application

2.3 CI in classical software process

2.3.1 “Traditional” process problems

In the traditional software development there exist problems which would be solved by using continuous integration. In waterfall model, there are many strict phases and after each phase is finished. Before moving to the next phase, review is necessary to ensure that the phase is indeed complete; it is like passing through a “gate” from the current phase to the next. Once finished, it is highly unadvisable to revisit or rectify the previous. This kind of "inflexibility" has been a source of criticism by other more "flexible" models. [26]

To make a summary of discussed above, it has the following disadvantages: [27]

- Testing may not be done efficiently
- Integration is long and difficult
- Poor visibility on development progress
- Functional tests are done too late
- Raised issues are harder to fix
- the client gets a sub-optimal product

2.3.2 Problems solved through continuous integration

By automatically "integrating" and compiling source code from different developers on a central build server, researches have proven that the corresponding problems can be solved: [27]

- Smoother integration
- Automatic regression testing
- Regular working releases
- Earlier functional testing
- Faster and easier bug fixes
- Better visibility

2.4 Continuous Integration in the real world

As have been discussed above, in real business cases continuous integration has several advantages over classical development cycle. In combination with software metrics and automatic unit testing, it helps developers write better code, more readable, more maintainable, and because of more easily tested, it becomes more stable. Continuous integration, by building the software automatically at a high frequency, allows discovering problems very early during the development phase. Developers are pressed to commit source code only in case of stable release: the CI environment immediately reports cases of code not compiling, not installing or test-failing and sends a notification message (e.g. an e-mail) to an Administrator or a Technical Lead.

Typical tasks requested in real-world (e.g. Reply Technology company) CI environments (e.g.: Hudson, CruiseControl) are as follows. They are presented by Massimo Rosin, a senior software consultant and project manager from Reply Technology, which is famous IT Consultancy Company in Italy.

- check-out code from source repository (e.g.: CVS, Subversion)
- compilation
- assemble: artifacts (executables, libraries, etc.) are created, for testing purposes
- automatic unit test: software is tested against unit testing suites (e.g.: JUnit)
- Reporting: an unit test execution report is created.

In parallel, a code review tool takes the source code and/or the artifacts as its input to perform a static and/or dynamic code analysis (software metrics, common vulnerabilities, code coverage, etc.).

As a practice in the Software Factory, they embraced such scenario, and somehow extended it, in order to be more compliant to Customer's needs. Customer is always, and obviously, interested in having stable, eventually bug-free software, correctly installed and integrated in his own runtime environment, while keeping maintenance costs as low as possible.

This ideal-world picture declines in a real-world, Hudson-based CI environment in which the main steps are the following:

- check-out code from a Subversion source repository located either in-house or on the Customer's side or in a third-party repository or a combination of these
- compile the code: our software is Java and Pl/SQL; the two pieces of code related to the different languages are treated separately; most of the times, Java is "master" and Pl/SQL is "slave", in the sense that Java uses Pl/SQL and not vice-versa, so that by starting all CI tasks from Java we cover 99% of the work
- assemble: executables and libraries are created, most often they are EAR

o WAR files aimed to implement web applications and/or web services as well

- automatic unit test: JUnit is used in our case
- reporting: test execution report is created and, in case of successful building, a release note document is produced, showing the list of changes brought in the software since last release
- deploy: software packages are installed in a test runtime environment which emulates, as best as possible, Customer's runtime environment; in this sense, stubs may be provided to simulate third-party systems or endpoints for web services
- Software metrics: a synthetic static-analysis report is generated through a custom, Checkstyle-based metrics tool; Sonar is used as well in order to have more details, but our synthetic report is yet important because we can provide it to the Customer, after having shared with him some threshold values for critical metrics, such as complexity.

2.5 Chapter Summary

This chapter mainly describes what continuous integration is, the benefits of using continuous integration in modern software developing process, especially focused on the application of continuous integration in classical development model, as well as the typical steps of adopting CI. At last we present its practices in a real software consultancy company.

To put the continuous integration in a simple way, a build (no matter it is successful or not) is always better than no build. If there are changes to a component, the system should find a way to integrate them, even if builds of certain dependencies may have failed. It involves integrating early and often, so as to avoid the pitfalls of "integration hell". The practice aims to reduce timely rework and thus reduce cost and time.

3. Software Quality Metrics

3.1 Introduction

Software metric is a way of measuring some properties of software or its specification. There is one saying goes: You can’t control what you can't measure. Researchers have proven that metrics have made a big difference in modern software development and is becoming an integral part of the software development process. The difficulty is in determining which metrics matter, and what they mean.

In my dissertation, I will present the collection, analysis and applying of software metrics so as to improve the software development process and the product.

To have an effective management of software development process, it requires quantification and measurement. And software metrics provide a quantitative basis for the development and validation of models of the software development process. It can be used to improve software productivity and quality. [5]

Software Metrics can be divided into product metrics pr process metrics. Product metrics measure the software at a certain development phase. Product metrics may measure the complexity, the size of the final program (either source or object code), or the number of pages of documentation produced.

Process metrics, are measures of the software development process, such as overall development time, type of methodology used, or the average level of experience of the programming staff. [5]

3.2 Software quality

Software quality is a measurement of the software product and process. It

emphasize that how good the product is designed (quality of design) and how well the software product conforms to that specification (quality of conformance), although there are several different definitions. It is often described as the 'fitness for purpose' of a piece of software.

3.2.1 Software quality definitions

There are many different definitions proposed by international organizations or by computer professionals:

- "Quality comprises all characteristics and significant features of a product or an activity which relate to the satisfying of given requirements".

German Industry Standard DIN 55350 Part 11

- "Quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs".

ANSI Standard (ANSI/ASQC A3/1978)

- The totality of features and characteristics of a software product that bear on its ability to satisfy given needs: for example, conform to specifications.
- The degree to which software possesses a desired combination of attributes.
- The degree to which a customer or user perceives that software meets his or her composite expectations.
- The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer".

IEEE Standard (IEEE Std 729-1983)

- External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not.

Steve McConnell's Code Complete

- Conformance to requirements

Crosby, 1984, p80

- The composite product characteristics of engineering and manufacture determine the degree to which the product in use will meet the expectations of the customer

Fergenbaum, 1981, p13

- Quality is product performance, quality is freedom from defects, quality is fitness for use

Juran, 1988, p11

- Meeting customer's requirements

Oakland, 1993, p4

- Zero defects

Shigo, 1986, p11

- Product quality is determined by the economics loss imposed upon society from the time a product is released for shipment

Taguchi,1987

All the definitions above are useful, contributive, constructive, and give us different views of what is software quality. However, neither of them is clear, completed and well defined.

So we need a more accurate and considerate definitions which is proposed by Ince (1994). It describes the modern view of quality:

"A high quality product is one which has associated with it a number of quality factors. These could be described in the requirements specification; they could be cultured, in that they are normally associated with the artifact through familiarity of use and through the shared experience of users; or they

could be quality factors which the developer regards as important but are not considered by the customer and hence not included in the requirements specification".

3.2.2 Quality classifications

Software quality can be divided into product quality and code quality.

1. Software product quality

It measures the conformance to requirements or specifications, there are:

- Reliability
- Scalability
- Correctness
- Completeness
- Absence of bugs
- Fault-tolerance
- Extensibility
- Maintainability
- Documentation

2. Source code quality

- Readability
- Maintainability
- Testability
- Portability
- Complexity

- Low resource consumption: memory, CPU
- Number of compilation or lint warnings
- Robustness

Computer itself does not have the awareness whether the code is well-written or not. So the code quality is discussed from a human point of view. Source code can be written in a way that has an effect on the effort needed to comprehend its behavior. There are many source code programming style guides, which often stress readability and usually language-specific conventions are aimed at reducing the cost of source code maintenance. And in the following part of my dissertation, I will focus on the code quality stated above instead of the software qualities.

3.2.3 Quality models

There are various quality models that have been developed over the years.

One of the oldest and most frequently applied software quality models is that of McCall et al. presented in 1979. McCall's model is used in the United States for very large projects in the military, space and public domain. It was developed in 1976-7 by the US Air force Electronic System Division (ESD), the Rome Air Development Centre (RADC) and General Electric (GE) with the aim of improving the quality of software products. One explicit aim was to make quality measurable. [4]

McCall model stated 55 quality characteristics and it is the first time to refer those software quality characteristics as “factors”. The model organizes quality around three uses of software: product revision, product operations, and product transition. Each of these uses is associated with a set of quality characteristics. Product revision includes maintainability, flexibility, and testability. Product transition includes portability, reusability, and interoperability. Product operations include correctness, reliability, efficiency, integrity, and usability. And it reduced the 55 factors into the following eleven for sake of simplicity:

- McCall et al model
  - Integrity
  - efficiency
  - reliability
  - usability
  - accuracy
  - maintainability
  - testability
  - flexibility
  - interface facility
  - re-usability
  - transferability

Barry Boehm (Boehm, as cited in DeGrace, 1993) presented another similar model that was composed of 19 essential quality attributes. The Boehm model shares a common subset with the McCall model and identifies additional quality attributes.

- Boehm Model
  - Usability
  - clarity
  - efficiency
  - reliability
  - modifiability
  - re-usability
  - modularity
  - documentation
  - resilience

- correctness
- maintainability
- portability
- interoperability
- understandability
- integrity
- validity
- flexibility
- generality
- economy

By using a quality model and the standardized approach, many benefits can be obtained, and let's consider some of the benefits before going into the details of quality factors. [5]

- Validate the completeness of a requirements definition
- Identify software requirements
- Identify software design objectives
- Identify software testing objectives
- Identify user acceptance criteria for a completed software product

A quality model that specifies usable and practical metrics can improve the quality of design models. We can address many of the issues associated with achieving quality after using quality model. It improves communications between acquirers, architects, and developers and results in quality requirements being specified more precisely and more frequently.

3.2.4 Quality factors

Till now we have been talking software quality in general. What it means to be a quality product. We also looked at quality models in brief. We need to know various quality factors upon which quality of software produced is evaluated. These factors are given below.

The various factors, which influence the software, are termed as software factors. To apply software metrics, having a good understanding of quality factors is very crucial. Many metrics are based on the measure of quality factors either quantitatively or qualitatively. There are some factors are still as fresh as they were in the 1977. Also some factors have been seen as redundant and have been integrated into other factors to better reflect modern practice. Next we will list all the most significant and widely applied quality factors.

◆ Performance (Efficiency)

Performance represents the responsiveness of the system, which can be measured by the time required to respond to events (stimuli) or by the number of events that are processed in a period of time. Typically, performance quality attributes are expressed as the number of transactions per unit of time or the length of time required completing a single transaction.

DeGrace and Stahl refer to this quality as efficiency. Code execution efficiency is the economy required by the customer, such as the run time, response time, and memory used. The characterization presented by DeGrace and Stahl focuses on the implementation (non-architectural) practices such as taking advantage of various compiler optimizations, keeping loop constructs free of unnecessary computations, grouping data for efficient processing, indexing data, and using virtual storage facilities (relational databases) to optimize data storage efficiency.

Performance has typically been a driving factor in system architecture and frequently compromises the achievement of all other qualities.

- Modifiability

A modifiable architecture is one that can be added, extended or grown over time, possibly by other developers or customers in a straightforward way. In other words, it is more cost effective to add features to the existing application than to build a new application. Modifiability is sometimes called maintainability. A modifiable application can have new features added without requiring architectural rework, such as changes to how functions are distributed across components. The measure of modifiability is the cost and effort required to make a change to an application.

The types of stimuli related to modifiability are change requests for functions, platforms, quality attributes, or operating environment. Function change requests are probably the most common event. The architectural response to modification change requests may be the addition, modification, or deletion of components, connectors, or interfaces, as well as a cost and effort measure.

- Usability

Usability typically refers to the usability with respect to the end user. However, usability also addresses other system users such as system maintainers, operators, and porting engineers.. End users (modeled as actors) are concerned with functionality, reliability, usability, and efficiency. Maintainers (often not modeled) are concerned with maintainability. Quality in use is the overall subjective quality of the system as influenced by multiple quality attributes. Quality in use is measured using scenarios.

- Portability

Portability is the ability to reuse a component in a different application or operating environment such as hardware, operating systems, databases, and application servers. Portability can be considered a specialized type of modifiability. The measure of the portability of a system is based on how localized the changes are.

Another word for portability is extensibility. If it has a higher extensibility, it

is more easily to be extended. For example, Eclipse, which is typical IDE for java application. Owing to its plug-in structure, it is quite easily to be extended. In Eclipse, expect the kernel platform; anything else in Eclipse is a plug-in.

Portability is often associated with porting the source code of an application from one operating system, such as Windows, to another operating system, such as Linux. Modifying a system or application so that users can access it from a Web interface is an example of extensibility. The change might not introduce new functionality into the system; rather, its operating environment has changed. Best says extensibility is "important because application designers can never foresee all the changes that will occur in an organization's operating environment."

◆ Testability

In the McCall et al model, it is defined as the cost of program testing for the purpose of safeguarding that the specific requirements are met.

Because of its traditional position in development models like the Waterfall or Boehm's spiral, testing is easily identified as a quality factor. The testing process is well matured at this stage. A substantial amount of item or unit testing is completed by programmers as part of their normal role. Testing interacts with all other quality factors. For example, to check accuracy a test plan is needed. To test reliability a test plan is needed. To test efficiency a test plan is needed and so on. So all testing must be performed in accordance with pre-defined plans, using pre-defined tests data to achieve pre-determined results. Numerous test strategies are used. They include functional or black box testing, structural or white box testing and finally residual defect estimation. These strategies can be employed using difference techniques. [4]

◆ Maintainability

It is defined as the cost of localizing and correcting errors by McCall.

We can say that maintainability has connection with testability and the

following reliability. If easily tested, it is going to be comparatively easier maintained. As long as it has higher maintainability, the localization and correction of errors are easier and to be more reliable. Finding and correcting errors is just one aspect of maintenance. Ghezzi et al. (1991) divide maintenance into three categories: corrective, adaptive and perfective and only corrective is concerned with correcting errors as suggested by McCall.

- Reliability

It is defined as the extent to which a program can be maintained so that it can fulfill its specific function by McCall

Reliability in engineering terms is the ability of a product or component to continue to perform its intended role over a period of time to pre-defined conditions. And the same applies to the systems environment where reliability is measured in terms of the mean time between failures, the mean time to repair, the mean time to recover, the probability of failure and the general availability of the system. [4]

3.3 Software metrics

To have more accurate schedule and cost estimates, better quality products, and higher productivity. We defined the above software quality factors. Now as we consider the above-mentioned factors it becomes very obvious that the measurements of all of them to some discrete value are quite an impossible task. Therefore, another method was evolved to measure out the quality. And these can be achieved through more effective software management, which can be facilitated by the usage of software metrics. Given that software development is extremely complex, especially for Oracle PL/SQL, in the current software market, there are few well-defined, reliable measures of either the process or the product to guide and evaluate development. Thus, accurate and effective estimating, planning, and control are nearly impossible to achieve. This is the goal of software metrics-the identification and measurement of the essential parameters that affect software development. [5]

3.3.1 Software metrics definition

The first definition of software metrics is proposed by Norman Fenton

(...)software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterize properties of software code (these are the classic software ’metrics’) through to models that help predict software resource requirement and software quality. The subject also includes the quantitative aspects of quality control and assurance - and this covers activities like recording and monitoring defects during development and testing.

Another definition of software metrics is done by Paul Goodman

The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products". Applied To Engineering & Management Processes, Products & To Supply

Essentially, software metrics deals with the measurement of the software product and the process by which it is developed. These metrics are used to estimate/predict product costs and schedules and most importantly, they measure the productivity and product quality. The measured information gained can be used to control, develop and lead to improved results. [5]

3.3.2 Metrics classification

Normally, software metrics can be classified into process metrics and product metrics. Process metrics measure the software development process, such as the total development time, the method that the developer used, the effort associated with process or activity, the number of incidents of a specified type arising during process or activity, and the average experience level of all the

developers. While as the product metrics are measurements of the software product itself. They measure the software product at any certain stage of development, from the very beginning requirement analysis stage to the end product releasing. We can say that all the artifacts, documents and prototypes produced during the process are considered as products. All these process outputs can be measured in term of quality, size or complexity.

Of course apart from the process and product metrics, there are other ways of classifying them. For instance, objective metrics and subjective metrics can be another type of classification of software metrics. Generally, object metrics shall have the identical results by measuring the same product or process. On the other hand, subjective metrics would lead to variant results even with qualified observers since the subjective judgment is involved in arriving at the measured values. [5] Take the product metrics as an example, the size of the product measured in lines of code (LOC) is an objective measure, for any observer, the same LOC should be obtained for a given program. For the subjective process metrics, we can consider the average experience level of the development group. Although most programs can be easily measure, those on the borderline between categories might be classified in different ways by different observers.

Software metrics can be also categorized into primitive metrics and computed metrics. Those can be directly observed and calculated are categorized as primitive metrics. For example, the programs size (in LOC or KLOC), number of defects, number of bugs. For the PL/SQL programming source, the number of tables, number of views, number of packages, and number of procedures are all primitive metrics. While as the computed metrics are those that cannot be directly observed but are computed in a certain manner from other metrics. In my project, the most used computed product metric is the Cyclomatic complexity. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. In the following, I will describe the cyclomatic complexity in detail. Other examples can be the number of defects per thousand line of code (defects/KLOC). We can see that the number of defects and KLOC themselves are primitive metrics.

