Verifica Statica

Ispezione

Ispezione (Fagan, 1976)

- Anomalie del codice possono influenzare in modo scorretto il codice
  - ✓ esempi: distribuzione linee bianche, numero linee di commento,...
- Tecniche di ispezione di codice possono rilevare ed eliminare anomalie fastidiose e rendere più precisi i risultati
  - ✓ una tecnica completamente manuale per trovare e correggere errori
    - » poco tecnologica, ma efficace
    - » ma sono possibili alcuni supporti automatici …
  - ✓ è estendibile a progetto, requisiti, ... seguendo principi organizzativi analoghi

Ruoli nell'ispezione di software

- moderatore:
  - ✓tipicamente proviene da un altro progetto. Presiede le sedute, sceglie i partecipanti, controlla il processo .
- lettori, addetti al test:
  - ✓leggono il codice al gruppo, cercano difetti
- autore:
  - ✓partecipante passivo; risponde a domande quando richiesto

3

Il processo di ispezione del software

- pianificazione
  - ✓ scelta di partecipanti e checklist, pianificazione di meeting
- fasi preliminari
  - ✓ fornite le informazioni necessarie e assegnati i ruoli
- preparazione
  - ✓ lettura del documento, individuazione dei difetti
- Ispezione
  - ✓ meeting: raccolta e discussione congiunta dei problemi trovati dai singoli revisori, ricerca di ulteriori difetti
- lavoro a valle
  - ✓ l'autore modifica il documento per rimuovere i difetti
- seguito (possibile re-ispezione)
  - ✓ controllo delle modifiche, raccolta di dati

4

Nelle riunioni …

- obiettivo: trovare il maggior numero possibile di difetti
  - ✓ massimo 2 riunioni al giorno di 2 ore ciascuna
  - ✓ circa 150 linee di codice sorgente all'ora
- approccio: parafrasare il codice linea per linea
  - ✓ ricostruire l'obiettivo dal codice sorgente
  - ✓ può essere "test manuale"
- necessario restare in tema
  - ✓ seguire le checklist
  - ✓ trovare e registrare difetti, ma non correggerli
  - ✓ il moderatore è responsabile di evitare anarchia

5

Checklist – Esempio NASA

- circa 2.5 pagine per codice C , 4 per FORTRAN
  - ✓ diviso in: funzionalità, uso dei dati, controllo, connessioni, calcolo, manutenzione, chiarezza
- esempi:
  - ✓ ogni modulo contiene una singola funzione?
  - ✓ il codice corrisponde al progetto dettagliato?
  - ✓ i nomi di costante sono maiuscoli?
  - ✓ si usa il cast di tipo dei puntatori?
  - ✓ "INCLUDE"annidati di files sono evitati?
  - ✓ gli usi non standard sono isolati in sottoprogrammi e ben documentati?
  - ✓ i commenti sono sufficienti per capire il codice?

6

Inspection Defect Log

Product	Simple Sort	**Date**	October 23, 1999	
Author	Fraser Macdonald			
Defect#	**Description**	**Type**	**Location**	**Severity**
1	Function max() is defined, but never used. No failure apparently, but checklist violation.	function calls	line 12, function max()	trivial
2	Parameters are passed by value, not by reference. "swap" doesn't correctly swap the numbers, so the sort is not carried out correctly.	function calls	line 5, function swap()	failure

7

Incentivi (Fagan, 1986)

- difetti trovati nell'ispezione non sono usati nella valutazione personale
  - ✓i programmatori non hanno motivo di nascondere i difetti
- difetti trovati durante il test (dopo l'ispezione) sono usati nella valutazione personale
  - ✓i programmatori sono incentivati a trovare difetti nell'ispezione, ma non inserendone intenzionalmente

8

Varianti

- Revisione attiva di progetto (Parnas & Weiss, 1985)
  - ✓ un revisore impreparato può sedere in silenzio ed essere inutile.
  - ✓ scegliere revisori con esperienze specifiche
    - » revisori differenti devono cercare difetti differenti
- Ispezione a fasi (Knight & Meyers, 1993)
  - ✓ una serie di fasi più piccole che mettono a fuoco problemi specifici in un ordine determinato
  - ✓ Ispettore singolo (e meno esperto) per controlli semplici
  - ✓ Più ispettori (esperti) per controlli complessi

9

Perché l'ispezione funziona?

