La qualità del software

Prof. Paolo Ciancarini
Corso di Ingegneria del Software
CdL Informatica
Università di Bologna

Obiettivi della lezione

- I rischi della cattiva qualità del software
- Gli standard di qualità
- Il testing

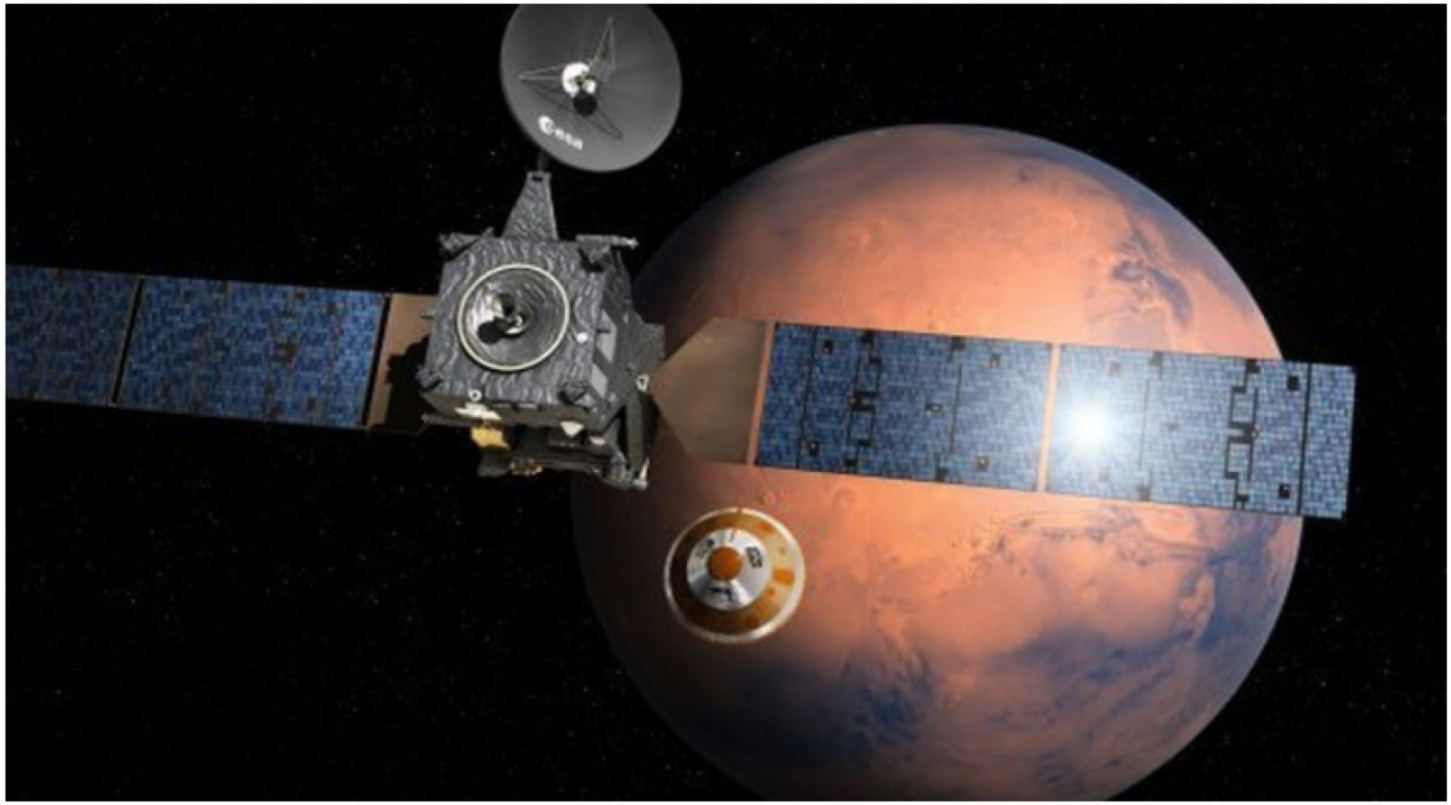
Software Engineering Disaster Hall of Fame

http://catless.ncl.ac.uk/Risks/
http://bugsniffer.blogspot.com/2007/11/infamous-software-failures.html