3.4 Software metrics selected for the PL/SQL

3.4.1 Product Metrics

In my dissertation, a number of product metrics are chosen and discussed given that they are more widely used and applied. At the beginning of most software development, more concerns are about the product metrics and the measurement of the characteristics of the source code.

- Lines of Code

The simplest way to measure the size of a program is to count the lines. This is the oldest and most widely used size metric. Lines of code, or LOC, dose look like a simple concept. However, it's not. There are several ways to count the lines. Depending on what you count, you get a low or a high line count. These differences involve the treatment of blank lines and comment lines, non-executable statements, multiple statements per line, and multiple lines per statement. Moreover, how to count the reused lines of code is a big issue.

The physical lines count (LINES) is a simple but not a perfect way to measure code size. Since a logical line can expand over several lines, the physical line count exaggerates code size. A common problem in line counts is also that empty (or whitespace) lines, as well as comments, are included in the count. With improper line counts, you can appear really productive by hitting the Enter key, or alternatively, pretend that you are writing tighter code by deleting all comments.

The logical lines of code metric (LLOC) has both advantages and disadvantages. It is a simple measure, easy to understand, and widely used. You can use it to measure productivity, although you need to be cautious, because programming style can have an impact on the values. You can also estimate the number of defects per 1000 LLOC. [28]

Nevertheless, LOC has been studied to be a useful metric as a predictor of

program size and complexity. But the problem is that for different programming language, same LOC absolutely does not mean same or even similar program size and complexity. For example, according to my previous experience, 1000 physical lines of code of Matlab could be enough to do the simple vehicle plate extraction from a .jpg image while as to implement the same function, C needs 5000 lines of code or much more. In a study, Levitin concludes that LOC is a poorer measure of size than Halstead's program length, N. It will be discussed below.

- Function Points

Function points were defined in 1979 in A New Way of Looking at Tools by Allan Albrecht at IBM.[2] The functional user requirements of the software are identified and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces. Once the function is identified and categorized into a type, it is then assessed for complexity and assigned a number of function points. Each of these functional user requirements maps to an end-user business function, such as a data entry for an Input or a user query for an Inquiry. [6]

Function points are intended to be a measure of program size and thus effort required for development.

- Cyclomatic Complexity – V(G)

The above two metrics are developed specially for measuring the size of the software program. When considering the measurement of software complexity. The first metric I present here is the cyclomatic complexity.

Cyclomatic complexity (or conditional complexity) is a software metrics (measurement). It was developed by Thomas J. McCabe, Sr. in 1976 and is used to indicate the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. The concept, although not the method, is somewhat similar to that of general text complexity measured by the Flesch-Kincaid Readability Test. [8]

Cyclomatic complexity is computed using the control flow graph (see figure 3.1) of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

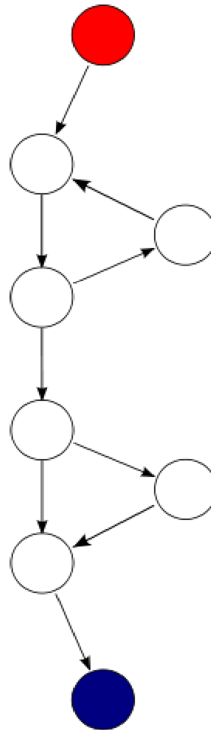


Figure 3-1 A control flow graph of a simple program

For any software program, a control flow graph, G, can be drawn as we the graph above. In which each node corresponds to a block of sequential code and each arc corresponds to a branch or decision point of the program. And then the cyclomatic complexity can be computed from such a graph by a simple formula:

$$V(G) = E - N + 2P \qquad 3.1$$

In the formula, E is the number of edges, N is the number of nodes, and P is

the number of connected components in the graph. For a single program (or subroutine or method), P is always equal to 1. Cyclomatic complexity may, however, be applied to several such programs or subprograms at the same time (e.g., to all of the methods in a class), and in these cases P will be equal to the number of programs in question, as each subprogram will appear as a disconnected subset of the graph. So normally, we have the formula as follow: [8]

$$V(G) = E - N + 2 \qquad 3.2$$

A number of studies have shown the correlation between cyclomatic complexity and the number of defects contained in a module. Those modules having the higher complexity tend to also contain the more defects.

For example, a 2008 study by metric-monitoring software supplier Enerjy analyzed classes of open-source Java applications and divided them into two sets based on how commonly faults were found in them. They found strong correlation between cyclomatic complexity and their faultiness, with classes with a combined complexity of 11 having a probability of being fault-prone of just 0.28, rising to 0.98 for classes with a complexity of 74. [7]

Overly complex modules are more prone to error, are harder to understand, are harder to test, and are harder to modify. So it is advisable to limit the value of the cyclomatic complexity.

It is been widely applied in many companies to set the limit of cyclomatic complexity. But it is controversial regarding the limit. The original limit of 10 as proposed by McCabe has significant supporting evidence, but limits as high as 15 have been used successfully as well.

In my project, for the sake of tracking and assurance the quality of the PL/SQL program, I divide the thresholds of cyclomatic complexity into four.

- N. of objects with V(G) between 0 and 10 (good)
- N. of objects with V(G) between 11 and 20 (acceptable)

- N. of objects with V(G) between 21 and 50 (to be reengineered)
- N. of objects with V(G) higher than 50 (untestable)

➢ Information Flow

The information flow within a program structure may also be used as a metric for program complexity. It is proposed by Kafura and Henry. It counts the number of local information flows entering (fan-in) and exiting (fan-out) each procedure. The procedure's complexity is defined as:

$$C = [\text{procedure length}] * [\text{fan-in} * \text{fan-out}] \qquad 3.3$$

Researches' have shown that such metric is a useful measure of software program maintainability.

➢ Halstead's Software Metrics

Halstead distinguished software science from computer science. The premise of software science is that any programming task consists of selecting and arranging a finite number of program "tokens", which are basic syntactic units distinguishable by compiler. In Halstead's software metrics, a computer program is considered as a collection of tokens that can be classified as either operators or operands. The primitive measures of Halstead's software science are:

- n1 = the number of distinct operators
- n2 = the number of distinct operands
- N1 = the total number of operators
- N2 = the total number of operands

Based on these primitive measures, Halstead developed a system of equations expressing the total vocabulary, the overall program length, potential volume

for an algorithm, the actual volume, program difficulty, and other features such as development effort and the projected number of faults in the software. They are express as follows: [8]

- Program length: $N = N1 + N2$
- Program vocabulary: $n = n1 + n2$
- Volume: $V = N \times \log_2 n$
- Difficulty : $D = \frac{n_1}{2} \times \frac{N_2}{n_2}$
- Effort: $E = D * V$

Although Halstead's metrics has had a great impact on software measurement, it also has been criticized for many years. Empirical studies provided little support to the equations except for the estimation of program length. It is criticized that to predict program length, data on N1 and N2 must be available, and meanwhile, they must be determined, the program should be completed or near completion. Therefore, the productiveness of the equation is limited.

➢ Reliability Metrics

It is also very important to have an idea of the probability of software failure, or the rate at which software errors would occur. Software Reliability is an important factor affecting the reliability of the whole system.

It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The high complexity of software is the major contributing factor of Software Reliability problems. No good quantitative methods have been developed to represent Software Reliability without excessive limitations. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability. [29]

But until now, there is no clear definition to what aspects are related to software reliability. We cannot find a suitable way to measure software reliability, and most of the aspects, such as mean time to failure (MTTF) related to software reliability. Even the most obvious product metrics such as software size have not uniform definition. As we have discussed before.

- Maintainability Metrics

Numbers of efforts have been exerted to figure out the definition of metrics that can be used to measure or predict the maintainability of the software product. As the early study directed by Curtis, Etal shows that the Halstead's metrics, cyclomatic complexity which used for the prediction of psychological complexity of software could be profitably used as a measure of the maintainability. From the maintainability definition: Propensity to facilitate updates to satisfy new requirements. Thus the software product that is maintainable should be well-documented, should not be complex, and should have spare capacity for memory, storage and processor utilization and other resources. We see that a lot of aspects should been concerned in the maintainability metrics.

3.4.2 Metrics chosen for PL/SQL

Numerous evidences have been shown both from university academic research and from industry experience that the conscientious application of software metrics can significantly improve our understanding and management of the software development. Many software quality metric models have been developed and applied for the estimating, planning and controlling of project. However, great care must be taken in selecting the metrics and recalibrating them, if necessary, making them source code oriented. Which means a specific model and set of metrics is selected based upon the objectives defined and cost considerations identified.

Metrics and models available should be compared with respect to their apparent ability to meet the objectives (goals) requested. Plenty of papers have dealt with the size and complexity of the software product and they are also the most widely recognized and used in the practical application of

software quality improvement and assurance.

In our project, the set of metrics chosen and implemented are as follows:

- Table Count: It counts the total number of tables of a Oracle database source
- View Count: It counts the total number of views of a Oracle database source
- Package Count: It counts the total number of packages of a Oracle database source
- Procedure Count: It counts the total number of procedures of a Oracle database source
- Function Count: It counts the total number of functions of a Oracle database source
- NCSS: It counts the total number of Non Commented Source Statements of a Oracle database source. It counts only statements.
- Cyclomatic Complexity V(G)
    - N. of objects with V(G) between 0 and 10 (good): low complexity
    - N. of objects with V(G) between 11 and 20 (acceptable): medium complexity
    - N. of objects with V(G) between 21 and 50 (to be reengineered): high complexity
    - N. of objects with V(G) higher than 50 (untestable): high complexity
    - Overall % of objects with high complexity (21+): the overall percentage of objects with cyclomatic complexity higher than 20.

The reason that we did not choose the LOC as the metric is given by Bill Gates:

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight." By only examine the line of code or logic line of code, we cannot actually discover anything meaningful. From the business point of view, it is not worthy of doing so because in the software metrics market, there already exits software which supply the measurement of line of code for programming language PL/SQL. It is ClearSQL. ClearSQL is a code review and quality control tool for Oracle PL/SQL. Furthermore, our purpose is to exam and ensures the quality of whole Oracle software, not just a measures metrics on a per-subroutine basis.

And that's why we have chosen the number of tables, views, packages, function and procedures as the very basic and fundamental metrics. Have a broad perspective of the whole software project can be more beneficial.

It is generally accepted that more complex procedures are more difficult to understand and have a higher probability of defects than less complex procedures. As a result, complexity has a direct impact on overall quality and, more importantly, on maintainability. While there are many different types of complexity measurements, the one used by us is Cyclomatic Complexity (v(G)), which is the amount of decision logic in a single software module. It gives the number of linearly independent tests and is used to predict the test and maintenance effort. Although ClearSQL has the function of calculating cyclomatic complexity as a metrics, it would not make our project redundant since what we have is not just the value of cyclomatic complexity, we have divided the cyclomatic complexity into four levels according to Original McCabe theory. It makes us possible to have a more specific control of the Oracle database development and the final product.

The more paths the software module can take, the higher the complexity it has. Overly complex modules are prone to error, and are harder to understand, test and modify. Limiting complexity at all phases of the development life cycle helps avoid the pitfalls associated with high complexity software.

3.5 Software quality control tools

There are several tools to measure the code quality.

1. FindBugs– uses static analysis to look for bugs in Java code. It is free software, distributed under the terms of the Lesser GNU Public License. It discovers possible NullPointerExceptions and a lot more complicated bugs in projects.

2. PMD– scans Java source code and looks for potential problems like:

- Possible bugs - empty try/catch/finally/switch statements
- Dead code - unused local variables, parameters and private methods
- Suboptimal code - wasteful String/String Buffer usage
- Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
- Duplicate code - copied/pasted code means copied/pasted bugs

With the maven plugin you can do: *mvn pmd:pmd*

3. Checkstyle- It is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration file is supplied supporting the Sun Code Conventions. As well, other sample configuration files are supplied for other well known conventions. [43]

Checkstyle can check many aspects of your source code. Historically its main functionality has been to check code layout issues, but since the internal

architecture was changed in version 3, more and more checks for other purposes have been added. Now Checkstyle provides checks that find class design problems, duplicate code, or bug patterns like double checked locking.

The very popular IDE Eclipse has a Checkstyle plug-in. With the Checkstyle Eclipse plug-in (figure 3-2) your code is constantly inspected for problems. Within the Eclipse workbench you are notified of problems via the Eclipse Problems View (figure 3-3) and source code annotations just as you would see with compiler errors or warnings.

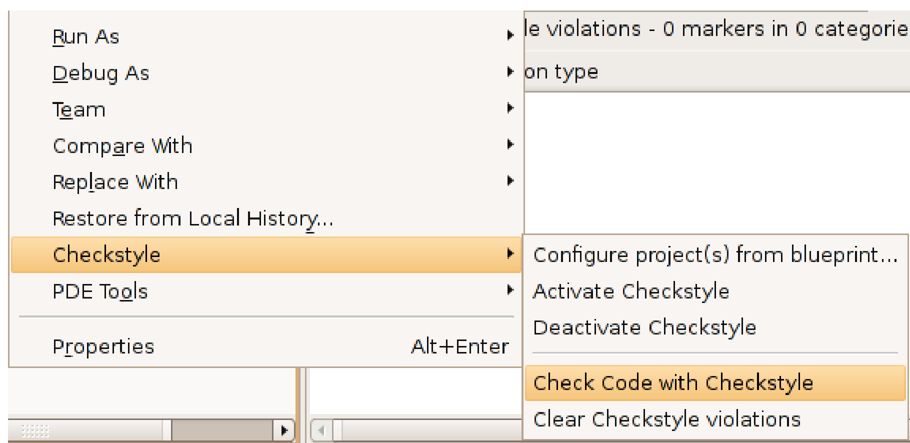


Figure 3-2 Checkstyle Eclipse Plug-in

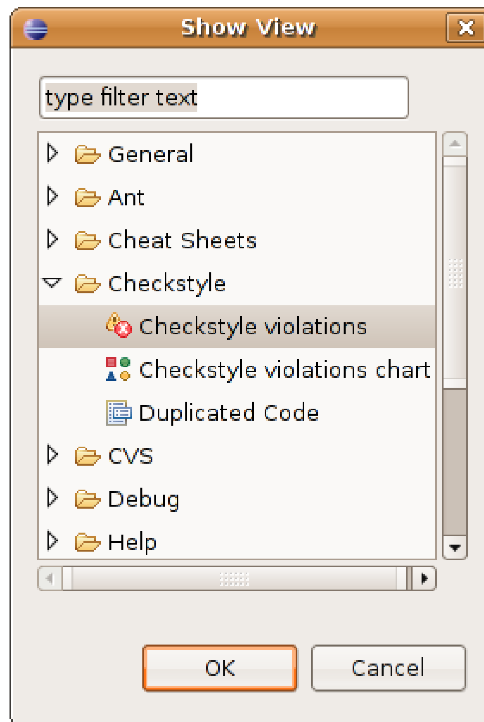


Figure 3-3 Eclipse show view of Checkstyle plug-in

4. JarAnalyzer – Is a dependency management utility for jar files. Its primary purpose is to traverse through a directory, parse each of the jar files in that directory, and identify the dependencies between the jar files.

5. HammurAPI – a code quality governance platform

6. ClearSQL- is a code review and quality control tool for Oracle PL/SQL. It generates a series of industry standard quality control metrics about PL/SQL source code to identify potential problems in the development and maintenance of your software and to fine-tune your software development process. [44]

ClearSQL creates and visualizes "clickable" flowcharts and call tree diagrams and CRUD matrices of PL/SQL code that help you find the point of possible code refactoring or module restructuring, discover data flows between subroutines and dataset objects due to DML statement execution, and analyze

the consistency of functional requirements and to identify performance problems.

Search/Error List | Structure View | Code Review | Code Metrics | Instrumented Code

Subprogram	LOC	eLOC	lsLOC	Comment Lines	Blank Lines	Input parameters	Return Points	V(g)	Interface Complexity
MakeTransPackage	33	29	18	0	4	0	0	4	0
MakeTrans	24	22	8	1	5	1	0	1	1
MakeTransAux	61	57	14	0	16	1	0	3	1
CheckTRN	63	51	24	0	15	1	10	11	11
CheckAmount	46	35	12	0	7	2	5	10	7
CreateTRN	49	47	9	0	3	6	1	1	7
GetStatusLogType	13	11	4	0	2	1	2	2	3
GLOBAL	6	3	1	10	14	0	0	0	0

Figure 3-4 ClearSQL

7. Sonar- another interesting approach to use several code quality tools at a time. With Sonar it is possible to see the violations or possible bugs. So you are looking at the improvements and you will not get lost in the mass of bugs at the beginning. In our project, in order to use sonar as a platform to improve the quality of software development process and product, we developed a plug-in of sonar. In default sonar is for the analysis of the Java code, with our plug-in, we could have the analysis also for the PL/SQL. In next chapter, we will present of design and development of this plug-in in detail.

3.6 Chapter summary

In this chapter, we have made an introduction of software quality, the definition, the classification and the software quality model and factors, as well as the benefits of considering the quality control in software development. Afterward, based on the quality, we explained the software metrics and its classification and application to improve software quality. And then we presented the metrics that have been chosen and implemented in our PL/SQL analyzer. At last, a variety of most commonly used software quality control tools are discussed.

