Quality Assurance

Ammar Joukhadar
Mhd. Abou Zliekha
Samer Al-Moubayed
Bashar Awad

Information Technology Faculty, Damascus University
ammarj@scs-net.org

Abstract

E-business systems involve a large number of non functional requirements, such as performance, distribution, reliability, security, scalability, auditing, fail over, fault tolerance, resuming, clustering/load balancing, portability, accessibility, usability, etc.

Such requirements increase the complexity of the system, and depending on programmers' experience this may reduce dramatically code quality.

Different techniques can be applied to ameliorate code quality; this includes applying some standard design pattern, decomposing code into a set of modules, applying AOP techniques, using generic programming techniques, or using code generators.

All of these techniques are very important and useful, but are not sufficient. We still need to write a minimum of manual code, so human resources remain the major factor that affects code quality.

This paper addresses the problem concerning the quality of manual code. What is code quality? What is the effect of a bad code quality on the program lifecycle, and how to assure this quality?

We propose a tool that allows analyzing code in order to satisfy a set of quality directions. These directions can be added dynamically with no need to modify the code.

1. Introduction

1.1 Why we need to assure quality of software

As software development is a manual work, most of the mistakes are caused by the programmers:

- Programmers may get bored or tired. This will affect their programming abilities, so the code quality will vary consequently according to the programmer.
- Junior programmers in companies will take a lot of time to reach the company's standards, and best practices. During this time they will cost the company a lot of time and effort to manually revise their code. So signor programmers have to verify the code of junior programmers, and all resources will be busy.

This is why we need a quality assurance tools to automatically verify and assure the code quality regardless of the programmer's ability.

1.2 What is the Quality of software?

Quality of software covers all non functional features of the software such as reliability, security, robustness, performance, memory management, scalability, usability, platform independence, etc.

Distributed applications need also other non functional requirement such as distribution, clustering, non concurrent accesses, etc.

There are many tools that allow us to check programs in order to detect quality problems and fix theme. Some of theme bench an application to measure its performance such as OpenSTA[1] (http://www.opensta.com/) for web based applications. Jbuilder/OptimizeIt Software allows detecting performance bottlenecks and tuning applications. Rational software (http://www.rational.com) allows testers to automatically record the workload of 1 to 1000's of virtual user sessions, create test scripts, execute test sessions, and evaluate the summarized reports and graphs. Beirekdar[2] offers a scalable way to verify ergonomic quality for web applications.

All these methods are very useful and allow detecting a wide range of general quality defects at compile time or at run time. Our approach consists of dividing quality check procedure into three phases:

1) Analysis and design time: using a framework, such as Spring [3] [4], or Struts [5] [6] we can reduce code size, make use of generic code, and make use of some design pattern templates [7] [8]. This phase will limit the number of possible code quality degradation.
2) Compile time: we have built our own tools: (1) an automatic verification tool which allow us to check manually added code against a set of directions, and (2) a specification tool which allows us to introduce new framework specific directions.
3) Compile or Run time: at this step market tools will be efficient to detect general code quality defects.

In this article we will focus on the second phase: a code analyzer (verification) tool that reads the source code and finds the quality errors within the code depending on a quality rules that can be customized by the user using the specification tools.

2. Context and overview of the system

A technical framework is a skeleton of semi finished application. This application is based on some predefined abstract design patterns. A user or a code generator generates a code according to these patterns. The programmer needs to fill in a kind of template classes in order to complete the program.

In such a context we are not only interested by general quality direction but also by specific ones (ie. we need to be sure that not only Java rules are satisfied but also framework ones). The user needs to add new directions whenever his framework and best practices evolve.

2.1. Java Rules

Java quality rules are a set of rules for common programming in Java; these rules are written by Sun or have been collected by the experiences of java programmers (see http://www.javapractices.com).

Such rules are important to assure the performance and the understandability of the code. They allow us to minimize the bugs at the testing phase as much as possible.