#	Name	When	Cost	Impact	Cause(s)
1	Soviet Nuclear Missile False Alarm	Sep 26, 1983	500,000,000 lives at risk	nearly-averted nuclear holocaust	» deadline pressure » faulty signal analysis » false positive
2	NORAD Nuclear Missile False Alarm	Jun 3-6, 1980	500,000,000 lives at risk	nearly-averted nuclear holocaust	» scope creep » corrupted data » test message was a blank attack warning message
3	HMS Sheffield Exocet Missile Mis-classification	May 4, 1982	30 deaths 257 lives at risk £23,200,000	» failure to intercept incoming missile » ship loss	» inadequate requirements » unnecessary simplicity » insufficient maintenance
4	Chinook Helicopter Crash	Jun 2, 1994	29 deaths		
5	Patriot Missile Clock Drift	Feb 25, 1991	28 deaths 98 injured	failure to intercept incoming missile	» short-sighted temporal requirements » rounding error » clock drift
6	Panama Radiation Therapy Overdose	2000	18 deaths 10 injured	radiation overdose (10-110%)	» overreliance on automation » inconsistent output » double counting » inadequate testing
7	V-22 Osprey Crash	Dec 11, 2000	4 deaths	aircraft crash	confusing output
8	Therac-25	1985-1987	3 deaths 3 injured	radiation overdose (10000%)	» unrealistic risk assessment » lack of defensive design » inadequate testing » arithmetic overflow » race condition » complacency
9	USS Yorktown	Sep, 1996	400 lives at risk 2 day delay	» complete systems crash » propulsion system failure » ship disabled	» lack of defensive design » inadequate training » insufficient input validation » edge case » divide by zero
10	F-22 Raptor International Dateline Crossing	Feb 11, 2007	6 lives at risk	» complete system crash » mission aborted » retreated by visually following tankers	» lack of defensive design » date/time mishandling

SCHIAPPARELLI POTREBBE ESSERSI SCHIANTATA PER UN ERRORE DI SOFTWARE

DI SMARTWEEK 27 OTTOBRE 2016 6 MIN.



La missione **ExoMars**, finanziata da **Europa** e **Russia**, potrebbe aver fallito a causa di un errore di programmazione. Errore che ha causato lo **schianto della sonda Schiapparelli** sulla **superficie di Marte** la scorsa settimana. Stando a quanto riportato da Nature, probabilmente il software ha ingannato la sonda facendole credere di essere atterrata prima del previsto sulla superficie del pianeta rosso. Il difetto di sistema avrebbe poi condotto la sonda allo schianto.

Fiat Chrysler recalls more than 1-million vehicles due to software problem

12 MAY 2017 - 14:34 *by* DAVID SHEPARDSON

Fiat Chrysler Automobiles CEO Sergio Marchionne. Picture: REUTERS

Washington — On Friday, Fiat Chrysler Automobiles said it was recalling more than 1.25-million trucks worldwide to address a software error linked to reports of one death resulting from a crash, and two injuries.

The Italian-American vehicle maker said it would reprogram computer modules because an error code could temporarily disable side airbag and seatbelt pretensioner deployment when a car was rolled, if the vehicle "were subjected to a significant underbody impact".

The recall covers about 1-million Ram 1500 and 2500 pick-up vehicles made between 2013 and 2016, and Ram 3500 vehicles made between 2014 and 2016 in the US, 216,007 vehicles in Canada; 21,668 in Mexico and 21,530 outside North America, Fiat Chrysler said.