4. Metrics Plug-in design and implementation on the Client-side: Eclipse

4.1 Eclipse Plug-in Infrastructure

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. [21]

It is an extensible platform for tool integration. To the thousands of students and researchers, Eclipse represents a stable platform to innovation, freedom, and experimentation. Meanwhile to all those individuals, groups and organizations, Eclipse is a vendor-neutral platform to tool integration supported by a diverse Eclipse Ecosystem. [22] For example, it has been well integrated with Maven 2 for project building and management and Checkstyle for static code inspection, as well as Hudson for continuous integration.

Before elaborating the Eclipse Plug-in and its development, I would like to present the whole architecture of the Eclipse and its plug-in structure.

4.1.1 Eclipse platform architecture

The Eclipse Platform (see figure) is a framework with a powerful set of services that support plug-ins, such as JDT and the Plug-in Development Environment. It consists of several major components: the Platform runtime, Workspace, Workbench, Team Support, and Help. The primary purpose of the Platform subproject is to enable other tool developers to easily build and deliver integrated tools. [23]

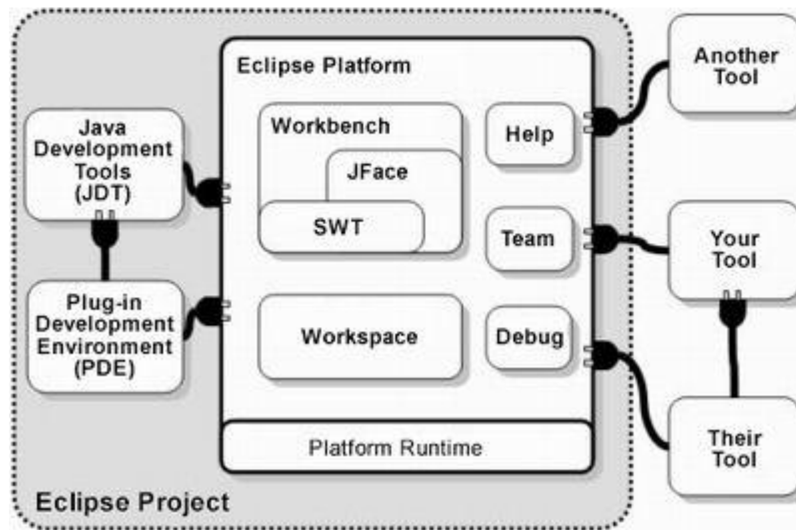


Figure 4-1 Eclipse platform architecture

- Platform Runtime - The Platform runtime is the kernel that discovers at startup what plug-ins are installed and creates a registry of information about them. It only loads the plug-ins when they are necessary. Apart this kernel, anything else is plug-in.
- Workbench – Eclipse workbench provide the UI (User Interface) service for Eclipse. It is built by Eclipse SWT (standard widget tools), which is an alternative for Java AWT/Swing, and JFace which is a higher level tool for UI design.
- Workspace – It is a logical collection of projects, a directory on your hard drive where Eclipse stores the projects that you have created. The Workspace is also responsible for notifying other interested plug-ins about resource changes, such as files that are created, deleted, or changed. The first time you start a Eclipse, you will be required to specify the directory of workspace, that the place where your Eclipse project locate.

These are the most important three components of Eclipse platform. Besides, to develop a Eclipse project, we also need JDT and PDE.

- JDT - JDT (java development tools) is the tools to provide Eclipse to develop Java applications. It adds a Java project nature and Java perspective to the Eclipse Workbench as well as a number of views, editors, wizards, builders, and code merging and refactoring tools.

- PDE – Plug-in development environment provides a comprehensive set of tools centered around OSGi bundle development. It provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP (Rich Client Platform) products.

From the figure above, we see that tools of different parts are all plugged into the Eclipse platform and coordinate to run as a whole. This is how the Eclipse works. The platform is like a aircraft carrier. The fighters, military helicopters, and Cruise ships will participate the war all based on that aircraft carrier.

4.1.2 Plug-in Structure

Eclipse is not a single, monolithic program, but rather a small kernel surrounded by hundreds of plug-ins.

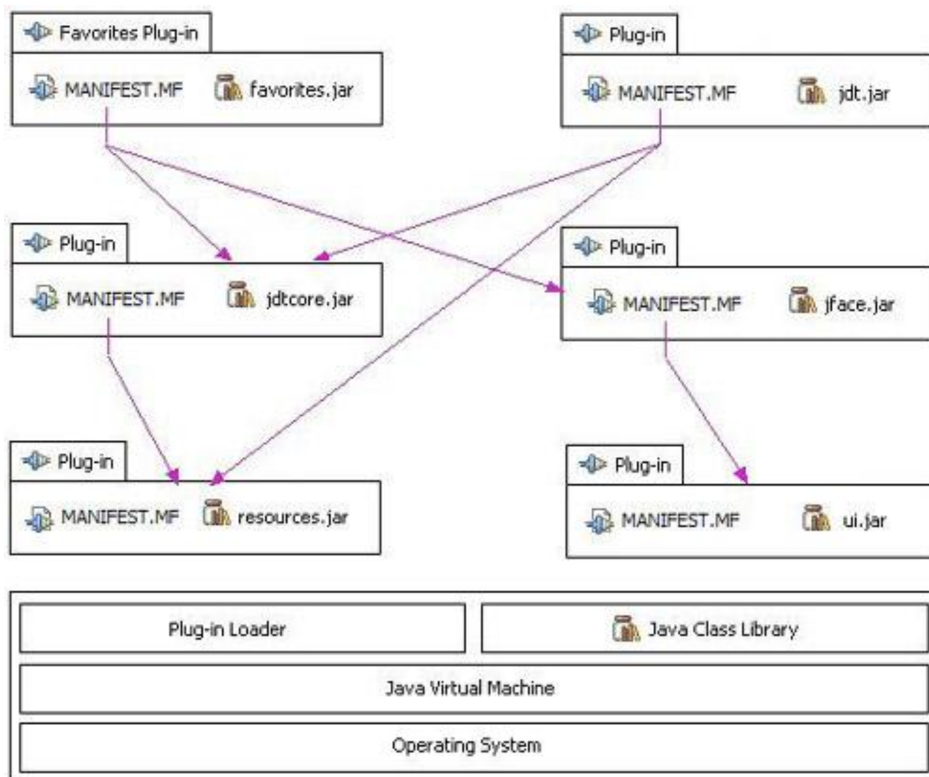
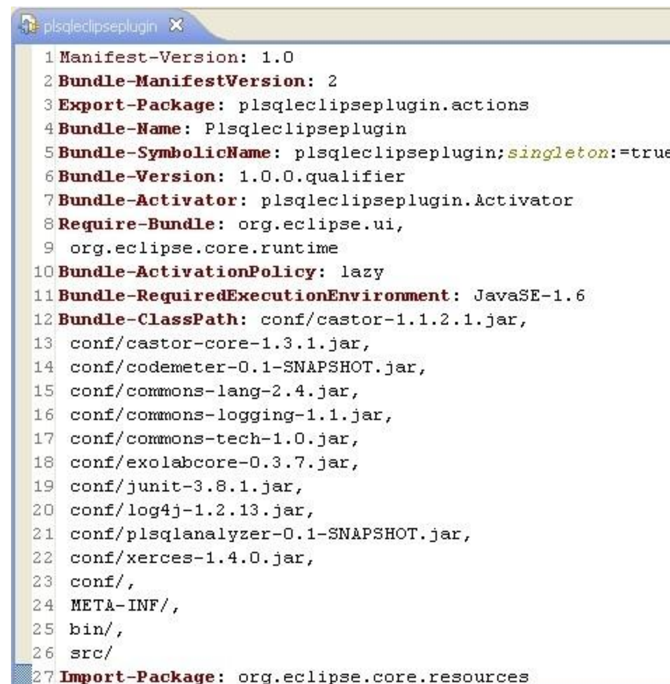


Figure 4-2 Eclipse plug-in structures

In the figure above, we assume that the top left plug-in is newly developed. It is depended on another two plug-ins which refer to jdtcore.jar and jface.jar. And the jdtcore.jar plug-in relies on services provided by plug-in resource.jar while as the jface.jar plug-in relies on the ui.jar plug-in. This is how Eclipse works while it is running. Each plug-in may rely on the service provided by another plug-in, and may in turn provide services on which other plug-in may reply. This plug-in mechanism is a lightweight software component framework. In addition it allows Eclipse to be extended using other programming languages such as C and Python.

From the figure above we can also see that the dependency arrow lines all start from a MANIFEST.MF file and point to another plug-in Jar file. So what is exactly the MANIFEST.MF file?

In fact, each plug-in has two manifest files, META-INF/ MANIFEST.MF and plugin.xml (see figure 4-3), which define how the plug-in related to external resources of the system.



```
plsqleclipseplugin
1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Export-Package: plsqleclipseplugin.actions
4 Bundle-Name: Plsqleclipseplugin
5 Bundle-SymbolicName: plsqleclipseplugin;singleton:=true
6 Bundle-Version: 1.0.0.qualifier
7 Bundle-Activator: plsqleclipseplugin.Activator
8 Require-Bundle: org.eclipse.ui,
9  org.eclipse.core.runtime
10 Bundle-ActivationPolicy: lazy
11 Bundle-RequiredExecutionEnvironment: JavaSE-1.6
12 Bundle-ClassPath: conf/castor-1.1.2.1.jar,
13  conf/castor-core-1.3.1.jar,
14  conf/codemeter-0.1-SNAPSHOT.jar,
15  conf/commons-lang-2.4.jar,
16  conf/commons-logging-1.1.jar,
17  conf/commons-tech-1.0.jar,
18  conf/exolabcore-0.3.7.jar,
19  conf/junit-3.8.1.jar,
20  conf/log4j-1.2.13.jar,
21  conf/plsqlanalyzer-0.1-SNAPSHOT.jar,
22  conf/xerces-1.4.0.jar,
23  conf/,
24  META-INF/,
25  bin/,
26  src/
27 Import-Package: org.eclipse.core.resources
```

Figure 4-3 Manifest.mf file

The behavior, dependencies and services of a plug-in are all declared in the special XML file named Plugin.xml. But in the newer versions of Eclipse that use OSGi, dependency information has been broken out into the manifest.mf file, leaving the plugin.xml file containing only XML definitions of extensions and extension points.

It is worthy take a look at the manifest.mf file. There are many options specified in this file:

- Export-Package: This property specifies all the packages to publicly expose to other plug-ins.

- Bundle-Name: It specifies the name of this plug-in application

- Bundle-Version: This property specifies the version number of the bundle. Package imports and required bundle specifications may include a bundle version number.

- Bundle-Activator: This class is used to start and stop the bundle. In the example above, the bundle-activator class is plsqleclipseplugin.Activator, which extends org.eclipse.ui.plugin.AbstractUIPlugin.

- Required-Bundle: This property specifies which bundles and their exported packages to import for use in the given bundle. The required bundle or required plug-ins org.eclipse.ui, org.eclipse.core.runtime is two of the most widely used plug-ins imported.

- Bundle-ClassPath: This property specifies the CLASSPATH to use for the bundle. The property may contain references to directories or jar files inside the bundle jar file. This property is very important to ensure the good behavior of the plug-in. If not well specified, basically, the plug-in does not work after installation.

```xml
<plugin>
   <extension
         point="org.eclipse.ui.actionSets">
      <actionSet
            label="Sample Action Set"
            visible="true"
            id="plsqleclipseplugin.actionSet">
         <menu
               label="ORACLE"
               id="sampleMenu">
            <separator
                  name="sampleGroup">
            </separator>
         </menu>
         <action
               label="Last Analysis"
               icon="icons/sample.gif"
               class="plsqleclipseplugin.actions.SampleAction"
               tooltip="Previous analysis report"
               menubarPath="sampleMenu/sampleGroup"
               toolbarPath="sampleGroup"
               id="plsqleclipseplugin.actions.SampleAction">
         </action>
         <action
               label="Launch new analysis"
               class="plsqleclipseplugin.actions.SampleAction2"
               menubarPath="sampleMenu/sampleGroup"
               id="plsqleclipseplugin.actions.SampleAction2">
         </action>
      </actionSet>
   </extension>
</plugin>
```

Figure 4-4 Plugin.xml

As we have discussed above, this XML file only defines the extension points of each plug-in. When the facilities of a plug-in are to be made directly available to the user, one or more user interface elements have to be added to the base Eclipse workbench. For example, Checkstyle has the Eclipse plug-in to enable it execute the static code analysis. To make this plug-in available to users, a menu is added into the Eclipse workbench.

The process of adding some processing element or elements to a plug-in is known as an extension. This process is not restricted to UI elements, however. Any plug-in may allow other plug-ins to extend it by adding processing elements. [24]

Extension and extension-point are standard Eclipse plug-in terminology. Host

plug-in, extender plug-in, and callback object are terms which are commonly used to describe the relationship between the plug-ins.

To be simple, an extension is defined by an extender plug-in which adds the host plug-in a function. For example, our PL/SQL analysis plug-in enables the Eclipse to do PL/SQL code analysis and report, in this case, the host plug-in is the Eclipse kernel. To make this function available to users, a simple menu or menu item is added to the Eclipse workbench, and this menu or menu item is the so called callback object, through which the host and extender plug-ins communicate. A single act of extension can also add more than one callback object to the environment. For example, Eclipse allows a set of menus to be added to its user interface via a single extension.

Take our plug-in as a vivid example. In the Plugin.xml file only one plug-in extension point is defined which is org.eclipse.ui.actionSets. Inside this extension, there is one menu labeled name Oracle which has two actions, which are labeled as Last analysis and Launch new analysis (see figure 4-5).

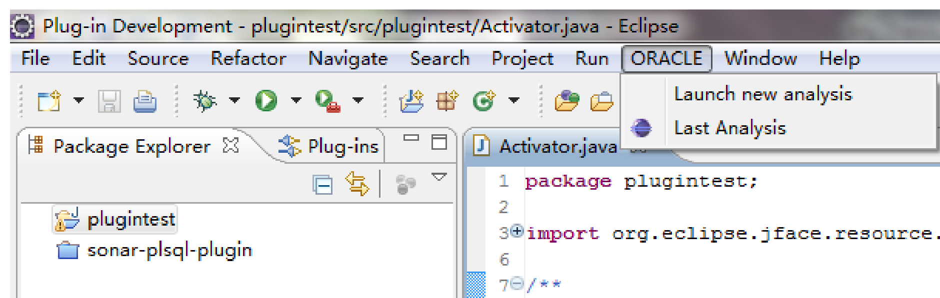


Figure 4-5 Two actions of PL/SQL plug-in

One more point we should notice about the plugin.xml is the action class. It refers to the class which defines the behavior of the action. And I will elaborate this in the next chapter.

4.2 PL/SQL plug-in development

4.2.1 Plug-in interface implementation with SWT/JFace

There have already existed plenty of papers and resources addressing this topic. What I will do is describing how I have adopted the SWT/JFace to develop my PL/SQL Eclipse plug-in.

The Standard Widget Toolkit (SWT) is a thin layer on top of the platform's native controls. SWT provides the foundation for the entire Eclipse user interface (UI). It provides a rich set of widgets that can be used to create either standalone Java applications or Eclipse plug-ins. [22]

Widgets, Shell and Display are the basic blocks for a Eclipse plug-in. Display is responsible for managing the event loops and controlling communication between UI threads and non-UI threads. Every SWT should have at least one Display; otherwise this application won't be visible. Shell is the window for user interface managed by the operation system. Each SWT could have several Shells. Based on each Shell, man Widgets can be defined, such as button, text, label, menu, menu option, list, table as so on. These widgets have plenty of attributes can be specified while being defined. The structure can be seen as follow.

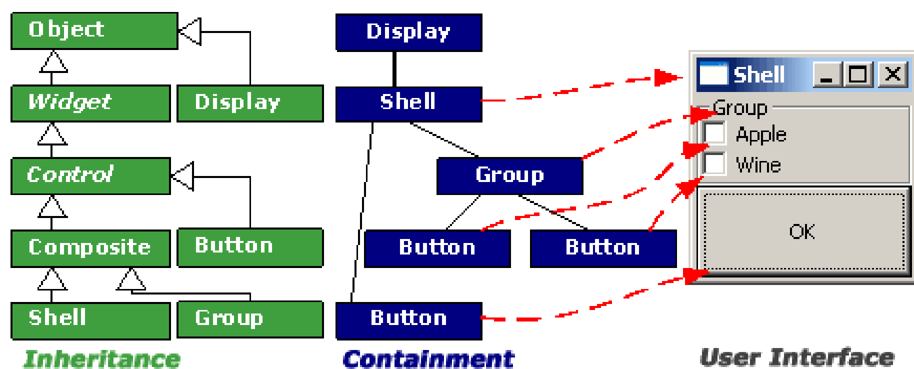


Figure 4-6 structure and relationship of SWT

This figure shows the SWT from two perspectives. 1) It is from the inheritance point of view. 2) It relates the exact block or widget to the real object.

In our PL/SQL plug-in, since we want to see the static analysis of the PL/SQL code, it would be better to show them with tables. Table is able to specify the metric name, the measurement description, and the results clearer. Next I list all the steps I have done to create an Eclipse plug-in with SWT/JFace.

1. Create a Display

To create a Display, we need to define an action which should be specified in the Plugin.xml file as I have mentioned before. In the Plugin.xml file, there is extension point and several actions which is also called call back object can be defined as much as we want.

We will define two actions, one action for displaying the previous analysis result. Another is for initializing new code analysis.