Examples of such rules:

Category	Code Convention
Rule Name	Avoid changing condition variable in loop body
Severity	MEDIUM
Description	
Reason	Control variable should not be changed inside the FOR loop body.
Example	Violation

	Class Test {
	    void method() {
	      for(int i=0; i < 100;i++) {
	         i *= 2;
	      }
	    }
	}

	Correction

	Class Test {
	    void method() {
	        for(int i=0; i < 100; i=i*2+1) {
	        }
	      }
	    }

Reference	

Category	Optimization
Rule Name	Avoid Synchronized blocks
Severity	CRITICAL
Description	Avoid synchronized methods as much as you can. If you cannot, synchronize on methods rather than on code blocks.
Reason	…..
Example	Violation
	…..
	Correction
	….
Reference	http://www-2.cs.cmu.edu/~jch/java/speed.html

2.2. Framework rules

A framework (Spring [3] [4] or Struts [5] [6]) aims to satisfy a set of design patterns such as MVC (Model View Controller) [9][10], AOP (Aspect Oriented Programming) [11][12], IoC (Inversion Of Control) [13][14], Transparent persistence, etc.

It is important that the manually added code satisfies also these patterns.

Examples of framework rules:

Category	AOP
Rule Name	Tracing
Severity	High
Description	Never use tracing inside business methods they will be added by the AOP engine
Reason	This will complicate the maintenance and the delivery if we change tracing configuration or tracing module.
Example	Violation

	Class Test {
	  void f() {
	    Trace.in(“f”);
	    <body>
	   Trace.out(“f”)
	}

	Correction

	Class Test {
	  void f() {
	    Trace.in(“f”);
	    <body>

	Trace.out("f")
	}

Reference	

Category	Transparent persistence
Rule Name	Isolation
Severity:	High
Description	Usage of D.B. connection is not allowed
Reason:	Direct access to DB by different programmer may influence the performance of the, may cause concurrent access and may affect the readability of the program.

Example	Violation
	void update(object) {
	Connection=getConnection();
	Sql=prepareSql();
	Statement=
	connection.prepareStatement(sql);
	Statement.
	
	execute(objet,getParameters()).
	}
	
	Correction
	void update(object) {
	TransparentPersistence.
	update(object);
	}

Reference	…

2.2. Requirement

The goal of quality assurance tools is to simplify the manual task of the programmer in order to ameliorate quality, so these tools must verify some conditions in order to be useful:

- Adding new rules must be simple and extendable; the system must provide different utilities that ease the building of new rules and integrating them into the system
- The design of new rule must be irrelative to the data structure of the system; it must be relative to the rule design only.
- The system must give different levels of error and warning reporting:
  - Error : Critical
  - Error: Medium
  - Error: Low
  - Warning
- The user must also be able to choose the level of errors needed to be checked.

3. Problem Specification

To understand the problem, we will present an example of a quality rule to be able to understand the process of quality checking.

We will take, for example, the rule **"Avoid changing condition variable in loop body"**.

The rule designer must try to figure out how to find such an error. By looking closer at the error pattern, it is clear that the first step is to look for a (for loop) inside a method. The Second step is to save the initialization variables of the (for loop). The final step is to take each statement inside the for-block and make sure that those initialization variables will not be changed by these statements.

Notice that variable can be changed in many ways; using assignment statement, or method call.

We proposed to specify this case using a state diagram (figure 1). The first state designate the fact that we are waiting for a "for loop". When we receive an event that indicates that a "for loop" is detected, an action will save the "loop variable". Then we pass to a state where we are waiting for any statement inside the body of the loop.

When receiving an event that a statement is detected, an action will verify that this statement will not change the loop variable. We pass to the end state when receiving an event indicating that the end of the loop is reached.

The parser is the only component that can send such events. Our initial objective requires that adding new rules does not affect the written code. So parser must send generic events and not events that are specific to each rule.