Reuters

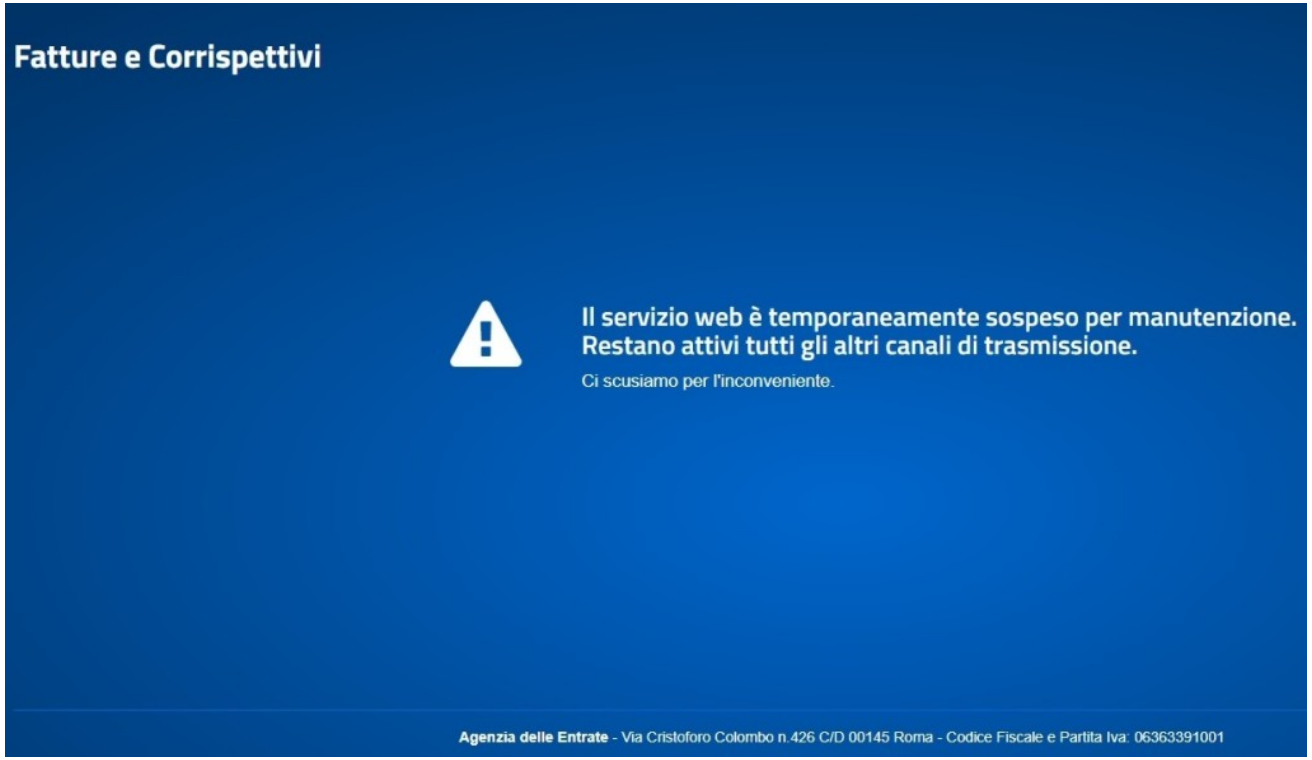
E-GOV

83 commenti 3 minuti 430 condivisioni

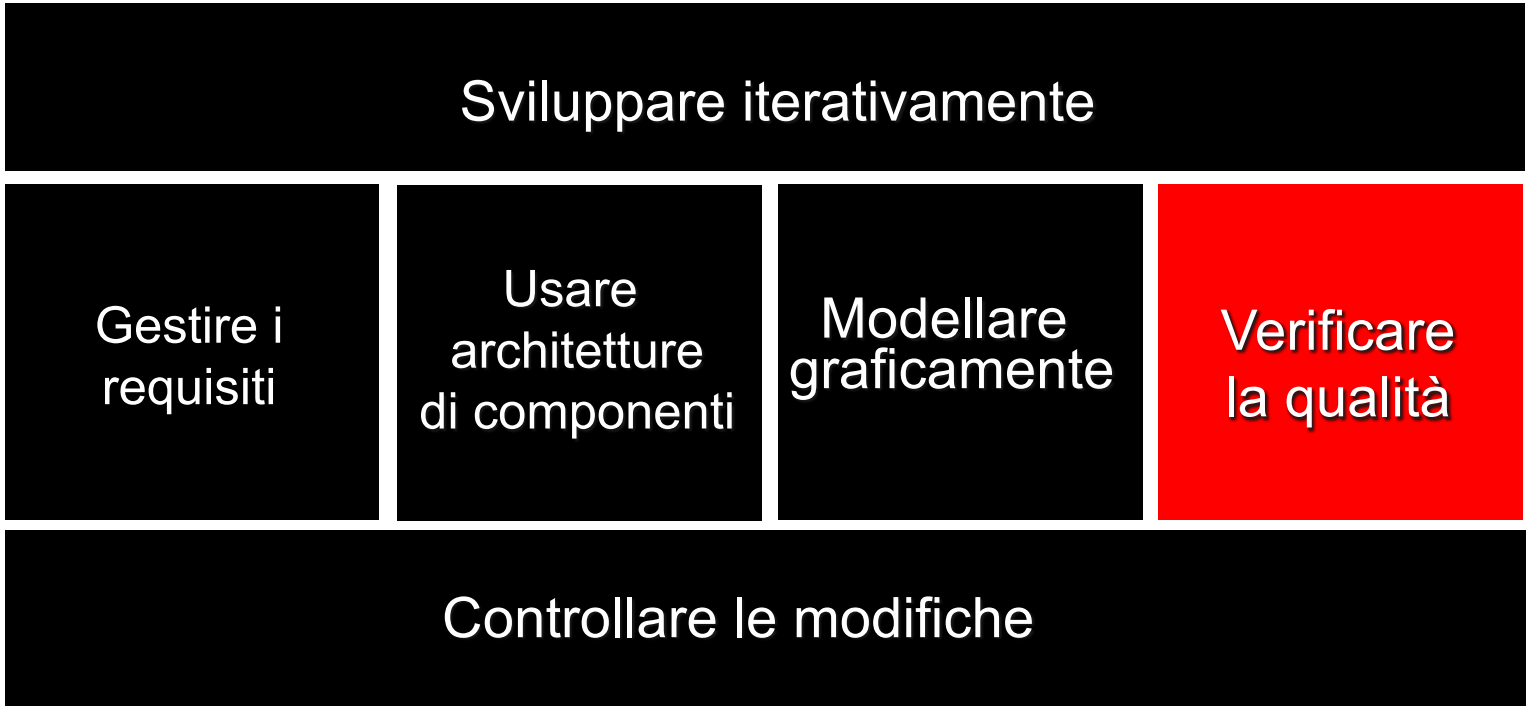
Liquidazione IVA e fatture online, sito chiuso per falla