2. Create Shells

Two actions correspond to two shells. The result action needs a shell with tables to display the general information and the details of PL/SQL. The action for launching new analysis needs a wizard or directory dialogue to enable the user to navigate to select the project to be analyzed.

3. Set the layout format

To show the analysis result clearly, we set the shell layout as vertical. The above table shows the general information about each Measurement and the lower table shows the detail information as long as we double click on one of the measurements. (See figure 4-7)

There are two tables in this figure. The upper table shows the index, measurement name, description, and the value while as the lower shows the

details of a certain measurement. For example, if we double click the 7th measure, cyclomatic complexity from 0 to 10 with the result vale of 149, we can see the corresponding source name, line, column, and value.

Project Database: Q:\TRK\database

File

Index	Measure	Description	Value
1	Table Count	Total number of tables is	75
2	View Count	Total number of views is	0
3	Package Count	Total number of packages is	4
4	Procedure Count	Total number of procedures is	2
5	Function Count	Total number of functions is	33
6	NCSS	Total NCSS (packages, procedures, functions)	7803
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	55
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	3
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	6
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	2

Source	Line	Column	Value
FUNCTION trk.AGGIORNA_OPERAZIONE.sql	0	0	1.0
FUNCTION trk.ALL_SC_BOX_IN_MORE_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_BOX_IN_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_BOX_NOT_INSTATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_CONF_IN_MORE_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_CONF_IN_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_CONF_NOT_INSTATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_SCATOLE_IN_MORE_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_SCATOLE_IN_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_SCATOLE_NOT_INSTATE.sql	0	0	1.0

Figure 4-7 Two tables to show the analysis result

4. Create widgets

The widgets we have defined are tables, menus and menu items.

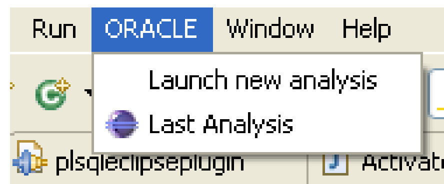


Figure 4-8 Menu Oracle

Project Database: Q:\TRK\database

File

Select Project	Ctrl+S
Create Html Report	Ctrl+H
Exit	

		Description	Value
		Total number of tables is	75
		Total number of views is	0
3	Package Count	Total number of packages is	4
4	Procedure Count	Total number of procedures is	2
5	Function Count	Total number of functions is	33
6	NCSS	Total NCSS (packages, procedures, functions)	7803
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	55
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	3
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	6
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	2

Figure 4-9 Menu File

The above two figures show the menus and menu items created. By clicking ORACLE menu, we have two menu items, launch new analysis and last analysis. By clicking the File menu, we have three menu items, select projects, create html report and exit.

To launch a new analysis, we implemented a directory dialogue to let user navigate between the files to select an Oracle project. (See figure 4-10)

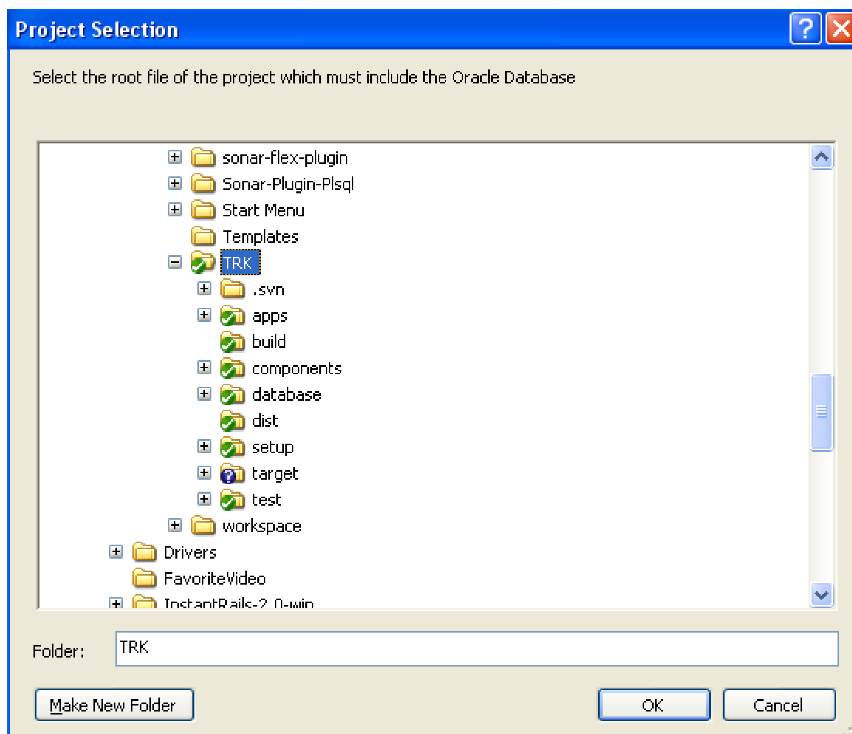


Figure 4-10 Directory dialogue to select a project

5. Code the logic part.

After creating the framework for the plugin, it's time to enable the plugin to execute analysis. This part is essential and crucial. It is core of the PL/SQL plugin not only in the client side Eclipse, but also in the server side Sonar. Both analyzers are based on it. We will split one chapter to describe it.

4.2.2 Plug-in logic implementation

The main function of logic part is executing the analysis on a given piece of code, getting analysis results and displaying them to the right place.

Both the plug-in of client side and server side are based on a core PL/SQL Analyzer. Pl/SQL Analyzer is a custom Java tool which computes software metrics for Oracle Pl/SQL database scripting language. Pl/SQL Analyzer was

developed by Technology Reply's internal Software Factory.

Before explaining the PL/SQL Analyzer, I would like have a brief introduction of Codemeter which is the provider of the PL/SQL Analyzer.

CodeMeter's purpose is to collect data (statistics, measures) and aggregate them by means of functions, in order to generate a quality report. CodeMeter was developed by Technology Reply's internal Software Factory as well.

A provider is a module able to take measures on a given piece of source code, written in a specific programming language, given a list of software metrics; providers are Java classes which implements the MeasureProvider interface and implement the necessary logic to collect measures over defined software metrics for that specific language; this way, also third-party analyzers can be used (e.g.: Checkstyle for Java language), just building up a suitable wrapper class which implements the named interface.

CodeMeter receives information from the different providers and applies the configured function logic to all the data at once. Configuration is maintained in a XML file, in which providers, statistics, functions and thresholds are defined; after a measurement session, a new XML file with the same structure and elements is saved, enriched with detailed results; the file can then be transformed in other formats, such as HTML or PDF, to create a human-readable document. For example, in our project, the plug-in of client side is able to produce a HTML report based on the result XML file. And it is done by using XSLT and XPATH.

CodeMeter, by default, has two main providers:

1. Java Provider, based on the open source tool Checkstyle: this is used to collect measures for the Java language; the tool has been marginally adapted in order to implement the MeasureProvider interface and expose some new methods needed for embedding in CodeMeter;
2. Pl/SQL Provider, based on Technology Reply's Pl/SQL Analyzer custom tool: this is used to collect measures for the Oracle Pl/SQL database scripting language.

Pl/SQL Analyzer takes as input a XML configuration file and some database schemas coordinates and produce a textual report containing measures on those database schemas; the output file is used mainly for debug purposes, since Pl/SQL Analyzer is intended for use within CodeMeter as a provider.

Pl/SQL Analyzer core engine is made up of a connector and a number of checks: the connector creates the data sources according to the given input configuration, reads the Pl/SQL source code, and tokenize it in smaller pieces (statements, keywords) in order to model source code's structure; then the connector notifies the available checks with tokens and checks compute the configured metrics, each applying its own check logic (e.g.: number of tables, cyclomatic complexity).

Database schema objects taken into account by the tool are:

- Stored procedures: FUNCTIONs, PROCEDUREs and PACKAGEs
- TABLEs, VIEWs, SEQUENCEs, TRIGGERs, SYNONYMs and DBLINKs.

Currently, only the former objects are considered when looking at Pl/SQL source code, whereas the others are used only for counting purposes.

Out plug-in of Eclipse is based on the above mentioned PL/SQL Analyzer; it takes the result XML with details as an input and extracts the data of XML for displaying.

Have a look of the following figure; this is the user interface of the plug-in before the implementation of logic part. In the forth column which is "Value", the data are all zero because these are the default data.

Project Database: Q:\TRK\database

File

Index	Measure	Description	Value
1	Table Count	Total number of tables is	0
2	View Count	Total number of views is	0
3	Package Count	Total number of packages is	0
4	Procedure Count	Total number of procedures is	0
5	Function Count	Total number of functions is	0
6	NCSS	Total NCSS (packages, procedures, functions)	0
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	0
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	0
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	0
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	0

Figure 4-11 Analysis with default value zeros

And the logic part of our plug-in is to filling out this column with correct numbers. Moreover, it should have one more function of displaying the details of each measurement. As the figure 4.7 shows, when clicking on the fifth measure “function count”, the second table below will show the details of this measure.

As we explained before, after executing the PL/SQL Analyzer, we will have an xml file named cm_codemeter-results.xml which includes all the information of the analysis result. So what we should do is extracting the document objects from the XML file.

To realize this function, two libraries are used in my code: 1) org.w3c.dom 2) javax.xml.parsers. The first one is the API which provides the interfaces for the Document Object Model (DOM) which is a component API of the Java API for XML Processing. And the second one defines the API to obtain DOM Document instances from an XML document. Using this class, an application programmer can obtain a Document from XML.

With these two API we can freely navigate in the XML document and extract all the necessary information as I want.

4.3 Chapter Summary

This chapter mainly explains the design and development process of the PL/SQL analyzer plug-in in the client side Eclipse.

Based on the implemented PL/SQL analyzer, we developed the corresponding plug-in in Eclipse. Firstly, we introduced the Eclipse platform and its composition. Thanks to the flexible structure kernel structure. We can extend Eclipse and enable it more functionalities with Eclipse plug-ins, just like our PL/SQL plug-in. We then illustrated the structure of plug-in of Eclipse and its development steps as well as SWT/JFace which are the most essential tools of plug-in development. In the end, we briefly explained the methods and steps of developing our plug-in.

With this PL/SQL analyzer plug-in installed in the Client side Eclipse, developers are able to monitor the Oracle project developing process at local with higher freedom and faster feedback.

5. Metrics Plug-in design and implementation on the Server-side: Sonar

5.1 Introduction of SonarSource

Sonar is a open source software quality platform. It is a web based application which uses a variety of static code analysis tools such as Clover, FingBugs, PMD, Checkstyle to implement the software metrics extraction. Sonar is designed to improve software code quality in 7 axes: [11]

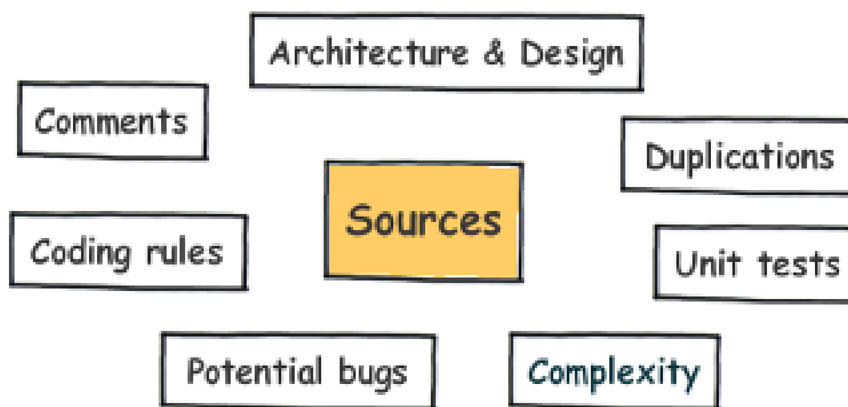


Figure 5-1 Axes of Sonar

Many language are covered by sonar between which java is default built in. There are other commercial plugins which enable sonar to analyze other languages, such as Flex, PHP, Cobol, Visual Basic 6 as well as PL/SQL.

Like the Sonar company's slogan says: Put your technical debt under control. Confess your source code to clean it up. As a continuous inspection engine to manage the technical debt, Sonar is the perfect reporting tool as it is accessible to everybody and centralizes the information through its web server as can be seen as follow:

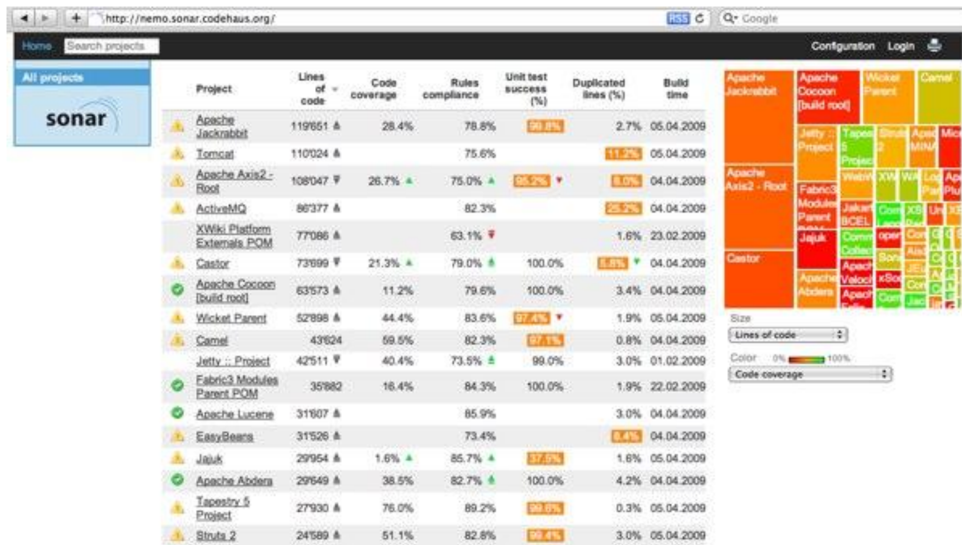


Figure 5-2 The homepage of sonar

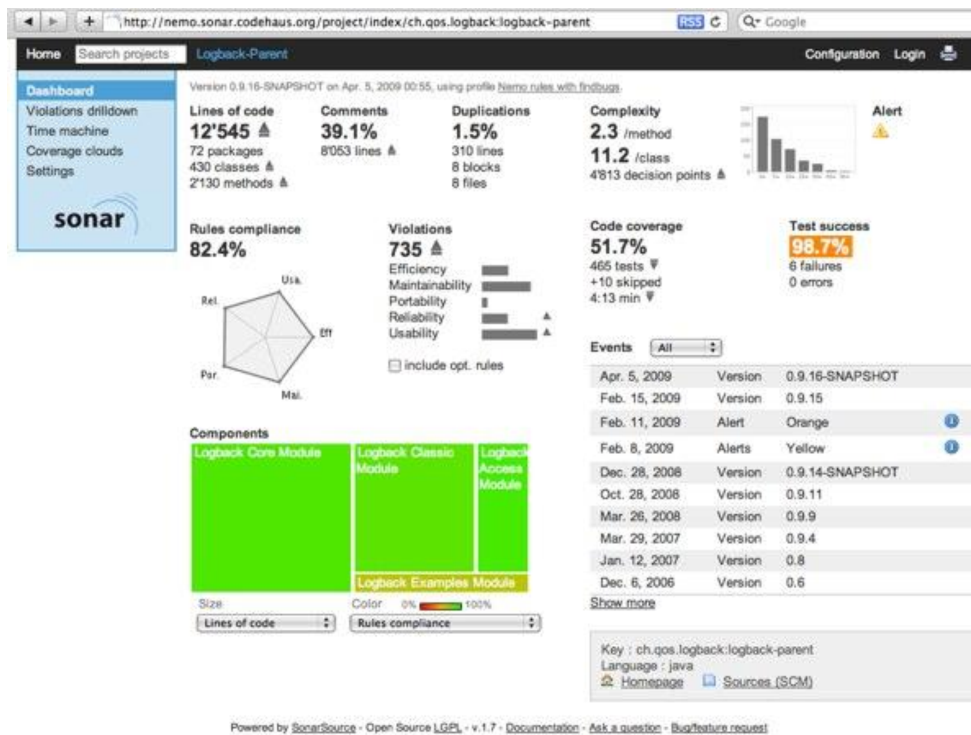


Figure 5-3 The homepage of a specific project

5.2 The structure of SONAR

Basically, Sonar can be categorized into two parts:

1. Server side

As the above figure shows, it works as a web application. It collects the data generated from client part and centralizes all the measurements in one web page.

The main page shows all the information briefly. If necessary, more details of a certain measurement can be seen by clicking it. As the following figure shows, if clicked on the comment lines and navigate to another web page which presents not only the values of comment lines, but also the number of comment lines of each java source package. For example in the package of *it.reply.technology.commons.adudit.error*, there are 30 comment lines and inside this package, there are two .java file, they have 14 and 16 comment lines respectively. And next is the attribute that makes sonar an attractive software quality platform. When clicking the specific java file, it drills down and shows the code java. This figure shows the code of java file *AuditError*.