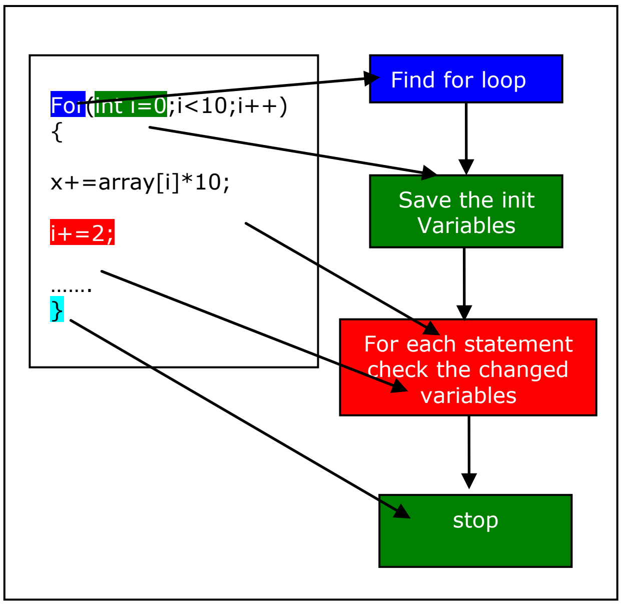

Figure 1. Quality verification steps.

4. The Strategy

Our strategy has three steps:

1- Representing rules by a finite automaton and adding them to the system.

2- Developing a generic two pass parser:

   a. In the first pass the parser collects some useful information about methods; such as:
      - which parameters are modified in the method,
      - which input or output devices are used inside that method.

      It makes all the usful information available for the different rules at runtime.

   b. In a second pass the parser looks for some kind of code patterns and throws events indicating that such a pattern is detected.

5. Rule Representation

A rule is represented by a finite automaton where:

1- States represent the different phases of the detection process such as detecting a loop, listen to statements, and finish listening.

2- Events are sent by the parser and they indicate the start and the end of a pattern or non terminal. These events contain some parameters according to the detected event. For example the event that indicate the starting of a for-loop must passes the initialization variables, their initial values, the loop condition, and the increment statement.

3- Three action types are available:
   - Listen to an event.
   - Stop listening to an event.
   - Fire quality violation alert.

6. Generic Events Representation

A code is represented by a context free grammar using some notation such as BNF notation. These grammars are composed of a set of rules. The left side of the rule is a non terminal which represents the rule name. The right side consists of terminals and non terminals and called reduction.

For example the following grammar can represent a statement:

```
<statement> :- <assignment> |  <for> | <bloc> | <if>
<assignment> :- <var> = <exp>
<if> :- if (<exp>) <statement> else  <statement>
<bloc> :- { <statement>* }
<for> :- for(<type> <var>= <exp>;  <condition>;
                    <inc> ) <statement>
.
.
```

Non terminals are surrounded by <>. Terminals appear free in the grammar such as =, if, else, for, etc.

When the parser reaches the start of a reduction of a grammar it sends an event and passes the terminal that indicates the start of that reduction to the corresponding listener.

Rules can listen to non terminals by specifying the name of that non terminal.

Figure(2) illustrates the architecture of our system.