Buco nella piattaforma online dell'Agenzia delle Entrate, gestita da Sogei, per trasmettere liquidazioni IVA e fatture. Quanti e quali dati sensibili dei contribuenti sono stati diffusi per errore? Non si sa ancora.

di *Pino Bruno @pinobruno* • 25 Settembre 2017, 11:15 • (Fonte **Il Sole 24 Ore - Il Corriere della Sera**)



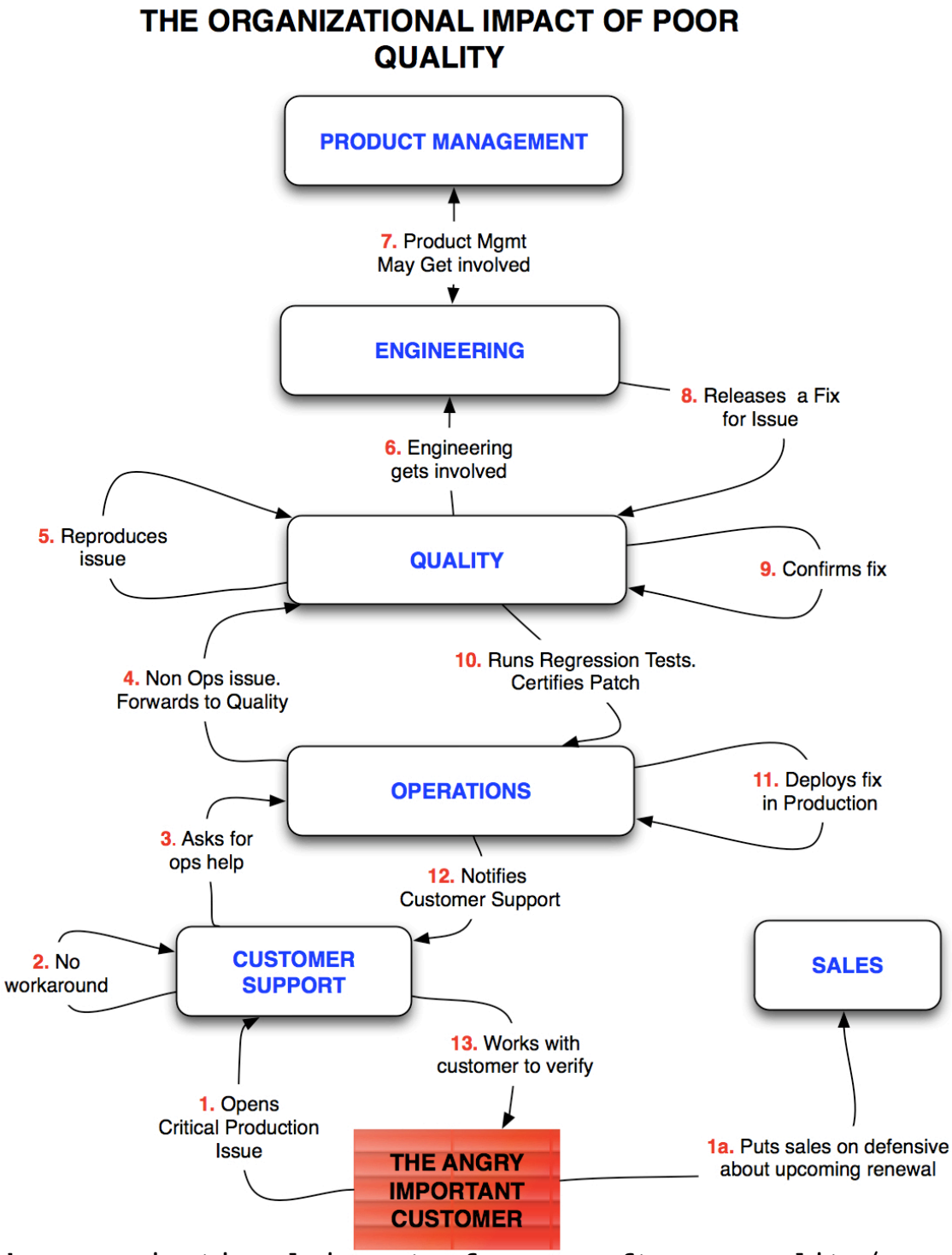
Principi guida dello sviluppo software



La qualità del

Cos’è la qualità del software?

