

MIEIC - Engenharia de Software - 2010/11

Code Quality

João Pascoal Faria

6/10/2009

Index

- Introduction
- Coding standards
- Code reviews
- Unit testing

INTRODUCTION

The need for quality work

- Software quality matters because software matters
 - Increasing dependence on software
 - Increasing criticality of software systems
- Quality work saves time and money
 - In current industry practice, it is not uncommon to spend half of the project time in testing
 - By focusing on defect prevention and early defect removal, it is possible to increase significantly the quality of delivered products, and reduce significantly system testing and maintenance costs
- Quality work is more predictable
 - The testing and repair effort of a bad quality product is unpredictable

Personal responsibility

- A software system is as weak as the weakest of its parts
- Even experienced programmers introduce about 100 defects/KLOC (before compile)
- The only way to build high-quality products in a cost-effective way, is by having developers being personally responsible for the quality of their products
- Since defects can best be managed where they are injected, developers should
 - remove their own defects
 - determine the causes of their defects
 - learn to prevent those defects

CODING STANDARDS

The need for code conventions

- Code conventions are important because:
 - 80% of the lifetime cost of a piece of software goes to maintenance.
 - Hardly any software is maintained for its whole life by the original author.
 - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- Code conventions typically cover:
 - filenames, file organization, indentation, comments, declarations, statements, white space, naming conventions, programming practices, examples.
- See also <http://java.sun.com/docs/codeconv/>
- Agile practice: Shared team coding standards



Example of Java Coding Standard

Purpose	To guide the implementation and review of Java programs
Program Organization	Create a separate source file for each top-level (non-nested) class. Write each instruction in a separate line.
Documentation Comments	Use standard Java documentation comments (multi-line comments starting with <code>/**</code> and ending with <code>*/</code>), that can be exported with the Javadoc tool, for all relevant non-private classes, methods and fields. Documentation of private classes, methods or fields is optional, as well as public self-explanatory methods and fields not intended for reuse.
Class Headers	Precede all non-private class with a descriptive header using standard Java documentation comments and tags. In the main class of a program, describe program usage, input and output formats, constraints on the input values, error handling and limits of its operation.
Class Header Format	<pre>/** * Short description of class responsibilities, * collaborations and usage. * * @author author name * @created date and time */</pre>

Full text in [JavaCodingStandard.doc](#).

More information on documentation comments in <http://java.sun.com/j2se/javadoc/>.



CODE REVIEWS



The need for code reviews

- Testing alone is not enough
 - Some internal quality attributes cannot be verified by testing
 - Maintainability
 - Adherence to coding standards
 - Security vulnerabilities (see www.securecoding.cert.org/)
 - Exceptional conditions are very difficult to verify by testing
 - For large numbers of defects, testing is less effective, efficient, and predictable
 - Testing can only prove the existence of defects, but not their absence
 - Even with thorough testing, subtle defects can escape
- Reviews are predictable and efficient
 - Defects are immediately located
 - Time is proportional to the size of the code
- Testing and reviews play complementary roles



Exercise 1: Discover off-by-one errors

```
1. int main(int argc, char* argv[]) {
2.     char source[10];
3.     int i;
4.     strcpy(source, "0123456789");
5.     char *dest = (char *)malloc(strlen(source));
6.     for (i=1; i <= 11; i++) {
7.         dest[i] = source[i];
8.     }
9.     dest[i] = '\0';
10.    printf("dest = %s", dest);
11. }
```

What are the errors?

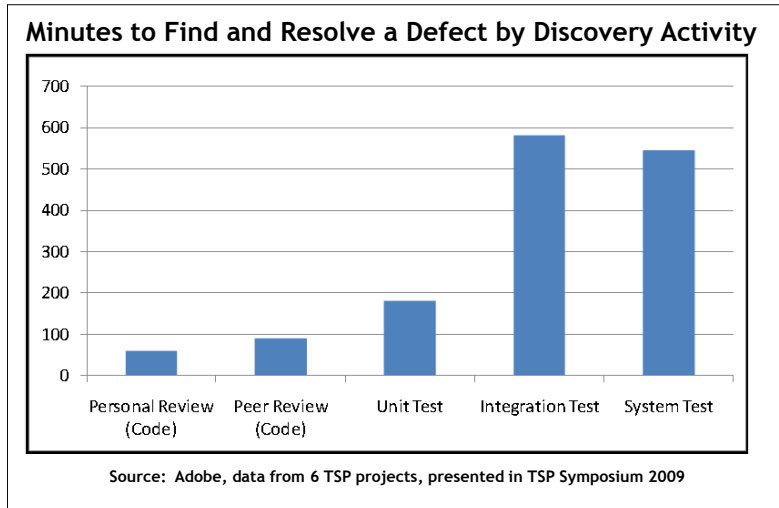
What is the program output?