- L'evidenza dice che è cost-effective, perchè?
  - ✓ Processo formale, dettagliato con tracciamento dei risultati
  - ✓ Check-lists: processo che si automigliora
  - ✓ Aspetti sociali del processo, specialmente per gli autori
  - ✓ Considera l'intero spazio di ingresso
  - ✓ Si applica anche a programmi incompleti
- limiti
  - ✓ Scala: tecnica inerentemente a livello di unità
  - ✓ Non incrementale

10

Esempio: Checklist per codice Java

- Commenti
  - ✓ Sintassi: controlli ortografici e grammaticali della lingua
  - ✓ Struttura
  - ✓ Stile
  - ✓ Contenuto
- Codice
  - ✓ Stile
  - ✓ Correttezza Semantica
  - ✓ Ridondanza
  - ✓ Qualità

11

Commenti: Sintassi e Struttura

- Sintassi
  - ✓ controlli ortografici, grammaticali e linguistici
  - ✓ commenti formattati per javadoc …
- Struttura
  - ✓ ci sono dati identificativi (titolo, data, versione, autore)
  - ✓ librerie non standard usate, piattaforma Java richiesta

12

Commenti: Stile

- Non c'è un uso indiscriminato di abbreviazioni
- Non ci sono troppe ripetizioni
- Linguaggio comprensibile e frasi leggibili
- Il commento è presente in ogni classe
- Il commento è ben visibile nel file
- Le variabili sono commentate quando sono dichiarate
- I commenti non sono dispersivi (mal disposti, interrotti e ripresi)
- I commenti non sono esageratamente densi nel codice (compromettendo la leggibilità del codice)

13

Commenti: Contenuto

- L'intestazione dei metodi e delle classi corrispondono ai commenti
- Il commento descrive i metodi principali della classe
  - ✓ *Commentati tutti i metodi con il corpo più lungo di tre LOC*
- Sono spiegati i passaggi più difficili
  - ✓ *I commenti non sono mai banali*
- Il commento dà indicazione chiara (almeno generale) della funzione della classe
- Il commento è completo in ogni sua parte
- Il commento è contestuale (pertinente al codice circostante)

14

Codice: Stile

- Gli identificatori di costante sono espressi con una convenzione diversa dagli identificatori di variabile
- I nomi delle classe sono espressi con una convenzione diversa dagli identificatori di variabile
- I nomi delle variabili e delle classi sono significativi
- Il codice si presta alla lettura oppure è troppo concentrato o troppo diluito
  - ✓ Si applica uno stile
  - ✓ Lo stile è leggibile
  - ✓ Lo stile è mantenuto per tutto il progetto
  - ✓ Lo stile prevede una buona indentazione

15

Codice: Correttezza semantica e Ridondanza

- Correttezza semantica
  - ✓ L'esecuzione del codice rispetta le specifiche dei commenti
  - ✓ Non vengono usate librerie non standard
- Ridondanza
  - ✓ Non ci sono variabili dichiarate ma non utilizzate
  - ✓ Non ci sono metodi con funzionalità simili
  - ✓ Non ci sono variabili con funzionalità simili

16

Codice: Qualità

- Ogni classe ha almeno un costruttore
- Altrimenti
  - ✓ Ha solo metodi statici
  - ✓ Si tratta di una classe astratta o di un'interfaccia
- Non si usano metodi con side-effects non necessari
- Non si usano variabili globali non necessarie
- C'è omogeneità nella dimensione delle classi
- Ci sono metodi "equilibrati"
  - ✓ Non ci sono metodi che incorporano troppe funzionalità e altri troppo elementari

17

Codice: Qualità

- Ogni metodo rispecchia una funzionalità precisa
- Non si fa uso di metodi deprecati
- Si fa uso di eccezioni
  - ✓ Fanno effettivamente riferimento a situazioni anomale
  - ✓ Vengono usate in modo coerente
  - ✓ Coprono tutte le parti critiche
- Si fa buon uso del paradigma ad oggetti
  - ✓ Buon uso di ereditarità, polimorfismo …
- Si usa uno stile elegante nella stesura del codice
  - ✓ Non troppi casting, buon uso delle funzionalità di Java

18

Esercitazione: Uso di Checklist per codice C (NASA)

- Effettuare code inspection dell'applicazione in questione utilizzando le checklist a seguire
- Due checklist
  - ✓Rispetto coding standard
  - ✓Ricerca di potenziali difetti

19

Checklist A

C coding standards

20

Functionality

1. Does each module have a single function?
2. Is there code which should be in a separate function?
3. Is the code consistent with performance requirements?
4. Does the code match the Detailed Design? (The problem may be in either the code or the design.)

21

Data Usage – Data and Variables

1. Are all variable names lower case?
2. Are names of all internals distinct in 8 characters?
3. Are names of all externals distinct in 6 characters?
4. Do all initializers use "="? (v.7 and later; in all cases should be consistent)?
5. Are declarations grouped into externals and internals?
6. Do all but the most obvious declarations have comments?
7. Is each name used for only a single function (except single character variables "c", "i", "j", "k", "n", "p", "q", "s")?