Gestione della qualità nelle grandi aziende



http://saasinterrupted.com/2010/02/01/the-organizational-impact-of-poor-software-quality/

Che cos'è la qualità?

"E ciò che è bene, Fedro, e ciò che non è bene - dobbiamo chiedere ad altri di dirci queste cose?"

Date un voto (“to rate”)

- alla vostra colazione di stamane
- alla mia lezione sui design pattern
- al film che avete visto ieri sera
- a quest’aula
- a Powerpoint
- ad una presentazione

Classificate (“to rank”)

- I vostri tre migliori esami
- I vostri cinque film preferiti
- I vostri post su Facebook
- …

Non confondere rating e ranking!

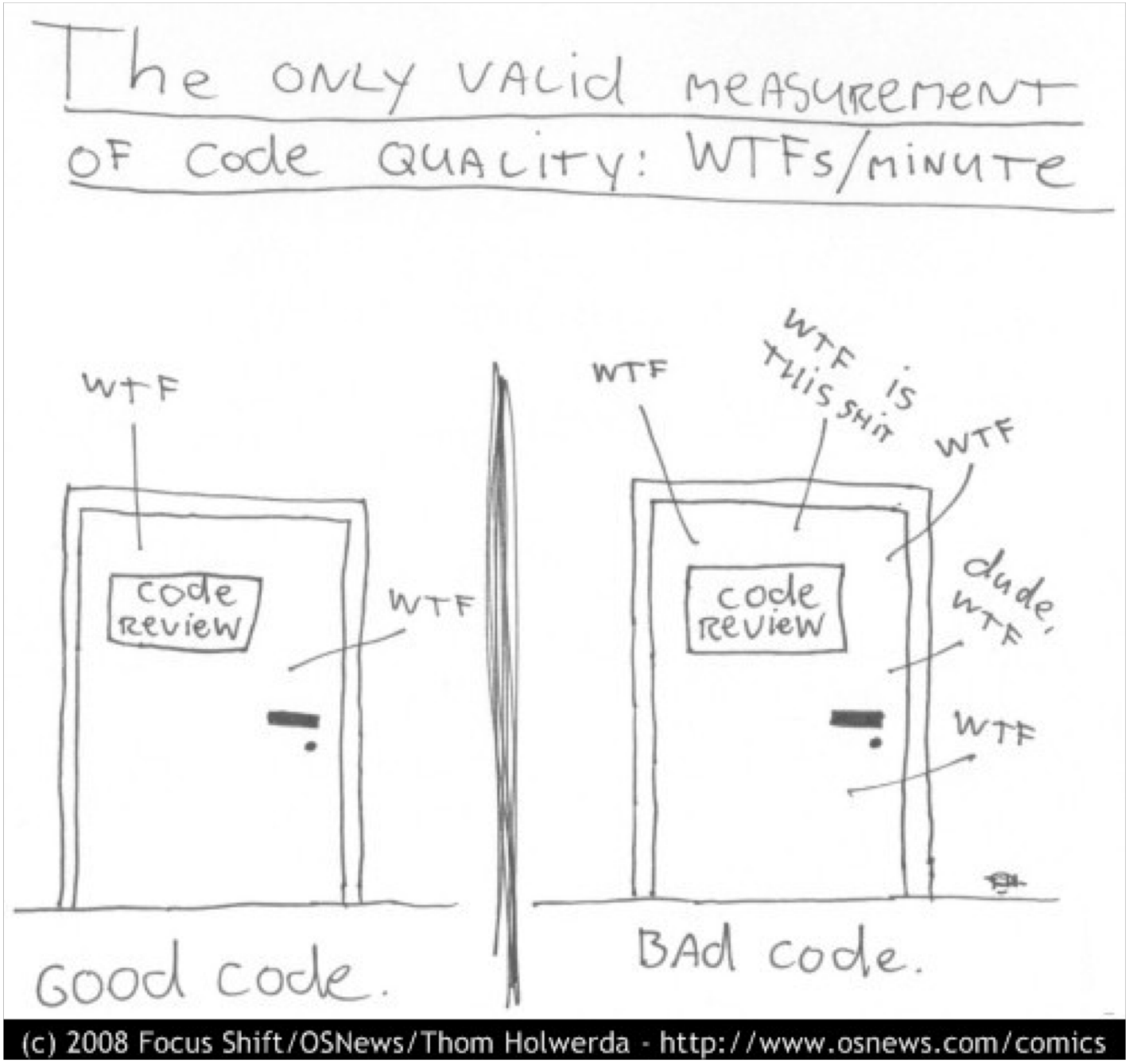
- Un rating è una *valutazione puntuale*, cioè specifica di una entità. Per es.: il rating di un film o la media di libretto di uno studente
- Un ranking è un *confronto* di più rating, *inseriti in una classifica*, per es. per assegnare un premio o confrontare prestazioni