Figure 5-4 Drilldown to see the code

2. Client side

It analyzes the code and generates the data which will be collected by the server part. Client part is mounted as a plug-in of Maven, which is an open source software build tool. When initializing the quality analysis, the quality information could be acquired by calling the diagnostic tools such as Checkstyle, PMD, Findbus through Maven. And the results would be feedback to sonar server side. Finally, the combination of all the diagnosed quality data are generated and presented. The whole structure is described as the figure 5.5 shows:

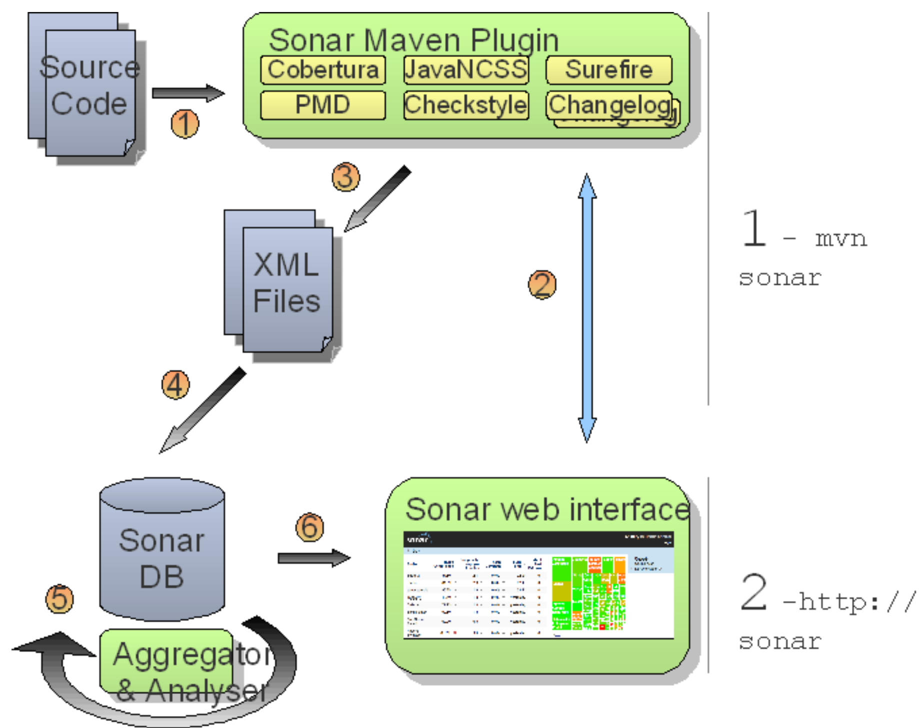


Figure 5-5 Steps of doing a code analysis in sonar

5.3 Maven

5.3.1 Maven Introduction

To have a good understanding and operation of Sonar, it is inevitable to learn about Maven, based one which sonar is run.

The maven, what we are talking about now is referring to Maven 2. Apache describes Maven as: "Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information."

Maven is a popular open source build tool for enterprise Java projects,

designed to take much of the hard work out of the build process. The goal of Maven is to standardize the software build process with all aspects and to provide easy-to-use tool to perform these processes.

5.3.2 Maven POM.xml

Maven uses templates archetypes to define how to build a project and uses an XML file to describe the project. Maven's Project Object Model (POM) file, which is a xml file, declares all the information about the project and configuration details that are need to build a project, such as build directory, source directory, test source directory, project dependencies, the plugins or goals that can be executed, the build profiles. Other information such as the project version, description, developers, mailing lists and such can also be specified. The POM file was named project.xml in Maven 1 and now in Maven 2 it is pom.xml file. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal. A example of the project POM file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>it.reply.technology.sonar</groupId>

<artifactId>sonar-plsql-plugin</artifactId>

<packaging>jar</packaging>

<version>0.1-SNAPSHOT</version>

<name>Sonar PL/SQL plugin</name>

<properties>
```

```
<sonar.plugin.class>it.reply.technology.sonar.plsqlplugin.PlSqlPlugin</
sonar.plugin.class>
        <sonar.version>0.1-SNAPSHOT</sonar.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.codehaus.sonar</groupId>
            <artifactId>sonar-plugin-api</artifactId>
            <version>2.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.5</source>
                <target>1.5</target>
            </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

From preceding xml source illustrating a typical POM file of one Maven project, we can see that the version of this Maven is 4.0.0, and the project belongs to a Italian company Reply Technology from the tag “groupId”. The

"artifactId" defines the unique artifact identification for this project which is sonar-plsql-plugin. The name tag specified the name of this project "Sonar PL/SQL plugin". Between tag properties, there are the classes specified. The dependencies defined the dependency which is the "sonar-plugin-api" with version 2.1. In the build tag, we can see that this project needs a plug-in "maven-compiler-plugin" from the company Apach Maven. Of course we can define as much dependencies and plug-ins as needed all in one POM file.

5.3.3 Using of Maven

One powerful characteristic of Maven is its standardization. Once you have the experience of working in one Maven project, time won't waste much for you to start being familiar with another new one. According to my experience, two attributes of Maven made the standardization possible. The Project builds lifecycle and standard Maven project directory.

When Maven is run, it progresses the first lifecycle phase to the specified one. And all the lifecycle are as follows:

validate

initialize

generate-sources

process-sources

generate-resources

process-resources

compile

process-classes

generate-test-sources

process-test-sources

generate-test-resources

process-test-resources test-compile process-test-classes **test** prepare-package **package** pre-integration-test integration-test post-integration-test verify **install** **deploy**

Table 5-1 Project builds lifecycle

Those phases with bold font are the build phase used frequently in Maven.

- Initialize: Initialize the build state, for example it will create directories.
- Compile: Compile the source code
- Test: Run tests with a certain type of test framework
- Package: Package the code into a distributable form such as Jar or War.
- Install: Install the package into the local repository
- Deploy: Copy the final package to the remote repository for sharing with other developers and projects.

And the Maven standard Maven project directory is another power that

enables Maven to be so popular like it is today. Having a common directory layout would allow users familiar with one Maven project to immediately feel at home in another Maven project.

The following figure 5.6 shows a good example of Maven convention, depicting the folder structure for a standard Maven web application project. There are just two subdirectories of this structure: src and target. The only other directories that would be expected here are metadata like CVS or .svn.

The target directory is used to house all output of the build. The src directory contains all of the source material for building the project, its site and so on. It contains a subdirectory for each type: main for the main build artifact, test for the unit test code and resources, site and so on. [9]

And the first pom.xml file is the Project Object Model (POM) file we discussed above. Without that POM file, the project cannot be a Maven project since no configuration information can be obtained for building the project.

```
|-- pom.xml
|-- target
`-- src
    |-- main
    |   `-- java
    |       |-- org
    |       |   `-- jboss
    |       |       `-- mavenapp
    |       |           `-- App.java
    |       `-- webapp
    |           `-- WEB-INF
    |               `-- web.xml
    `-- test
        `-- java
            `-- org
                `-- jboss
                    `-- mavenapp
                        `-- AppTest.java
```

Figure 5-6 Standard directory of maven project

5.3.4 Using of Sonar based on Maven

If the Maven is well installed in your computer, as well as the Sonar, the Java environment， then the using of Sonar is becoming quite easy.

Every time when you need so start an analysis of your project code with Sonar, what need to be done is initialize the Sonar web services, and the local host http://0.0.0.0:9000/ is ready. Then you launch the Maven goal as “mvn sonar: sonar” under the root directory of the project to be analyzed in the command prompt.

The following image illustrates the initialization of one Sonar analysis of the Maven project “PlSqlSonarPlugIn”. And the root of this project is “C:\Documents and Settings\s.xu\PlSqlSonarPlugIn”.

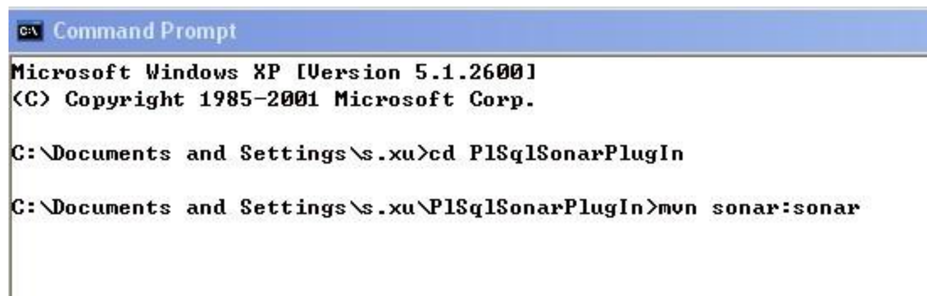


Figure 5-7 Maven command to launch an analysis

If the POM.xml file is well specified, you will get a successful build result as figure 5-.8 shows:

```
Command Prompt
--> DETAILS:

--> FUNC: null
--> FUNC: null
--> FUNC: Overall % of objects with high complexity (21+)
+ saving measures
+ finishing
[INFO]  Sensor it.reply.technology.sonar.plsqlplugin.PlSqlMetricSensor@c33893 do
ne: 1703 ms
[INFO]  Sensor SurefireSensor...
[INFO]  parsing C:\Documents and Settings\s.xu\PlSqlSonarPlugIn\target\surefire-
reports
[INFO]  Sensor SurefireSensor done: 16 ms
[INFO]  Execute decorators...
[INFO]  ANALYSIS SUCCESSFUL, you can browse http://localhost:9000
[INFO]  Database optimization...
[INFO]  Database optimization done: 1000 ms
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 32 seconds
[INFO] Finished at: Fri Nov 05 15:24:20 CET 2010
[INFO] Final Memory: 29M/89M
[INFO] ------------------------------------------------------------------------
C:\Documents and Settings\s.xu\PlSqlSonarPlugIn>
```

Figure 5-8 Successful Build

5.4 Sonar Web Application & ROR

5.4.1 Sonar Plug-in

Sonar is an open source software quality management platform. On this platform, we can build into many other plug-ins to enable Sonar more functional. Amongst all the strengths, the built-in extensibility is really worth mention.

As is known to all, it is the extensibility of software or a tool that makes it widely adopted, such as the Java IDE Eclipse. Sonar itself is a very light core, and we can build into it with the plug-ins which would meet our specific requirements. Everything else in Sonar is a plug-in.

Sonar has an abundant number of plug-ins to fulfill distinctive purposes. These plug-ins functionalities have covered the areas of software metrics, software quality, IDE built-in, additional language analysis, visualization, and Software integration. The details can be seen from the following table [11].

Plug-in Types	Plug-in examples
Additional Metrics	**Artifact Size** - Reports on the size of the artifact generated by projects. **Build Stability** - Reports on stability of project build using Continuous Integration engine data. **Clirr** - Checks Java libraries for binary and source compatibility with older releases. **Security Rules** - Enables to zoom on security rules violations to keep them under control.
Quality Governance	**Quality Index** - Calculates a global Quality Index based on coding rules, Style, Complexity and Coverage by unit tests. **Sqale (Quality Model)** (Commercial) - An implementation of the SQALE Methodology, which supports the evaluation of a software application's source code in the most objective, accurate, reproducible and automated way possible. **Views - Portfolio Management** (Commercial) - Enables aggregation of projects. Projects can be grouped into applications, applications into teams, teams into departments...
IDE plug-in	**Eclipse** - See defects gathered by Sonar directly in Eclipse and fix them on the spot. **IntelliJ IDEA** - See defects gathered by Sonar directly in IntelliJ IDEA and fix them on the spot.

Additional Languages	**C** - The C plugin associated to its set of rules enables to perform objective and automated C code reviews against pre-defined or homemade coding best practices. **Flex / Action Script** - Enables analysis of Action Script projects into Sonar. **PL/SQL** (Commercial) - Enables analysis and reporting on PL/SQL projects. As an option, the plug-in can extract PL/SQL code from Oracle Forms. **Visual Basic 6** (commercial) - Enables to perform objective and automated Visual Basic 6 reviews against coding best practices. **Cobol** (Commercial) - Enables to perform objective and automated Cobol code reviews against pre-defined or homemade coding best practices.
Visualization / Reporting	**PDF Report** - Generates a PDF report with the results of projects analysis. **Radiator** - Displays measures using a big tree map that can then be explored. **Timeline** - Displays measures history using a Google Timeline Chart to replay the past.
Integration	**Bamboo** - Enables to configure and launch Sonar analysis from Bamboo, the Atlassian CI engine. **Hudson** - Enables to configure and launch Sonar analysis from Hudson CI engine.

	Twitter - Creates tweet, when project analyzed by Sonar.

Table 5-2 Sonar Plug-ins

From this table, we can see the great extensibility of Sonar with so many plug-ins.

5.4.2 ROR (Ruby on Rails): the future of web application framework

To develop the plug-in of Sonar, it is necessary to know Ruby and the agile web development new tool ROR, which is Ruby on Rails.

Ruby is a dynamic, reflective, general purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features. Ruby originated in Japan during the mid-1990s and was first developed and designed by Yukihiro "Matz" Matsumoto. It was influenced primarily by Perl, Smalltalk, Eiffel, and Lisp.

Ruby supports multiple programming paradigms, including functional, object oriented, imperative and reflective. It also has a dynamic type system and automatic memory management; it is therefore similar in varying respects to Python, Perl, Lisp, Dylan, Pike, and CLU. [12]

At first glance of ruby, I was thinking, it is nothing, just another type of OOP language like Java. However, as the creator of ruby says: “Ruby is simple in appearance, but is very complex inside, just like our human body, I’m trying to make Ruby natural, not simple.”

The more learn about Ruby, the more I found it attractive, powerful and even complex inside, especially while I am dealing with the ROR. However, while we are talking about Ruby itself, really, it is a absolutely special from any

other existing language. Let me try to define the ‘special’ qualities of Ruby. And I bet this special quality can be found from other single language, but you are never able to find any language can combine these special qualities in one:

1. A terse language

Is it possible to find any other language can be simpler than this to output “hello world” like this?

```
puts "hello"
```

2. Clear

```
until i == arr.length
 puts(arr[i])
  i +=1
end
```

It’s quite easy to understand what this part of code to do.

3. Totally object oriented programming language

Ruby sees everything is an object, even a number. In ruby a simple and basic number has its method.

In many languages, numbers and other primitive types are not objects. Ruby follows the influence of the Smalltalk language by giving methods and instance variables to all of its types. This eases one’s use of Ruby, since rules applying to objects apply to all of Ruby. For example, what do you think the following formula result? `5.*6 =?` 30? Not exactly in ruby, be alert. Because we have this

```
class Numeric
def (x)
  self.+(x)
end
end
y=5.*6
```

So here y equals 11, not 30 since we defined the method *. Inside this method we specified the add operation and it even applies to a number.

4. Ruby's Mixin

Like Java, Ruby features single inheritance only, on purpose. But Ruby knows the concept of modules (called Categories in Objective-C). Modules are collections of methods.

Classes can mixin a module and receive all its methods for free. For example, any class which implements the each method can mixin the Enumerable module, which adds a pile of methods that use each for looping. [13]

```
class MyArray
  include Enumerable
end
```

Ruby has a wealth of other features, among which are the following:

- Ruby has exception handling features, like Java or Python, to make it easy to handle errors.
- Ruby features a true mark-and-sweep garbage collector for all Ruby objects. No need to maintain reference counts in extension libraries. As Matz says, "This is better for your health."
- Writing C extensions in Ruby is easier than in Perl or Python, with a very

elegant API for calling Ruby from C. This includes calls for embedding Ruby in software, for use as a scripting language. A SWIG interface is also available.

- Ruby can load extension libraries dynamically if an OS allows.
- Ruby features OS independent threading. Thus, for all platforms on which Ruby runs, you also have multithreading, regardless of if the OS supports it or not, even on MS-DOS!
- Ruby is highly portable: it is developed mostly on GNU/Linux, but works on many types of UNIX, Mac OS X, Windows 95/98/Me/NT/2000/XP, DOS, BeOS, OS/2, etc. [14]

The web server side of Sonar is totally written in Ruby combined with HTML and JavaScript. While we embedding these three elements into one file, basically this file is the ERB file or embedded ruby. Like many web frameworks, Ruby on Rails uses the Model-View-Controller (MVC) architecture pattern to organize application programming.

Ruby on Rails includes tools that make common development tasks easier "out of the box", such as scaffolding that can automatically construct some of the models and views needed for a basic website. Also included are WEBrick, a simple Ruby web server that is distributed with Ruby, and Rake, a build system, distributed as a gem. Together with Ruby on Rails these tools provide a basic development environment. [17]

5.4.3 MVC Structure: the web design pattern

While doing the web application, it is common to think of an application as having three main layers: presentation (UI), application logic, and resource management. And MVC (Model–View–Controller) pattern, which applies to ROR, isolates the application logic for the user, from the user interface, permitting independent development, testing and maintenance of each.

Model: the model part is in charge of the controlling of data. The model is another name for the application logic layer (sometimes also called the domain layer). It manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

View: The view part is for the presentation only, renders the model into a form suitable for interaction, typically for a user interface element. Sometimes a single model could have several different views for different purpose. While as a view has only one way of rendering. MVC is often seen in web applications where the view is the HTML or XHTML generated by the app.

Controller: Processes and responds to events, typically user actions, and may invoke changes on the model and view. It receives a input from the user and initializes an response and performs an action according to that input.

A simple figure 5-9 as follows makes a well illustration on the coordination of MVC. The solid line represents a direct association, the dashed an indirect association. [18]

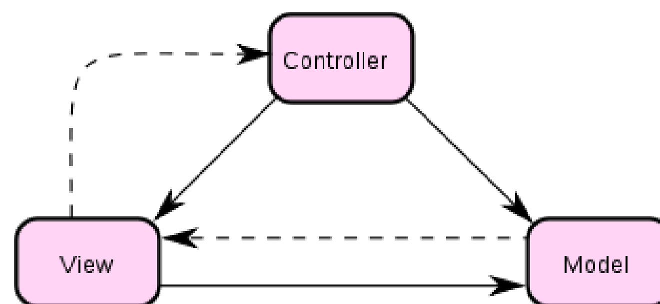


Figure 5-9 MVC schema

What I have covered above is just a general description of a MVC framework. Regarding different programming language, there can be different way of practices of MVC. Different flavors can be .NET, J2EE, and ROR.