Can the errors be discovered by testing?

Types of code reviews

- Personal reviews
 - In a personal review, you privately review your product
 - Properly done, personal reviews are one of the most efficient defect removal techniques
- Peers reviews
 - Have one or more peers review your code
 - Independence of peers lead to the discovery of defects that could pass unnoticed by the author
 - Only have peers review your code after you have reviewed it
 - Show respect, don't waste their time
 - Allow peers to concentrate on more fundamental problems
 - In pair programming, one of the elements may act as reviewer
- Best to use in combination

Efficiency of code reviews (Example)



Code reviews best practices

- Produce reviewable products
- Use a checklist derived from your historical defect data
- Take enough review time
 - A review rate of 200 LOC/hour usually gives a good balance between efficacy and efficiency in code reviews (look at your data to confirm!)
 - Or take $\leq 50\%$ of the development time
- Review on paper, not on screen
- Take a break between developing and reviewing (in personal reviews)
- Review in multiple passages, for different types of problems
- Review before testing (or even compile; why?)
- Measure your review process and use data to improve your reviews

Defeitos mais frequentes

- Para ser mais eficaz, revisão deve ser baseada em *checklist* de problemas mais frequentes (do próprio ou da comunidade), como por exemplo a seguinte lista de falhas mais frequentes (dados da IBM)

Classificação	Descrição	Frequência
Algoritmo	execução incorrecta ou em falta que pode ser corrigida sem ser necessário introduzir alterações arquitecturais no software	43.4 %
Atribuição	valores incorrectamente atribuídos ou não atribuídos	22.0 %
Teste	validação de dados incorrecta ou expressões condicionais incorrectas	17.5 %
Função	falha que afecta uma quantidade considerável de código e refere-se a uma capacidade do software que está em falta ou construída incorrectamente	8.7 %
Interface	interacção incorrecta entre módulos/componentes	8.2 %

(*) Orthogonal Defect Classification (ODC)

Example of personal review checklist

Developer João Pascoal Faria Date 10/Nov/08
Program Source file comparator Program # 8 (Java)

Past Defects	Category	What to verify	(1)	(2)	(3)	(4)	(5)	(6)
#4, #5	Project and environment settings	Verify that all project and environment settings have been appropriately set for this program, namely java version, and regional settings.	✓					
#17, #18, #28	Logic	Verify method bodies (instructions) for logical (algorithmic) correctness and consistency. Verify all conditions in branch and loop statements.		#59 (literal)	✓	#60 (varref)	✓	✓
#29, #32, #20	Calls	Verify that all method and library calls are used correctly, without violating any known pre-conditions. Verify that the correct methods are being called.		✓	✓	✓	✓	✓
#13, #33	Exception handling	Verify that the applicable exceptions are handled. Verify that all applicable pre-conditions are checked or intentionally unchecked.		✓	✓	✓	✓	✓
#12, #15, #16, #19, #26	Comments and messages	Verify comments and IO messages for correctness and typing errors.		✓	✓	✓	✓	✓
#27	Standards	Ensure that the code conforms to the coding standards. Check reuse and change control tags.		✓	✓	✓	✓	✓

(1) Overall Program, (2) FileDifferenceCLI, (3) SourceCodeParser, (4) Operation, (5) Delta

Example of Code Review Script (PSP)

Purpose	To guide you in reviewing programs
Entry Criteria	<ul style="list-style-type: none"> - A completed and reviewed program design - Source program listing - Code Review checklist - Coding standard - Defect Type standard - Time and Defect Recording logs
General	Do the code review with a source-code listing; do not review on the screen!
Steps	(See next slide)
Exit Criteria	<ul style="list-style-type: none"> - A fully reviewed source program - One or more Code Review checklists for every program reviewed - All identified defects fixed - Completed Time and Defect Recording logs

Example of Code Review Script (PSP)

Step	Activities	Description
1	Review	<ul style="list-style-type: none"> - Follow the Code Review checklist. - Review the entire program for each checklist category; do not try to review for more than one category at a time! - Check off each item as it is completed. - For multiple procedures or programs, complete a separate checklist for each.
2	Correct	<ul style="list-style-type: none"> - Correct all defects. - If the correction cannot be completed, abort the review and return to the prior process phase. - To facilitate defect analysis, record all of the data specified in the Defect Recording log instructions for every defect.
3	Check	<ul style="list-style-type: none"> - Check each defect fix for correctness. - Re-review all design changes. - Record any fix defects as new defects and, where you know the number of the defect with the incorrect fix, enter it in the fix defect space.