Obiettivi di questa lezione

- Definire le qualità di processi e prodotti sw
  - Qualità interne - visibili al progettista
  - Qualità esterne - visibili all'utente
- Come valutare la qualità
  - La misurazione di indicatori di qualità
  - Il metodo GQM
- Verificare la qualità: tecniche di testing

Discussione

- Come possiamo dire che un prodotto software è di buona qualità?


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

The 5 Stages of Debugging

At some point in each of our lives, we must face errors in our code. Debugging is a natural healing process to help us through these times. It is important to recognize these common stages and realize that debugging will eventually come to an end.

Denial

This stage is often characterized by such phrases as "What? That's impossible," or "I know this is right." A strong sign of denial is recompiling without changing any code, "just in case."

Bargaining/Self-Blame

Several programming errors are uncovered and the programmer feels stupid and guilty for having made them. Bargaining is common: "If I fix this, will you please compile?" Also, "I only have 14 errors to go!"

Anger

Cryptic error messages send the programmer into a rage. This stage is accompanied by an hours-long and profanity-filled diatribe about the limitations of the language directed at whomever will listen.

Depression

Following the outburst, the programmer becomes aware that hours have gone by unproductively and there is still no solution in sight. The programmer becomes listless. Posture often deteriorates.

Acceptance

The programmer finally accepts the situation, declares the bug a "feature", and goes to play some Quake.

Qualità dei prodotti software

1. **Il prodotto deve soddisfare la sua specifica**
   - I test sui requisiti sono il fondamento su cui misurare la qualità
2. **Il prodotto non deve contenere difetti**
   - è molto difficile produrre sw privo di errori
3. **Il prodotto deve rispettare gli standard industriali**
   - Quali standard definiscono i criteri di qualità per valutare il prodotto e il suo processo di sviluppo?
4. **Il prodotto deve presentare le caratteristiche che ci si aspetta da una realizzazione professionale**
   - Per esempio, il suo costo di manutenzione deve essere contenuto e corrispondente alle aspettative del cliente

I test garantiscono la qualità

Code without tests is bad code.

It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is.

With tests, we can change the behavior of our code quickly and verifiably.