22

Data Usage – Constants

1. Are all constant names upper case?
2. Are constants defined via "# define"?
3. Are constants that are used in multiple files defined in an INCLUDE header file?

23

Data Usage – Pointers Typing

1. Are pointers declared and used as pointers (not integers)?
2. Are pointers not typecast (except assignment of NULL)?

24

Control

1. Are "else__if" and "switch" used clearly? (generally "else__if" is clearer, but "switch" may be used for not-mutually-exclusive cases, and may also be faster).
2. Are "goto" and "labels" used only when absolutely necessary, and always with well-commented code?
3. Is "while" rather than "do-while" used wherever possible?

25

Linkage

1. ARE "INCLUDE" files used according to project standards?
2. Are nested "INCLUDE" files avoided?
3. Is all data local in scope (internal static or external static) unless global linkage is specifically necessary and commented?
4. Are the names of macros all upper case?

26

Computation – Lexical Rules for Operators

1. Are unary operators adjacent to their operands?
2. Do primary operators "->" "." "()" have a space around them? (should have none.)
3. Do assignment and conditional operators always have space around them?
4. Are commas and semicolons followed by a space?
5. Are keywords followed by a blank?
6. Is the use of "(" following function name adjacent to the identifier?
7. Are spaces used to show precedence? If precedence is at all complicated, are parentheses used (especially with bitwise ops)?

27

Computation – Evaluation Order

1. Are parentheses used properly for precedence?
2. Does the code depend on evaluation order, except in the following cases?
   a. exprl, expr2
   b. exprl? expr2 : exp2
   c. exprl & & expr2
   d. exprl || expr2
3. Are shifts used properly?
4. Does the code depend on order of effects? (e.g., i = i++;)?

28

Maintenance

1. Are library routines used?
2. Are non-standard usages isolated in subroutines and well documented?
3. Does each module have one exit point?
4. Is the module easy to change?
5. Is the module independent of specific devices where possible?
6. Is the system standard defined types header used if possible (otherwise use project standard header, by "include")?
7. Is use of "int" avoided (use standard defined type instead)?

29

Clarity - Comments

1. Is the module header informative and complete?
2. Are there sufficient comments to understand the code?
3. Are the comments in the modules informative?
4. Are comment lines used to group logically-related statements?
5. Are the functions of arrays and variables described?
6. Are changes made to a module after its release noted in the development history section of the header?

30

Clarity - Layout

1. Is the layout of the code such that the logic is apparent?
2. Are loops indented and visually separated from the surrounding code

31

Layout – Lexical Control Structures

1. Is a standard project-wide (or at least consistent) lexical control structure pattern used?
   e.g.

- ✓ while (expr)
  {
  stmts;
  }
- ✓ or
- ✓ while (expr) {
  stmts;
  }

32

Checklist B

C code

33

Functionality

1. Is the functionality described in the specification fully implemented by the code?
2. Is there any excess functionality implemented by the code but not described in the specification?
3. Is the program interface implemented as described in the specification?

34

Initialization and Declarations

1. Are all local and global variables initialized before use?
2. Are variables and class members i) required, ii) of the appropriate type, and iii) correctly scoped?

35

Function Calls

1. Are parameters presented in the correct order?
2. Are pointers and & used correctly?
3. Is the correct function being called, or should it be a different function with a similar name?

36

Arrays

1. Are there any o -by-one errors in array indexing?
2. Can array indexes ever go out-of-bounds?

37

Pointers and Strings

1. Check that pointers are initialized to NULL
2. Check that pointers are never unexpectedly NULL
3. Check that all strings are identified by pointers and are NULL-terminated at all points in the program

38

Dynamic Storage Allocation

1. Is too much/too little space allocated?

39

Output Format

1. Are there any spelling or grammatical errors in displayed output?
2. Is the output formatted correctly in terms of line stepping and spacing?

40

Computation, Comparisons and Assignments

1. Check order of computation/evaluation, operator precedence and parenthesizing
2. Can the denominator of a division ever be zero?
3. Is integer arithmetic, especially division, ever used inappropriately, causing unexpected truncation/rounding?
4. Are the comparison and boolean operators correct?
5. If the test is an error-check, can the error condition actually be legitimate in some cases?
6. Does the code rely on any implicit type conversions?

41

Flow of Control

1. In a switch statement, is any case not terminated by break or return?
2. Do all switch statements have a default branch?
3. Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?

42

Files

1. Are all files properly declared and opened?
2. Is a file not closed in the case of an error?
3. Are EOF conditions detected and handled correctly?

43