Ruby on Roils MVC framework:

- Model (ActiveRecord)

The part defines the entities which play a role in the universe of the application. In Sonar, model part defines the entities such as project, metrics, user, user role, project measure as so on.

This subsystem is implemented in ActiveRecord library. Active Record connects business objects and database tables to create a persistent domain model where logic and data are presented in one wrapping.

- View (ActionView)

It is a presentation of data in a particular format. This part we normally script based system, such as JSP, ASP, PHP. This subsystem is implemented in ActionView library which is an Embedded Ruby (Erb) based system, which we have mentioned before, for defining presentation templates for data presentation.

- Controller (ActionController):

This subsystem is implemented in ActionController. Action Controllers are the core of a web request in Rails. They are made up of one or more actions that are executed on request and then either render a template or redirect to another action. It is a data broker sitting between ActiveRecord (the database interface) and ActionView (the presentation engine).

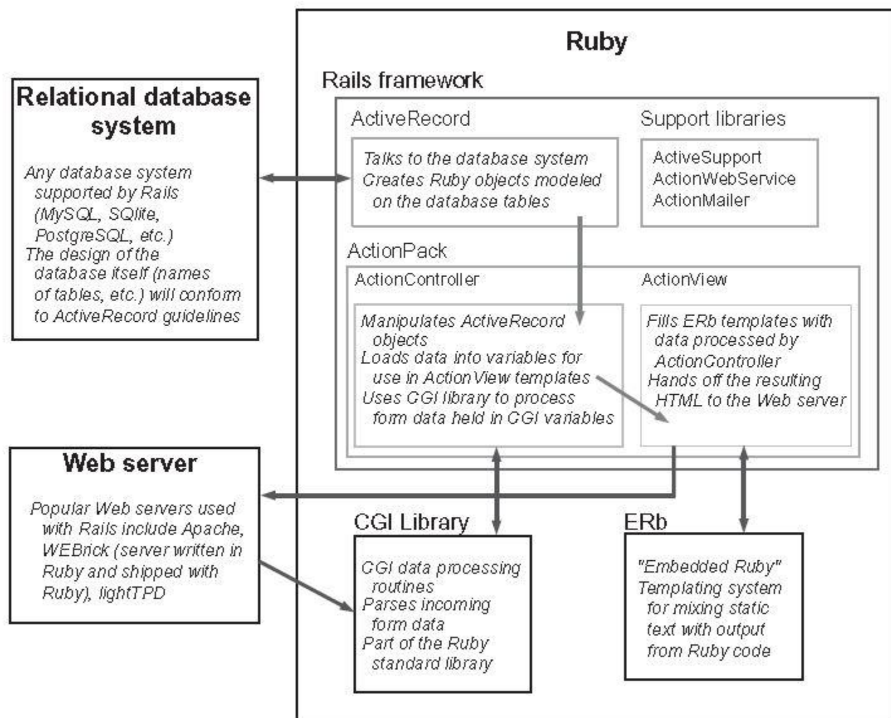


Figure 5-10 Schematic views of Ruby and the Rails framework [19]

5.5 Sonar Plug-in for programming language Pl/SQL

All the theories described above are the basis for developing Sonar Plug-in for the analysis Oracle database programming language.

Given that the customer's requirements specified that they need to have a view of the measurement results of both Java and PL/SQL in one web page, a plug-in which would initialize the analysis, collect the analyzed data and display them on the web page are needed.

5.5.1 Getting started

To develop a plug-in of sonar, the best practice is creating a new Maven project. In the command prompt, we type in the command like this.

```
mvn archetype:generate
```

And by following the hits (See figure 5.11), we will create a Maven project with a POM.xml in which all the need the parameters can be set.

```
Define value for property 'groupId': : it.technology.reply.www
Define value for property 'artifactId': : Sonar-Plugin-Plsql
Define value for property 'version': 1.0-SNAPSHOT:
Define value for property 'package': it.technology.reply.www:
Confirm properties configuration:
groupId: it.technology.reply.www
artifactId: Sonar-Plugin-Plsql
version: 1.0-SNAPSHOT
package: it.technology.reply.www
Y:
[INFO] ------------------------------------------------------------------------
---
[INFO] Using following parameters for creating OldArchetype: maven-archetype-qui
ckstart:1.0
[INFO] ------------------------------------------------------------------------
---
[INFO] Parameter: groupId, Value: it.technology.reply.www
[INFO] Parameter: packageName, Value: it.technology.reply.www
[INFO] Parameter: package, Value: it.technology.reply.www
[INFO] Parameter: artifactId, Value: Sonar-Plugin-Plsql
[INFO] Parameter: basedir, Value: C:\Documents and Settings\s.xu
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] ********************* End of debug info from resources from generated POM
 ***********************
[INFO] OldArchetype created in dir: C:\Documents and Settings\s.xu\Sonar-Plugin-
Plsql
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1 minute 30 seconds
[INFO] Finished at: Sun Nov 07 11:04:55 CET 2010
[INFO] Final Memory: 11M/27M
[INFO] ------------------------------------------------------------------------
C:\Documents and Settings\s.xu>
```

Figure 5-11 Successful creating a Maven project

5.5.2 Code the plug-in in Client side (Java)

After the creation of a simple Maven project, we can start to code this project with Java. Because a Sonar plug-in is a set of Java objects that implement extension points. These extension points are interfaces or abstract classes

which model an aspect of the system and define contracts of what needs to be implemented. They can be for example pages in the web application or sensors generating measures.

The extensions implemented in the plug-in must be declared in a Java class extending org.sonar.api.Plugin. This class must then be declared in the POM with the property <Plugin-Class> :

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <archive>
            <manifestEntries>
                <Plugin-Class>it.reply.technology.sonar.plsqlplugin.PlSqlPlugin</Plugin-Class>
            </manifestEntries>
        </archive>
    </configuration>
</plugin>
```

Figure 5-12 Plug-in-Classes

The highlight line is the class which defines the Plug-in-Class which extends the class org.sonar.api.Plugin.

The dependencies of this plug-in are specified like this.

```
<dependency>
    <groupId>org.codehaus.sonar</groupId>
    <artifactId>sonar-plugin-api</artifactId>
    <version>2.1</version>
</dependency>
<dependency>
    <groupId>it.reply.technology.codemeter</groupId>
    <artifactId>codemeter</artifactId>
    <version>0.1-SNAPSHOT</version>
</dependency>
```

Figure 5-13 Dependencies of sonar plug-in

Then we have defined a simple plug-in which can be installed in the web

server side. One more Erb file (See source code below) needs to be defined which specified how and what to display on the web page. It is like a template or dashboard for displaying the plug-in executing results.

```
<div class="dashbox">

 <h3>Oracle Pl/SQL</h3>

 <p><span class="big"><%= format_measure('tableCount.1',:url =>
url_plsql_drilldown('tableCount.1')) -%></span> tables</p>

 <p><%= format_measure('viewCount.1',:url => url_plsql_drilldown('viewCount.1'))
-%> views</p>

 <p><%= format_measure('packageCount.1',:url =>
url_plsql_drilldown('packageCount.1')) -%> packages</p>

 <p><%= format_measure('procedureCount.1',:url =>
url_plsql_drilldown('procedureCount.1')) -%> procedures</p>

 <p><%= format_measure('functionCount.1',:url =>
url_plsql_drilldown('functionCount.1')) -%> functions</p>

 <p><span class="big"><%= format_measure('NCSS.1',:url =>
url_plsql_drilldown('NCSS.1')) -%></span> lines of code (NCSS)</p>

 <p><%= format_measure('V(G).1',:url => url_plsql_drilldown('V(G).1')) -%> objects
with V(G) between 0 and 10 (good)</p>

 <p><%= format_measure('V(G).2',:url => url_plsql_drilldown('V(G).2')) -%> objects
with V(G) between 11 and 20 (acceptable)</p>

 <p><%= format_measure('V(G).3',:url => url_plsql_drilldown('V(G).3')) -%> objects
with V(G) between 21 and 50 (to be reengineered)</p>

 <p><span class="big"><%= format_measure('V(G).4',:url =>
url_plsql_drilldown('V(G).4')) -%></span> objects with V(G) higher than 50
(untestable)</p>

</div>
```

This table is the dashboard widget Erb file of my Sonar plug-in for PL/SQL. Every time an project analysis is launched, Sonar will load this Erb file and generate a dashboard on the Sonar webpage as is shown below:

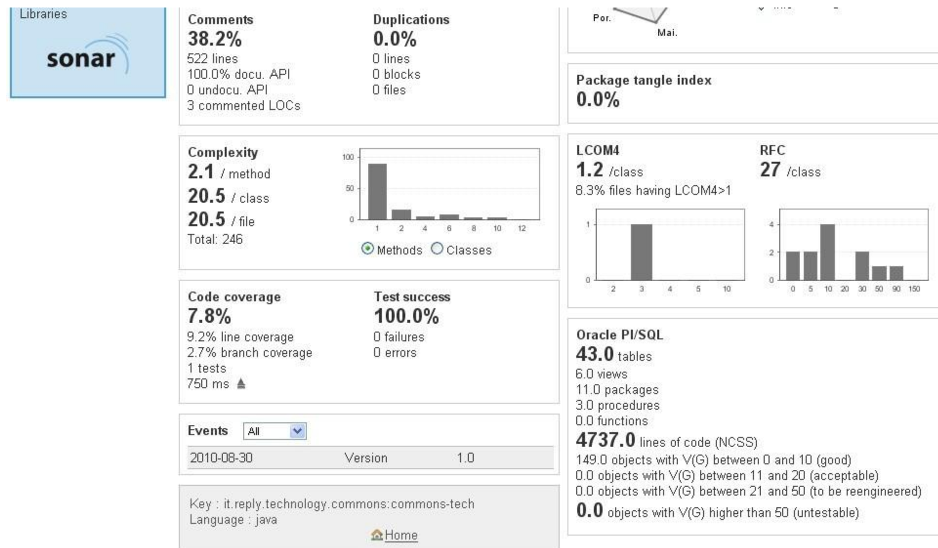


Figure 5-14 Oracle Pl/SQL Plug-in

In the right bottom part of the above image, we see a dashboard named "Oracle PL/SQL". All the information we see here are already defined in the dashboard widget Erb file.

Next I will illustrate how to make Sonar execute the analysis.

Firstly, we have to define the software code metrics which will execute the measures of PL/SQL. These code metrics are that we have chosen before. We defined that class as PlSqlMetrics which should implement the interface of org.sonar.api.measures.Metrics.

Then we need a class which is PlSqlMetricSensor, it implements org.sonar.api.batch.Sensor which would be invoked once during the analysis of a project. The sensor can invoke a maven plugin, parse a flat file, and connect to a web server. Every class implementing org.sonar.api.batch.Sensor should realize the method analyse. As long as an analysis is initialized, this

class would be invoked and the method analyse is called. This is where the execution begins and this is the method which initializes the code analyzer.

And then we should define a class which implements the interface org.sonar.api.Plugin, inside which the name, key, description as well as the list of all the extension points are defined.

```
40      // This is where you're going to declare all your Sonar extensions
41      public List<Class<? extends Extension>> getExtensions() {
42          List<Class<? extends Extension>> list = new ArrayList<Class<? extends Extension>>();
43
44          list.add(PlSqlLanguage.class);
45          list.add(PlSqlSourceImporter.class);
46          list.add(PlSqlCpdMapping.class);
47          list.add(PlSqlFilesDecorator.class);
48          list.add(PlSqlDirectoryDecorator.class);
49          list.add(PlSqlDashboardWidget.class);
50          list.add(PlSqlMetrics.class);
51          list.add(PlSqlMetricSensor.class);
52
53          return list;
54      }
```

Figure 5-15 list of all the extension points

In the end, another very important class need to be defined is PlSqlDashboardWidget which extends the abstract class AbstractDashboardWidget. It is the class which tells Sonar the place where to find the Erb file for displaying.

```
7  public class PlSqlDashboardWidget extends AbstractDashboardWidget {
8
9      protected String getTemplatePath() {
10         return "/it/reply/technology/sonar/plsqlplugin/plsql_dashboard_widget.erb";
11     }
12     public String toString() {
13         return "plsql-dashboard-widget";
14     }
15
16 }
```

Figure 5-16 PL/SQL Dashboard Widget

We can see that class PlSqlDashboardWidget return a path as a string type which specifies the relative path of the Erb file.

All the classes described above are the main composition of Sonar plug-in of

client side which is coded in Java. These classes job is to define the metrics, analyze the project, get the results and pass them to Sonar server part through the Erb file. Apart from these java classes, there are several others which are generated automatically as long as we generate Sonar plug-in and not need to be recoded.

After finishing the coding, we need to install the plug-in. For Sonar, it is very easy step, we just need to use Maven to package the plug-in into a Jar file and deploy it under the directory of "sonar-2.1.2\extensions\plugins" and then restart Sonar server. Once the server is launched, hit the homepage (http://localhost:9000) to see the installed plug-in. But there must be something wrong since the analysis result cannot be transferred to server side yet. And it is the next step to code the server side of Sonar.

5.5.3 Code the plug-in in Server Side (Ruby on Rails)

We have described the ROR above, it is MVC structure based. So let's take a look at the directory structure of the Sonar web application. ROR has a standard directory and this standard directory and file structure (Figure below) is one of the many advantages of Rails; it immediately gets you from zero to a functional (if minimal) application. Moreover, since the structure is common to all Rails apps, as well as Sonar, you can immediately get your bearings when looking at someone else's code. A summary of the default Rails files appears in the following table.

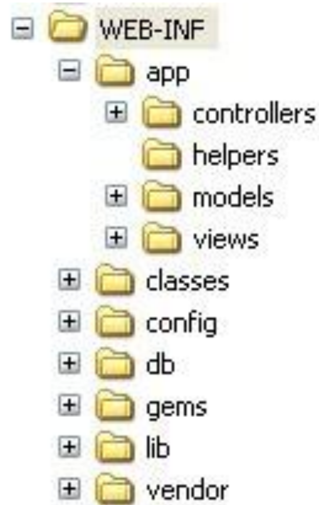


Figure 5-17 Sonar web application standard directory

File/Directory	**Purpose**
App	Core application (app) code, including models, views, controllers, and helpers
Web.xml	It describes how to deploy a web application in a servlet container.
Config	Application configuration. This directory contains the small amount of configuration code that the application will need, including database configuration (in database.yml), Rails environment structure (environment.rb), and routing of incoming web requests (routes.rb).

Db	Files to manipulate the database. Usually, the Rails application will have model objects that access relational database tables. You can manage the relational database with scripts you create and place in this directory.
Gems	Gem requirements for this app
Lib	Library modules
Vendor	Third-party code such as plugins and gems
Rakefile	This file helps with building, packaging and testing the Rails code.

Table 5-3 A summary of the default Sonar Rails directory structure

Our developing task focus on operation in the App file in which there are four folders: controllers, views, models and helpers. We have generally illustrated the working schema of the first three as a MVC structure except the last one helpers. Before introducing the helpers, I would like to mention one thing.

As has been discussed before, views folder have the ERb html or rhtml files which define the display template. All in all, these are just html files and to make the web page dynamic, the ruby codes are embedded. So it's okay to put ruby code in the template, however, it is not advisable to put larger amount of code into the temples.

There are three reasons not to put a bunch of ruby code into the view side. For one thing, the more code you put in the view side of your application, the easier it is to let discipline slip and start adding application-level functionality to the template code. So I dare to ask: what the meaning of MVC. So this is definitely poor form. For another, Rhtml is basically HTML. When you edit it,

you're editing an HTML file. Nowadays, there are companies employ professional designers create the layouts, dealing with the HTML, CSS and so on. Putting a bunch of Ruby code in there just makes them hard to work with. Lastly, it is more difficult to test the code in the views, while as code split into helpers can be isolated and more easily tested. And that's why we have helpers here. A helper is simply a module containing methods that assist a view. Helper methods are output-centric. They exist to generate HTML (or XML, or JavaScript)—a helper extends the behavior of a template. [20]

As we mentioned before, Sonar in the server part may not properly display the analyzing information. This is mainly because in the Erb file of the server side, there is method url_plsql_drilldown (figure below) which the default Sonar does not recognize. It is the method defined by us in the application_helper.rb which is in the helpers module. Defining this method is to make the user be able to drilldown the details of each measurement. For example, we have the analysis result as below, and we need to have a look at detail information of the 4737 NCSS. By clicking the number 4737.0, it navigates to another page which lists all the corresponding details as the next figure shows.