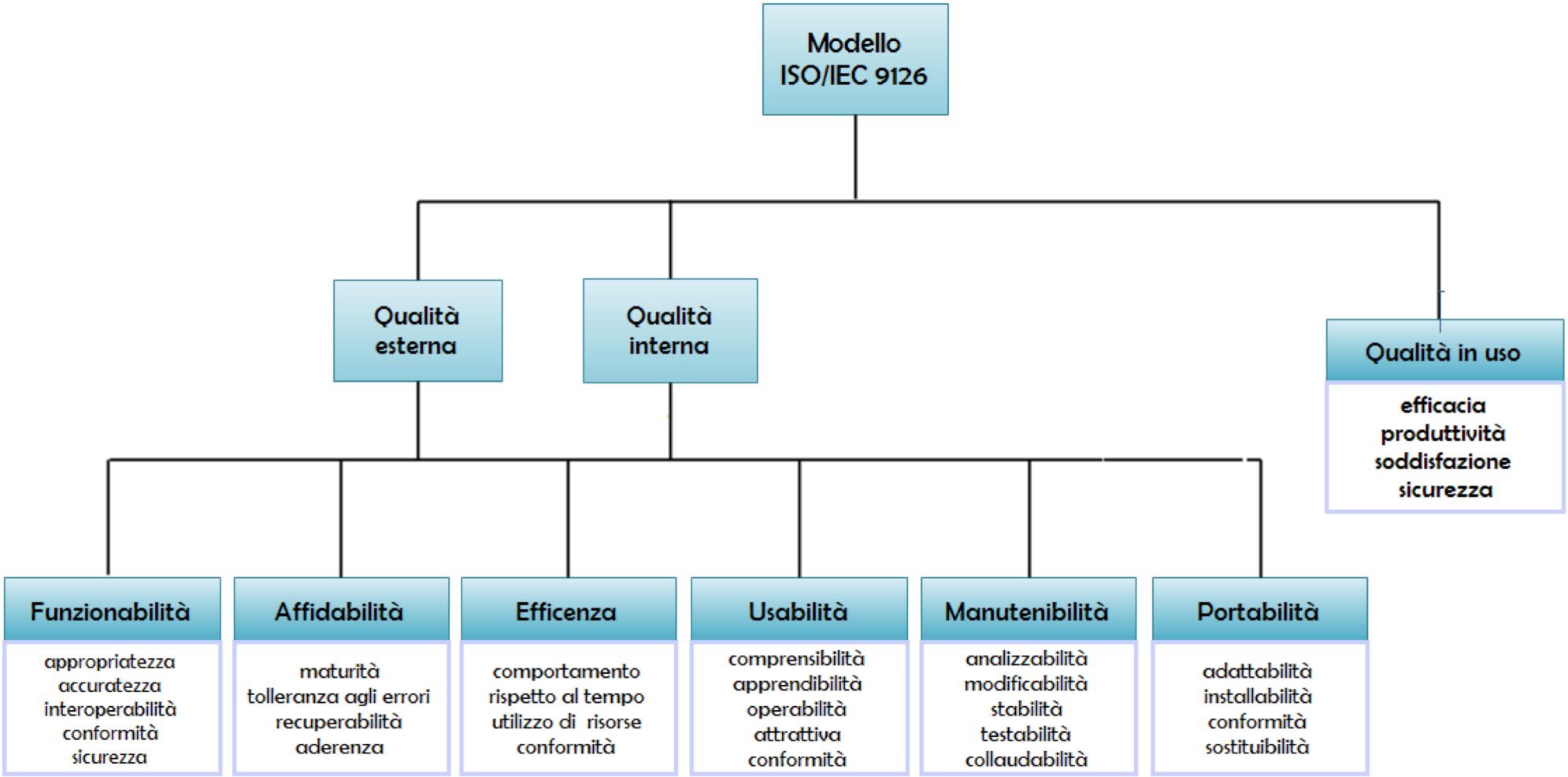
Without them, we really don't know if our code is getting better or worse.

M.Feathers *Working effectively with legacy code*, PH, 2004

Attributi di qualità di prodotto

- Legati alle caratteristiche operative
  - Correttezza (soddisfa la sua specifica funzionale?)
  - Affidabilità (correttezza nel tempo)
  - Efficienza (spreca risorse?)
  - Integrità (danneggia dati o risorse?)
  - Resilienza (capacità di recupero da situazioni anomale)
- Legati alla capacità di subire modifiche
  - Manutenibilità
  - Flessibilità
- Legati all'adattabilità a nuovi ambienti
  - Portabilità
  - Riusabilità
  - Interoperabilità
  - Usabilità (quanto è fruibile in contesti diversi?)

Lo standard ISO/IEC 9126 per la software quality



NB: Questo standard è stato sostituito da ISO/IEC 25010:2011

Qualità del software

- Qualità funzionali: grado di soddisfazione dei requisiti **funzionali**, valutato mediante **testing**
- Qualità strutturali: grado di soddisfazione dei requisiti **non funzionali** da parte dell'**architettura**, valutato mediante analisi architetturali