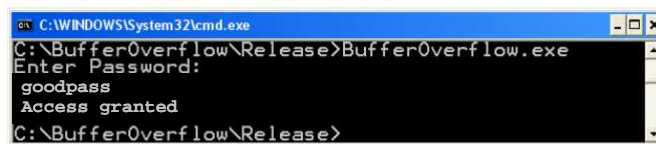
Exercise 2 - Source code (1/3)

```
bool IsPasswordOK(void) {
    char Password[12]; // Memory storage for pwd
    gets (Password);    // Get input from keyboard
    if (!strcmp (Password, "goodpass")) return (true); // Password Good
    else return (false); // Password Invalid
}

void main(void) {
    bool PwStatus;      // Password Status
    puts ("Enter Password:"); // Print
    PwStatus=IsPasswordOK(); // Get & Check Password
    if (PwStatus == false) {
        puts ("Access denied"); // Print
        exit (-1);             // Terminate Program
    }
    else puts ("Access granted");// Print
}
```

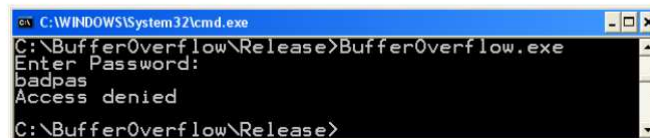
Exercise 2 - Example behavior (2/3)

Run #1 Correct Password



```
C:\WINDOWS\System32\cmd.exe
C:\BufferOverflow\Release>BufferOverflow.exe
Enter Password:
goodpass
Access granted
C:\BufferOverflow\Release>
```

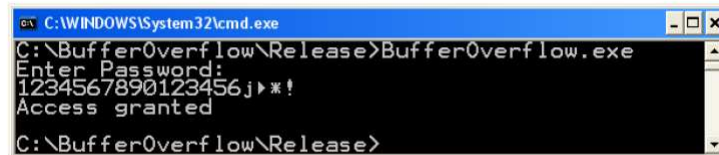
Run #2 Incorrect Password



```
C:\WINDOWS\System32\cmd.exe
C:\BufferOverflow\Release>BufferOverflow.exe
Enter Password:
badpas
Access denied
C:\BufferOverflow\Release>
```

Exercise 2 - Security flaw (3/3)

A specially crafted string "1234567890123456j►*!" produced the following result.



```
C:\WINDOWS\System32\cmd.exe
C:\BufferOverflow\Release>BufferOverflow.exe
Enter Password:
1234567890123456j►*!
Access granted
C:\BufferOverflow\Release>
```

What happened ?

What is the problem in the code?

Beneficts of applying coding standards and code reviews

- Microsoft secure code project
- Relying heavily on coding standards, personal reviews and peer reviews, besides testing
- 8-person software development team
- Created 30 K lines of new and modified code in 7 months

Phase	Post code complete defects	
	Prior similar release	TSP-Secure release
Integration Test	237	4
System Test	473	3
User Acceptance Test	153	10
Security code defects	Data not available	0
Total Defects	1072	17

(Source: TSP Secure, Noopur Davis et al, TSP Symposium 2009)

UNIT TESTING (REVISÃO)



Conceitos básicos

- Teste - técnica dinâmica de verificação de programas, em que se exercita o programa com determinados casos de teste e se verifica se produz os resultados esperados
 - O objectivo é descobrir defeitos e avaliar a qualidade
 - Testes não permitem provar que um programa está correcto devido à infinidade de casos de teste possíveis

- Caso de teste - dados de entrada + resultados esperados

Caso de teste	Dados de entrada		Resultados esperados
	a	b	mdc(a, by)
1	2	3	1
2	2	4	2

- Testes unitários - testes ao nível do método, classe ou módulo, normalmente realizados pelo próprio programador



Boas práticas de teste

- Testar o mais cedo possível
 - Custo de corrigir um *bug* cresce com o tempo decorrido
- Automatizar os testes \Rightarrow *JUnit*
 - Dada necessidade de re-executar frequentemente os testes
 - Automatizar sobretudo teste de APIs (GUIs é mais difícil)
- Escrever os testes antes do programa a testar \Rightarrow *TDD*
 - Pelo menos especificar os testes logo após a interface das classes
 - Ajuda a esclarecer requisitos
 - Casos de teste são especificações parciais
- Começar por criar testes baseados na especificação (*cx.negra*) e complementar c/ testes p/cobrir o código (*cx.branca*)