```
193  #   added method for the details of plsql
194  def url_plsql_drilldown(metric_or_measure, options={})
195    if metric_or_measure.is_a? ProjectMeasure
196      metric_key = metric_or_measure.metric.key
197    elsif metric_or_measure.is_a? Metric
198      metric_key = metric_or_measure.key
199    else
200      metric_key = metric_or_measure
201    end
202
203    if options[:resource]
204      url_for(:controller => 'drilldown', :action => 'plsql_details',
205        :id => options[:resource], :metric => metric_key,
206        :highlight => options[:highlight],
207        :viewer_plugin_key => options[:viewer_plugin_key])
208    elsif @project
209      url_for(:controller => 'drilldown', :action => 'plsql_details',
210        :id => @project.id, :metric => metric_key,
211        :highlight => options[:highlight],
212        :viewer_plugin_key => options[:viewer_plugin_key])
213    else
214      ''
215    end
216  end
217
```

Figure 5-18 Url_plsql_drilldown class

Sonar - Sonar PL/SQL plugin

Home | Search | Sonar PL/SQL plugin | Configuration | Log in

Dashboard
Components
Violations drilldown
Time machine
Clouds
Design
Hotspots
Libraries
sonar

Dashboard » Measures drilldown

NCSS

4737.0

Source	Line	Column	Value
PACKAGE schema.BATCHMGR.sql	0	0	491.0
PACKAGE schema.BLK_HUB_PKG.pks	0	0	1357.0
PACKAGE schema.FAILURE.sql	0	0	250.0
PACKAGE schema.HUB_LOG_PKG.pks	0	0	93.0
PACKAGE schema.HUB_LOG_PKG_V2.pks	0	0	397.0
PACKAGE schema.HUB_PKG.pks	0	0	430.0
PACKAGE schema.HUB_PKG_V2.pks	0	0	430.0
PACKAGE schema.PRO_PKG.pks	0	0	462.0
PACKAGE schema.PRO_PKG_V2.pks	0	0	455.0
PACKAGE schema.SMARTCARDMGR.pks	0	0	254.0
PACKAGE schema.USERS.pks	0	0	80.0
PROCEDURE schema.COUNTERFAILEDTRANSACTIONS.prc	0	0	10.0
PROCEDURE schema.COUNTERTRANSACTIONS.prc	0	0	9.0
PROCEDURE schema.SVECCHIA.prc	0	0	19.0

Figure 5-19 NCSS measurement details

The above web page is defined by the plsql_details.html.erb file by us which

is added in the drilldown folder under the views part.

Then you may ask how Sonar gets the information from the server part? It is the controllers; we add one controller class in the file drilldown_controller.rb which is under the controllers module.

```
51  def plsql_details
52    @metric = select_metric(params[:metric], Metric::NCLOC)
53    @highlighted_metric = Metric.by_key(params[:highlight]) || @metric
54    if params[:rids]
55      selected_rids= params[:rids]
56    elsif params[:resource]
57      highlighted_resource=Project.by_key(params[:resource])
58      selected_rids=(highlighted_resource ? [highlighted_resource.id] : [])
59    else
60      selected_rids=[]
61    end
62
63    selected_rids=selected_rids.map{|r|r.to_i}
64    @drilldown = Sonar::Drilldown.new(@project, @metric, selected_rids)
65    @snapshot = @drilldown.snapshot
66    access_denied unless has_role?(:user, @snapshot)
67
68    @highlighted_resource=@drilldown.highlighted_resource
69    if @highlighted_resource.nil? && @drilldown.columns.empty?
70      @highlighted_resource=@project
71    end
72  end
```

Figure 5-20 Added method plsql_details

5.5.4 Installation

In the end, we install the plug-in into Sonar by simply putting it under the directory of “sonar-2.1.2\extensions\plug-ins” and then restart Sonar server. There is one thing to take care is that it is necessary to put all the dependencies of this plug-in into the directory as well. Otherwise, the plug-in could not work.

5.6 Chapter Summary

In the chapter, we firstly introduced what is Sonar. We explained the structure and composition of Sonar as multi code examination open source software. And then we introduced Maven 2 based on which Sonar is run as well as we explained its characteristics and usage. Afterward, the structure of Sonar plug-in is explained in detail which is very important to know to develop a plug-in of Sonar.

Ruby on Rails which is normally call ROR is the key technology for developing Sonar plug-in. It is explained as well. Module-View-Controller (MVC) structure based on which ROR is organized is also covered as an important aspect of my project. Furthermore we did some customer analysis explaining why we are determined to develop Sonar Pl/SQL plug-in by ourselves instead of purchasing the commercial PL/SQL from Sonar Company. At last, we showed how we have done the design and development of this plug-in from the client side Eclipse to server side Sonar.

6. Experimental Results

In this chapter, we will show the tests results of the PL/SQL plug-in both on client side Eclipse and the server side Sonar. And analysis will be done based on these test results.

6.1 Plug-in test results and analysis on Eclipse

Plug-in one client server will be executed firstly to launch a new analysis by choosing one project which should have Oracle database. See figure 6.1, 6.2.

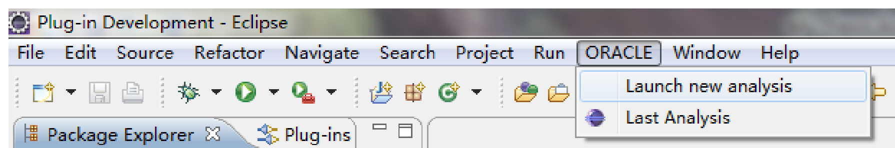


Figure 6-1 Launch new analyses

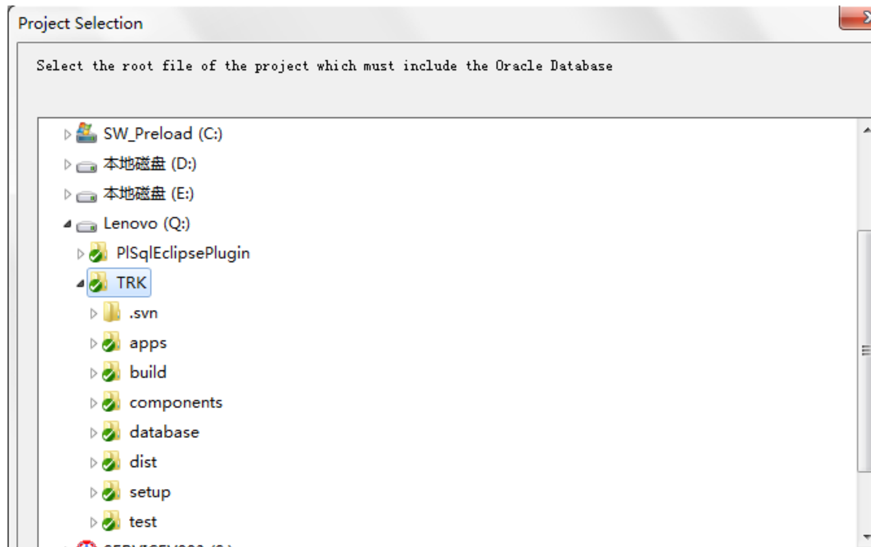


Figure 6-2 Oracle Project Selections

After clicking and confirming, analyses on project "TRK" will be executed. Actually the plug-in is doing the checking of the code under the "database" directory. And we will have a user interface as figure 6-3

Project Database: Q:\TRK\database

File

Index	Measure	Description	Value
1	Table Count	Total number of tables is	75
2	View Count	Total number of views is	0
3	Package Count	Total number of packages is	4
4	Procedure Count	Total number of procedures is	2
5	Function Count	Total number of functions is	33
6	NCSS	Total NCSS (packages, procedures, functions)	7803
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	55
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	3
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	6
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	2

Source	Line	Column	Value
FUNCTION trk.AGGIORNA_OPERAZIONE.sql	0	0	1.0
FUNCTION trk.ALL_SC_BOX_IN_MORE_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_BOX_IN_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_BOX_NOT_INSTATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_CONF_IN_MORE_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_CONF_IN_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_CONF_NOT_INSTATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_SCATOLE_IN_MORE_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_SCATOLE_IN_STATE.sql	0	0	1.0
FUNCTION trk.ALL_SC_SCATOLE_NOT_INSTATE.sql	0	0	1.0

Figure 6-3 Analyses results

After finishing this analysis, we could execute another analysis based on this user interface by clicking the menu "File" and menu item "Select Project", or just using shortcut key "Ctrl+S". See figure 6-4

File

Select Project Ctrl+S
Create Html Report Ctrl+H
Exit

		Description	Value
		Total number of tables is	76
		Total number of views is	0
3	Package Count	Total number of packages is	5
4	Procedure Count	Total number of procedures is	2
5	Function Count	Total number of functions is	33
6	NCSS	Total NCSS (packages, procedures, functions)	8003
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	58
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	3
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	6
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	2

Figure 6-4 Launch another new analyses

After launching another new analysis, the next moves are the same from figure 6-1. Besides, there are another two menu items "Create Html Report", "Exit"

Menu item "Create Html Report" will generate a HTML report (Figure 6-5) at local computer in order to make the customer have more freedom to make a record of the project development process and the code changing status. You can choose the directory under which to create the HTML report. After creating the report, the location of the report will be informed. While as menu "Exit" is simply used for exit the analysis and closes the table.

6-5

Project: Test

Software Metrics

Oracle RDBMS

Metric/Statistics	*Allowed values*	*Current value*	*Options*
Table Count			
Total number of tables is		76	[See details]
View Count			
Total number of views is		0	
Package Count			
Total number of packages is		5	[See details]
Procedure Count			
Total number of procedures is		2	[Hide details]

Source	Ln	Cl	Val
PROCEDURE trk.BLOCCO_VOUCHER.sql	0	0	1.0
PROCEDURE trk.recuperoSc.sql	0	0	1.0

Metric/Statistics	*Allowed values*	*Current value*	*Options*
Function Count			
Total number of functions is		33	[See details]
NCSS			
Total NCSS (packages, procedures, functions)		8003	[See details]
Cyclomatic Complexity V(G)			
N. of objects with V(G) between 0 and 10 (good)		58	[See details]
N. of objects with V(G) between 11 and 20 (acceptable)		3	[Hide details]

Source	Ln	Cl	Val
PROCEDURE ALIGNMENT.VOUCHER_CARNET_MANAGEMENT	23413	40	19.0
FUNCTION TSC.ASSEGNA_CONFEZIONATORE_BOX	28834	22	12.0
FUNCTION REGISTRA_ESITO_OPERAZIONE	33813	16	14.0

Metric/Statistics	*Allowed values*	*Current value*	*Options*
N. of objects with V(G) between 21 and 50 (to be reengineered)		6	[See details]
N. of objects with V(G) higher than 50 (untestable)	0.0 (max)	2	[See details]
Overall % of objects with high complexity (21+)		11.59%	

Figure 6.5 HTML Report

The HTML report generated locally can be open using any web browser such as IE7 or Firefox. The above figure is the report opened by Firefox. We can see that the analysis data are integrated into one web page. All the information are included. You can click on “See detail” to drilldown the details of the specific measure and then click on “Hide detail” to close the detail drilldown. This is the realized by using XSLT and XPATH techs to navigate in the XML files and show them in a web page.

If you remember well, at the beginning of the analysis, there is another menu item under the menu of “ORACLE” which is “last analysis” (See figure 6-1). It is used to show the previous analyzed project result.

The last test I will show is the parallel comparison of two results which is a good characteristic of our plug-in. (See figure 6-6)

Project Database: Q:\TRK\database

File

Index	Measure	Description	Value
1	Table Count	Total number of tables is	76
2	View Count	Total number of views is	0
3	Package Count	Total number of packages is	5
4	Procedure Count	Total number of procedures is	2
5	Function Count	Total number of functions is	33
6	NCSS	Total NCSS (packages, procedures, functions)	8003
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	58
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	3
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	6
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	2

Project Database: Q:\PlSqlEclipsePlugin\database

File

Index	Measure	Description	Value
1	Table Count	Total number of tables is	76
2	View Count	Total number of views is	0
3	Package Count	Total number of packages is	5
4	Procedure Count	Total number of procedures is	0
5	Function Count	Total number of functions is	33
6	NCSS	Total NCSS (packages, procedures, functions)	7894
7	Cyclomatic Complexity V(G)	N. of objects with V(G) between 0 and 10 (good)	56
8	Cyclomatic Complexity V(G)	N. of objects with V(G) between 11 and 20 (acceptable)	3
9	Cyclomatic Complexity V(G)	N. of objects with V(G) between 21 and 50 (to be reengineered)	6
10	Cyclomatic Complexity V(G)	N. of objects with V(G) higher than 50 (untestable)	2

Figure 6-6 Comparison of two projects

By performing parallel comparison of two projects or comparison of the project in different time instance, developers are able to see the changes of the software development process more clearly. That's the purpose we implement this function.

Up to now, we have finished the functional test of plug-in in client side Eclipse, and it works well regarding the functions which have been implemented. However, we don't think it is enough. As you can see that there are only 10 metrics employed to do code analysis, and they are mainly focus on the program size and software cyclomatic complexity.

So for plug-in on Eclipse, we hope to add more metrics to enrich the analysis capability of our PL/SQL plug-in. And this expectation will be covered in 7th chapter as a conclusion.

6.2 Plug-in test results and analysis on Sonar

To test the plug-in on Sonar, we should firstly open the local web service of Sonar. And then we begin to analyze an oracle project.

As long as the http://localhost:9000 port is ready, we chose one project and launched the Sonar analysis in the command prompt by executing mvn sonar: sonar. (Figure 6-7) If it successes in executing the analysis, we will see the information in command prompt as figure 6-8 shows.

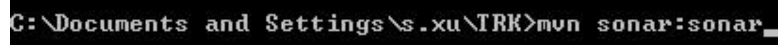


Figure 6-7 Launch a Sonar analysis