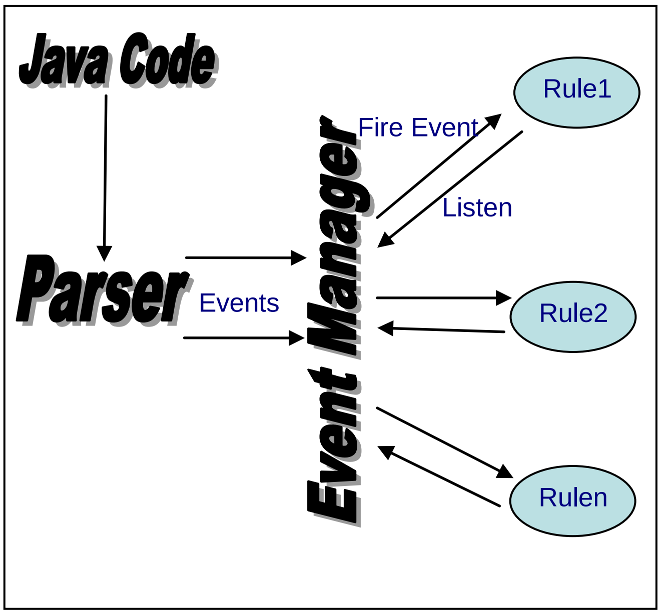

Figure 2. System architecture

The parser takes a Java code as input and sends the corresponding event to an event manager. Rules register their needs in the event manager.

The event manager receives events from the parser and notifies only rules that required listening to this event.

The system is implemented using Java programming language. Rules which are a kind of state diagrams can be edited using a normal UML editor. We have used "magic draw".

The system was plugged into JBuilder developing environment in order to be easy to access. It can be plugged into any pluggable environment.

7. Conclusion

Quality is a main aspect of a program. It affects the development time, the cost, the scalability and the usability. Frameworks can help to satisfy quality but they are not sufficient.

This paper presented an automatic tool for quality verification. This tool allows adding new quality directions dynamically. Directions are represented by a finite state automaton where events are the starting and ending of CFG non terminals. A generic parser allows the extraction of these events and sending theme to listening rules.

Current work aims to develop a framework that allows minimizing the quantity of manually added code.

8. References

[1] Clinton. Sprauve, "Implementing a Performance Test Strategy Using Open Source Software". PSQT/PSTT 2004 East Conference.

[2] Abdo Beirekdar, "A Methodology for Automating Guidline Review of Web Sites". Thesis of Facultés Universitaires Notre-Dame de la Paix, 2004.

[3] Eugene Kuleshov, "Using the Spring AOP Framework with EJB Components", dev2dev 2005, http://dev2dev.bea.com/pub/a/2005/12/spring-aop-with-ejb.html

[4] RICK HIGHTOWER, "An Introduction to Spring", Java Developer's Journal 2005.

[5] Casey Kochmer, "Introduction to Struts", JSP Insider magazine, April 2001. http://www.jspinsider.com/tutorials/jsp/struts/strutsintro.view

[6] Peter Varhol, "Applying the MVC Design Pattern Using Struts", javapro magazine, may 2002, http://www.fawcette.com/javapro/2002_05/magazine/columns/weblication/default.asp.

[7] Firesmith, Donald G. , Deugo, Dwight , "Applying Design Patterns in Java," in Java Gems book, Cambridge University Press, 1998.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design patterns: Abstraction and reuse of object - oriented design", In European Conference on Object-Oriented Programming, 1993, pages 406-431.

[9] Wei-Tek Tsai, Yongzhong Tu, Weiguang Shao, Ezra Ebner, "Testing Extensible Design Patterns in Object-Oriented Frameworks through Scenario Templates", COMPSAC 1999, pages 166-171.

[10] Sergio Antoy, Michael Hanus, "Functional Logic Design Patterns", FLOPS 2002, pages 67-87.

[11] Nadir Gulzar, Kartik Ganeshan, "Practical J2EE Application Architecture", 2003, McGraw-Hill Osborne Media.

[12] Karl Lieberherr, Doug Orleans, and Johan Ovlinger, "Aspect-Oriented Programming with Adaptive Methods", NU-CCS-2001-02, 15 pages.

[13] Martin Fowler, "Inversion of Control Containers and the Dependency Injection pattern", http://martinfowler.com/articles/injection.html.

[14] Christian Queinnec, "Inverting back the inversion of control or, continuations versus page-centric programming", SIGPLAN Notices 38(2), pages 57-64, 2003