JUnit

- *Framework* (conjunto de classes) para teste unitário da família xUnit
 - JUnit - Java; NUnit - C#; CppUnit - C++
- Integrado no Eclipse
- Permite criar classes de teste com métodos de teste contendo asserções
- *Test runner* executa os métodos de teste e mostra os que passaram (a verde) e os que falharam (a vermelho)
 - “Keep the bar green to keep the code clean”
- Ver mais detalhes em www.junit.org



TDD - Test Driven Development

- Development approach appropriate for unit testing
 - The rhythm of Test-Driven Development can be summed up as follows:
 1. Quickly add a test.
 2. Run all tests and see the new one fail.
 3. Make a little change.
 4. Run all tests and see them all succeed.
 5. Refactor to remove duplication.
- [Kent Beck, Test-Driven Development, Addison-Wesley, 2003]

Técnicas de teste de caixa negra

■ Partição em classes de equivalência



- Partir domínio de valores de entrada em classes com comportamento esperado similar, e testar pelo menos um valor de cada classe
- Exemplo a testar $\text{abs}(x)$: $x < 0$, $x \geq 0$
- Considerar classes de entradas válidas e entradas inválidas
- Exemplo a testar $\text{sqrt}(x)$: $x < 0$ (inválido), $x \geq 0$ (válido)

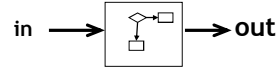


■ Análise de valores fronteira



- Testar valores na fronteira de cada classe e imediatamente acima e abaixo (além de / em vez de valores intermédios)
- Testar valores especiais (null, 0, etc.)
- “Bugs lurk in corners and congregate at boundaries.” (B.Beizer)
- Exemplo a testar $\text{abs}(x)$: $x = \text{min}$, $x = -1$, $x = 0$, $x = 1$, $x = \text{max}$

Técnicas de teste de caixa branca



- Cobertura de instruções
 - Garantir que todas as instruções são exercitadas
 - Usar ferramenta Coverlipse (Run As → Junit w/Coverlipse) para analisar cobertura, e conceber testes adicionais se necessário
- Cobertura de decisões
 - Garantir adicionalmente que todas as decisões (condições de **if**, **while**, **for**, etc.) tomam os valores **true** e **false**
 - `int abs(int x) {if(x<0) x = -x; return x;}`
→ `abs(-1)` cobre instruções mas não decisões
- Cobertura de decisões e condições
 - Garantir adicionalmente que todas as condições que compõem uma decisão composta tomam os valores **true** e **false**
 - `if (i>0 && a[i]>a[i-1]) ...` → testar **TT**, **TF** e **F-**

Exemplo com JUnit 3.8.1

```
class MyMath {  
    // Dá o maior divisor comum positivo de 2 inteiros  
    // não nulos pelo algoritmo de Euclides; se algum arg.  
    // for 0 dá (devia dar...) IllegalArgumentException.  
    public static int mdc(int a, int b) {  
        while (b > 0) {  
            int aux = a % b;  
            a = b;  
            b = aux;  
        }  
        return a;  
    }  
}
```

Buggy!?

métodos de teste: void testXXX()

`assertEquals(esperado, actual)`

`fail()` - assinala teste falhado

```
import junit.framework.TestCase;  
class MyMathTest extends TestCase {  
    public void testMdcPositive() {  
        assertEquals(1, MyMath.mdc(2, 3));  
        assertEquals(2, MyMath.mdc(2, 4));  
    }  
    public void testMdcNegative() {  
        assertEquals(2, MyMath.mdc(-4, 6));  
    }  
    public void testMdcZero() {  
        try { MyMath.mdc(0, 1); fail(); }  
        catch (IllegalArgumentException e) {}  
        catch (Exception e) { fail(); }  
    }  
}
```

Exemplo com JUnit 4.0

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

class MyMathTest {
    @Test public void testMdcPositive() {
        assertEquals(1, MyMath.mdc(2, 3));
        assertEquals(2, MyMath.mdc(4, 6));
    }
    @Test(expected=IllegalArgumentException.class)
    public void testMdcZero() {
        MyMath.mdc(0, 1);
    }
}
```

Tira partido das novas *features* do Java 1.5

References

- Software Engineering, 8th edition, Ian Sommerville, Addison-Wesley, 2006
 - Chapter 22 - Verification and Validation
- Winning with Software: An Executive Strategy, Watts Humphrey, 2001
- Java Coding Standards
 - http://www.ontko.com/java/java_coding_standards.html
- Code Conventions for the Java Programming Language
 - <http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>
- CERT Secure Coding Standards
 - <https://www.securecoding.cert.org/confluence/display/seccode/CERT+Secure+Coding+Standards>