```
+ saving measures
+ finishing
[INFO] Sensor it.reply.technology.sonar.plsqlplugin.PlSqlMetricSensor@f7cbd1 do
ne: 2516 ms
[INFO] Execute decorators...
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000
[INFO] Database optimization...
[INFO] Database optimization done: 375 ms
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 11 seconds
[INFO] Finished at: Mon Nov 15 19:03:30 CET 2010
[INFO] Final Memory: 12M/37M
[INFO] ------------------------------------------------------------------------
C:\Documents and Settings\s.xu\TRK>
```

Figure 6-8 Successful analysis

Then hitting the http://localhost:9000 we see the Sonar PL/SQL plug-in row which is the project we just analyzed. Then clicking the project analyzed, we will navigate to the index page of that project. We have both the Java analysis result and PL/SQL analysis result if the project has both JAVA source code and PL/SQL code. However, our TRK project has only PL/SQL source code and we will have the result as figure 6-9 shows. And figure 6.10 shows the analysis result of a project Codemeter has both JAVA and PL/SQL.

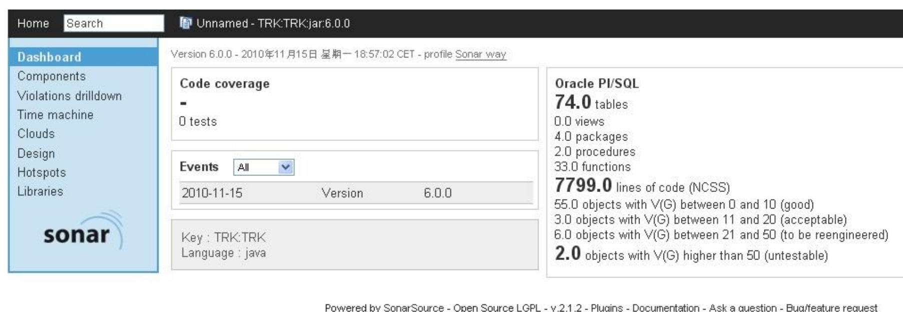


Figure 6-9 A project only has PL/SQL

To test the details display function, we click the NCSS of project Codemeter to see the details of NCSS and we have the result as figure 6-11 shows. From the result, we can see the source file name of the PL/SQL as well as the corresponding value which stands for the number of NCSS. Another test on cyclomatic complexity between 0 to 10 is shown in figure 6-12, from which

we see the source name, line, column and corresponding V(G) value. Since the limitation of page size, we cannot display all the details of V(G).

One thing we should mention here is that in figure 6-11, the “Line” and “Column” are all zeros instead in figure 6-12 under cyclomatic complexity situation, we have non-zero values.

Actually, "line, column" they are the coordinates at which the measure is taken.

In case they are 0, 0 it means the whole piece of code (function, procedure, table, etc.) is taken for the measurement. There are two situations here:

1. In the case of "V(G)", single functions/procedures are considered for the measure: in this case, "line, column" are the coordinates of the function/procedure inside a package; in case of single function/procedure, they should be 0,0
2. In the case of "number of tables", a single table is counted as a whole, so 0, 0 is taken (meaning "start of the CREATE TABLE script"). The same situation for NCSS. In NCSS, the single function/procedure is measured as a whole.

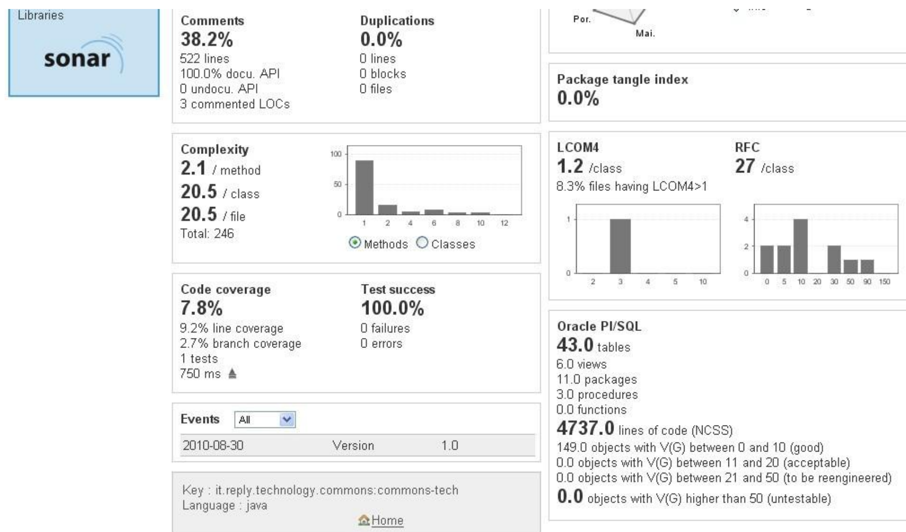


Figure 6-10 A project has both JAVA and PL/SQL

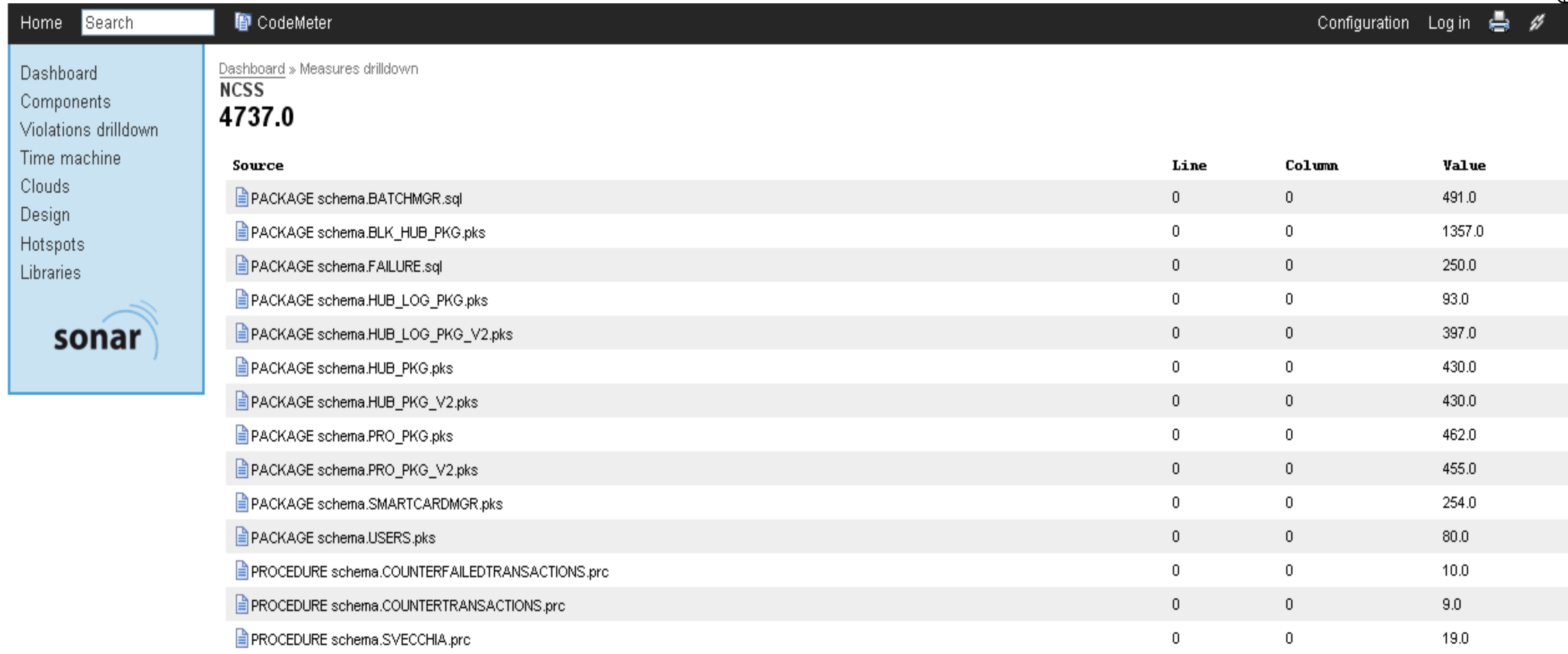
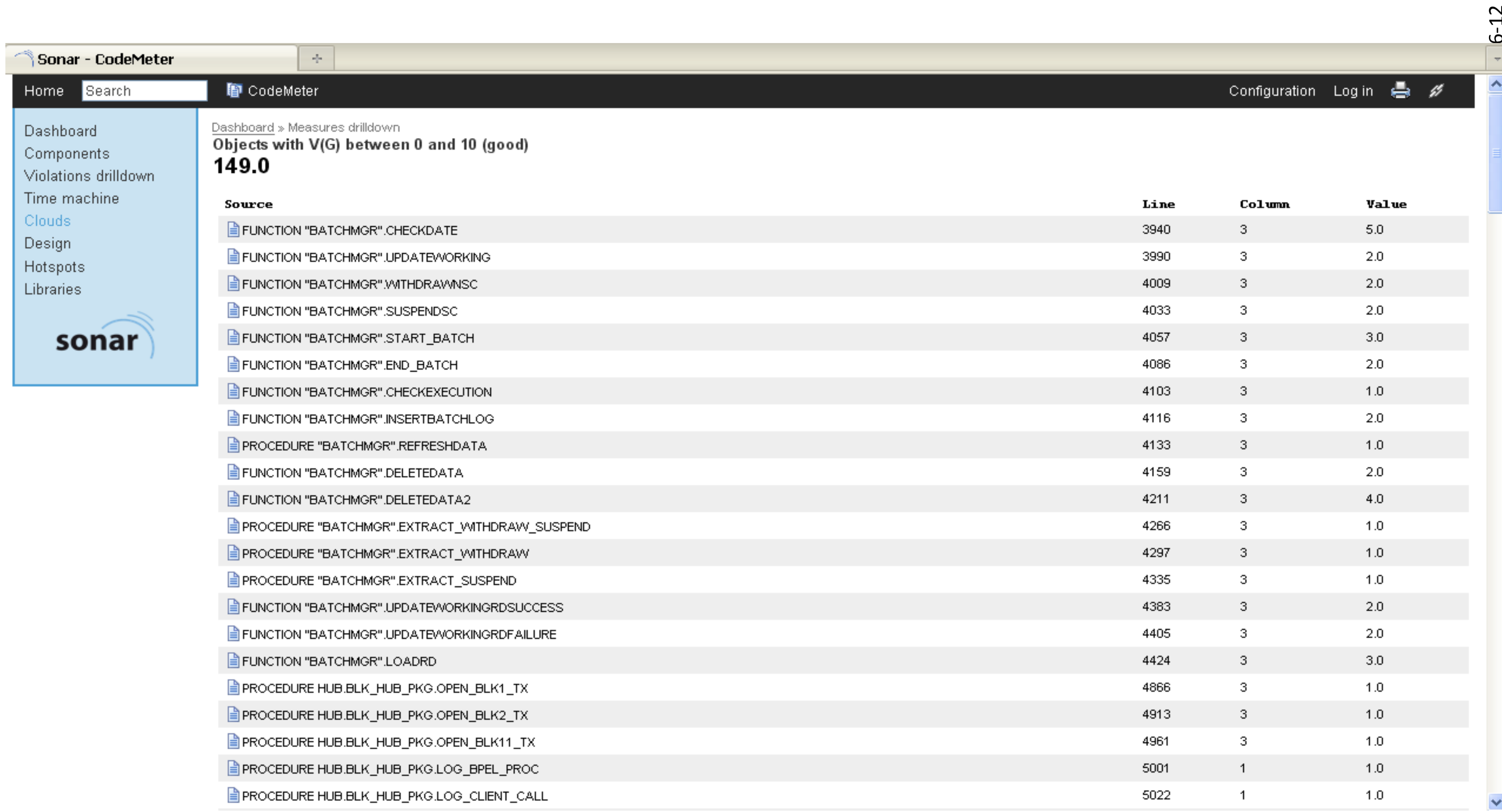


Figure 6.11 NCSS details



Source	Line	Column	Value
FUNCTION "BATCHMGR".CHECKDATE	3940	3	5.0
FUNCTION "BATCHMGR".UPDATEWORKING	3990	3	2.0
FUNCTION "BATCHMGR".WITHDRAWNSC	4009	3	2.0
FUNCTION "BATCHMGR".SUSPENDSC	4033	3	2.0
FUNCTION "BATCHMGR".START_BATCH	4057	3	3.0
FUNCTION "BATCHMGR".END_BATCH	4086	3	2.0
FUNCTION "BATCHMGR".CHECKEXECUTION	4103	3	1.0
FUNCTION "BATCHMGR".INSERTBATCHLOG	4116	3	2.0
PROCEDURE "BATCHMGR".REFRESHDATA	4133	3	1.0
FUNCTION "BATCHMGR".DELETEDATA	4159	3	2.0
FUNCTION "BATCHMGR".DELETEDATA2	4211	3	4.0
PROCEDURE "BATCHMGR".EXTRACT_WITHDRAW_SUSPEND	4266	3	1.0
PROCEDURE "BATCHMGR".EXTRACT_WITHDRAW	4297	3	1.0
PROCEDURE "BATCHMGR".EXTRACT_SUSPEND	4335	3	1.0
FUNCTION "BATCHMGR".UPDATEWORKINGRDSUCCESS	4383	3	2.0
FUNCTION "BATCHMGR".UPDATEWORKINGRDFAILURE	4405	3	2.0
FUNCTION "BATCHMGR".LOADRD	4424	3	3.0
PROCEDURE HUB.BLK_HUB_PKG.OPEN_BLK1_TX	4866	3	1.0
PROCEDURE HUB.BLK_HUB_PKG.OPEN_BLK2_TX	4913	3	1.0
PROCEDURE HUB.BLK_HUB_PKG.OPEN_BLK11_TX	4961	3	1.0
PROCEDURE HUB.BLK_HUB_PKG.LOG_BPEL_PROC	5001	1	1.0
PROCEDURE HUB.BLK_HUB_PKG.LOG_CLIENT_CALL	5022	1	1.0

Figure 6.12 V(G) < 10 details

6.3 Chapter summary

In this chapter, we mainly presented the test results of the plug-in both in the client side and server side as well as some analysis on those results. Based on these results, we also pointed out some pros and cons of the plug-in.

7. Conclusion

This chapter is organized as follows. Section 7.1 below summarizes the contributions made by my work and thesis. Section 7.2 outlines the lessons learnt from this project. Section 7.3 gives a future expectation of my work that has been done so far regarding on two aspects, the plug-in core PL/SQL analyzer and the server side Sonar.

7.1 Thesis contribution

My thesis proposed one idea that under the environment of continuous integration and software or code metrics, it is possible to achieve a further quality improvement of software development product and process.

Continuous integration has been already employed in software development process especially for open source software development; and studies have proven its positive impact on the software process quality and the product. By adopting a higher frequency of checking in the code, building the change, detecting the potential problems and reporting the failure, developers obtain a better control of the process of the development as well as the current situation.

And this can be further enhanced by employing the software/code quality metrics. To have a monitor on the quantitative analysis of the current software/code, in particularly, by making the analysis programming language oriented, developers are able to gain a software trend, which stands for where is development is going, better or worse. And proper actions can be taken according to the analysis. For instance, if after the code analysis, developers found that the cyclomatic complexity of a certain procedure is too high like 89, which is meaning a bad maintainability, they could go and check that procedure to fix the problem.

Moreover our work, the PL/SQL plug-in which is working well both on the client side Eclipse and server side Sonar made the above theory in real partially; and future works need to be carried on to this plug-in.

7.2 Lessons learnt

After a half year of working on this project in Reply Technology, which is a famous Italian software consultancy company originated from Turin and a global silver partner of Oracle, I have learnt a lot of things.

The first obstacle of working on this project is to get familiar with all of these concepts that I had not touched before in university. I should say that as a fresh man, new graduate, everything is new. To initialize my work, reading a lot of materials and papers are necessary. Even thought plenty of theories have been learnt in university, it does not necessary mean I have already good command on them. Because in industry application, solo understating of theory means nothing if you don't know how to apply them into practices. It is the know-how that rocks.

Regarding continuous integration, a number of theories can be related, such as software development process or life cycle, agile developing mode, XP (extreme programming) practice, and so forth. For software quality and software/code metrics, there are even more theories can be learnt. Thanks to the cooperation of my colleague Rosin Massimo, who is also my thesis supervisor in the company, this implementation part of work is done very well by him. And I am in charge of partial development of the PL/SQL core analyzer as well as realizing the function of PL/SQL analysis for Eclipse and Sonar based on that core analyzer.

Maven, Hudson, Ant, Sonar, Eclipse, Subversion, and so forth are those tools that essential to the project development. After these days of working with these tools, I have already obtained a much better command on them.

Regarding the technologies that have been using in this project, most of them have been explained in this dissertation, nevertheless, there are much more. The most important techs that I have learnt are Ruby, Ruby on Rails, XSLT, XPATH, SWT/JFace, Eclipse plug-in structure, Java. Although java is learnt in university, it is definitely different respect to its usage in industry purpose. In industry companies, J2SE (Java 2 standard edition) is not enough; J2EE enjoys a much wider and broader application.

7.3 Future research directions

There are two further directions that we will carry on:

1. Add more metrics, after comparison with other plug-ins and investigating the existing metrics which best fit in Pl/SQL.
   - Halstead complexity: we have discussed this type of metrics in chapter 3. Although many critics have been made on it, still at least two of the Halstead metrics worthy of implementation, Program length and Halstead volume.
   - DAC among packages: in a clean and correct construction of Pl/SQL code, all stored procedures are grouped into packages; if a package is considered as a "class" (in an OOP sense); we could compute DAC for a package and analyze how a package is coupled with others.
   - More Oracle-specific size metrics, e.g.: number of indexes, number of triggers.
   - More Oracle-specific complexity metrics, e.g.: cyclomatic complexity of triggers.

2. Add a parser and a lexicographical analyzer to the plug-in core: language syntax check and modeling of the source code as an AST (Abstract Syntax Tree) would improve the source tree reconstruction, thus increasing the confidence in the interpretation of statements, and also would allow to write more readable checks; tools like ANTLR (www.antlr.org) could be integrated to perform such tasks.

8. Bibliography

[1] Software configuration Management Best practices for continuous integration, Accurev.

[2] Tijs van der Storm, Backtracking Incremental Continuous Integration

[3] The Art of Software Architecture: Design Methods and Techniques, Stephen T. Albin, 2003

[4] Software Quality: Definitions and Strategic Issues, Ronan Fitzpatrick

[5] Software Metrics SEI Curriculum Module SEI-CM-12-1.1, Everald E. Mills, Seattle University, 1988

[6] http://en.wikipedia.org/wiki/Function_point

[7] Rich Sharpe. "McCabe Cyclomatic Complexity: the proof in the pudding"

[8] http://en.wikipedia.org/wiki/Cyclomatic_complexity

[9] Apache Maven Current version User Guide, The Apache Software Foundation 2010-09-05

[10] Software metrics an overview, Simon Alexandre, University of Namur, Software Quality Lab ,Belgium, July 2002

[11] http://www.sonarsource.org/

[12] http://en.wikipedia.org/wiki/Ruby_%28programming_language%29

[13] http://www.ruby-lang.org/en/

[14] Matz, in Blocks and Closures in Ruby, December 22nd, 2003.

[15] Matz, speaking on the Ruby-Talk mailing list, May 12th, 2000.

[16] Matz, in An Interview with the Creator of Ruby, Nov. 29th, 2001.

[17] http://en.wikipedia.org/wiki/Ruby_on_Rails

[18] http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller

[19] David A.Black, Ruby for Rails

[20] Dave Thomas and David Heinemeier Hansson, with Leon Breedt, Mike Clark, James Duncan Davidson, Justin Gehtland, and Andreas Schwarz, Agile web development with Rails.

[21] http://en.wikipedia.org/wiki/Eclipse_%28software%29

[22] Eric Clayberg, Dan Rubel, Eclipse building commercial quality plug-ins

[23] http://www.ibm.com/developerworks/opensource/library/os-ecov/

[24] http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html

[25] http://en.wikipedia.org/wiki/Continuous_integration
http://en.wikipedia.org/wiki/Software_development_process

[27] John Ferguson Smart, upping the game improving your software development process.

[28] http://www.aivosto.com/project/help/pm-loc.html

[29] http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

[30] Jon Bowyer, Janet Hughes, Assessing Undergraduate Experience of Continuous Integration and Test-Driven Development

[31] Tijs van der Storm, Backtracking Incremental Continuous Integration

[32] Jesper holck, CONTINUOUS INTEGRATION AND QUALITY ASSURANCE: A CASE STUDY OF TWO OPEN SOURCE PROJECTS, 2004

[33] Amit Deshpande, Dirk Riehle, Continuous Integration in Open Source Software Development

[34] Dr.B.R Sastry, M.V.Vijaya Saradhi, Impact of Software Metrics on Object-Oriented Software Development Life Cycle, 2010

[35] Tabinda Aftab, Improved Software Quality with Agile Process

[36] Cem Kaner, Walter P. Bond, Software Engineering Metrics: What Do They Measure and How Do We Know?

[37] Simon Alexander, Software Metrics An Overview Version 1.0, July 2002

[38] Wakalio Consulting, Java Software Quality Tools and techniques, 2008

[39] Thomas Haug, Using software metrics to detect refactorings, 2009

[40] Georg Fleischer, Continuous Integration: what companies expect and solutions provide, 2009

[41] http://codedcomplex.com/2010/05/the-importance-of-continuous-integration-for-software-development/

[42] http://community.serena.com/posts/775495a210

[43] http://checkstyle.sourceforge.net/

[44] http://www.conquestsoftwaresolutions.com/page/clearsql_at_a_glance