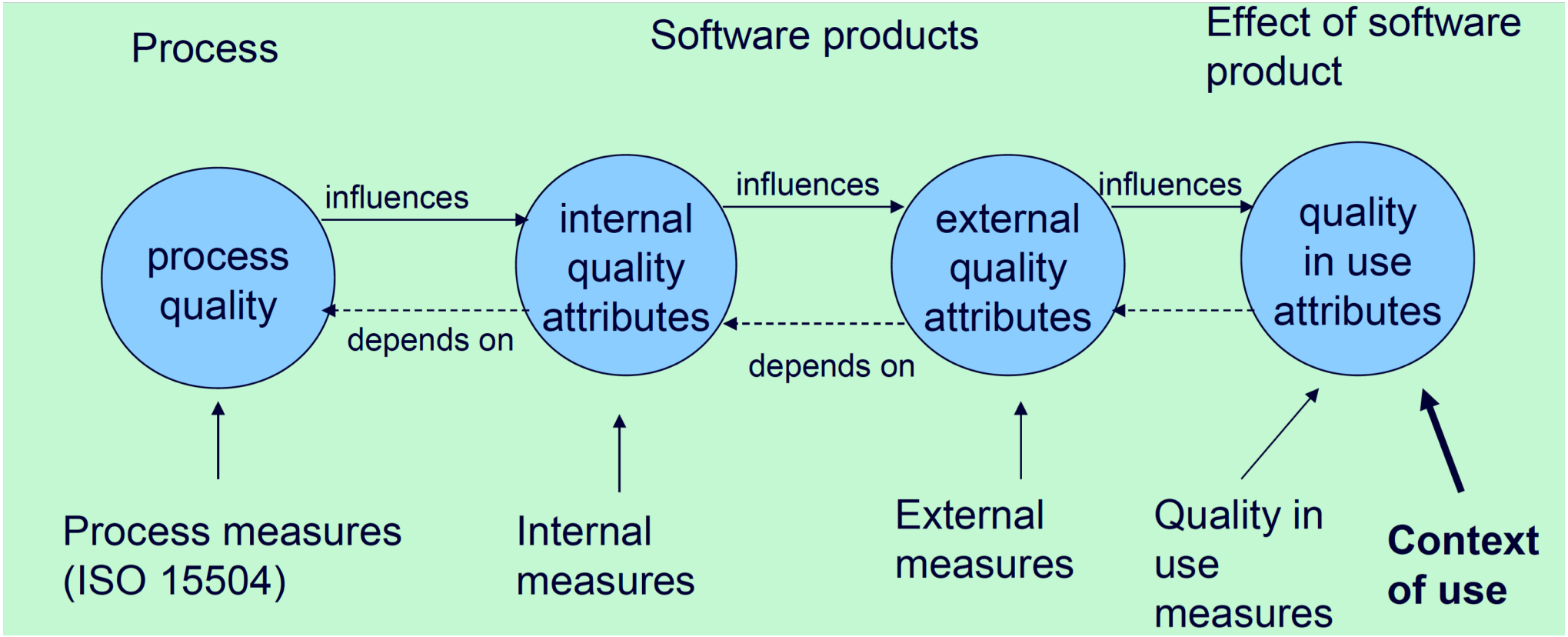
Principio di Tom DeMarco

Non puoi controllare quel che non puoi misurare!

Controlling Software Projects, Management Measurement & Estimation, 1982

Misurare le qualità del sw

La qualità del software si misura in molti modi



Quanto è importante misurare?

Progetti	Con misurazioni	Senza misurazioni
puntuali	75%	45%
In ritardo	20%	40%
cancellati	5%	15%
Rimozione difetti	95%	<85%
Stima risorse necessarie	Accurata	Ottimistica
Soddisfazione del cliente	Alta	Bassa
Morale del team	Alto	Basso

Fonte: Capers Jones, Measurement, Metrics and Industry Leadership, 2009 e Software Engineering Best Practices, McGraw Hill, 2010

Ragioni per cause legali sul software

Requisiti instabili, modificati continuamente	**95%**
Controllo di qualità inadeguato, senza misurazioni	90%
Cattivo monitoraggio del processo di sviluppo	85%
Stime sbagliate dei costi o dei tempi	80%
False promesse dei venditori	80%
Stime ottimistiche o arbitrarie	75%
Sviluppo informale, non strutturato	70%
Clienti inesperti incapaci di definire i propri requisiti	60%
Project manager inesperti	50%
Mancato uso di tecniche di analisi statica o ispezioni	45%
Riuso di software con errori	30%
Team di progettisti inesperto o non qualificato	20%

Fonte: Capers Jones

A chi interessano le misure del sw

- Sviluppatori individuali:
  - Distribuzione dello sforzo
  - Durata e sforzo a preventivo (stimati) e a consuntivo
  - Codice coperto da testing di unità
  - Numero di difetti trovati dal test di unità
  - Complessità del progetto e del codice
- Team di progetto
  - Dimensione del prodotto
  - Distribuzione dello sforzo
  - Stato dei requisiti (#approvati, #implementati, #verificati)
  - % dei casi di test superati
  - Durata stimata e reale tra due milestone principali
  - Livelli di staff stimati e reali
  - #difetti trovati dai test di integrazione e di sistema
  - #difetti trovati dalle ispezioni
  - Stato dei difetti
  - Stabilità dei requisiti (#requisiti modificati durante lo sviluppo)
  - Numero di task pianificati e completati
- Organizzazione che sviluppa software
  - Livelli dei difetti rilasciati (*critical*, *major*, *average*, *minor*, *exception*)
  - Tempo di ciclo di sviluppo del prodotto
  - Accuratezza della pianificazione e dello sforzo stimati
  - Riuso effettivo
  - Costo preventivato e reale

Misurare la qualità

In un qualsiasi ciclo di vita del software le entità visibili e **misurabili** sono i processi (le attività), le risorse impiegate e alcuni attributi dei prodotti

- **Qualità di un prodotto software:**

La qualità di un prodotto software è la misura in cui il prodotto soddisfa la sua specifica

- **Attributi di qualità interni ed esterni**

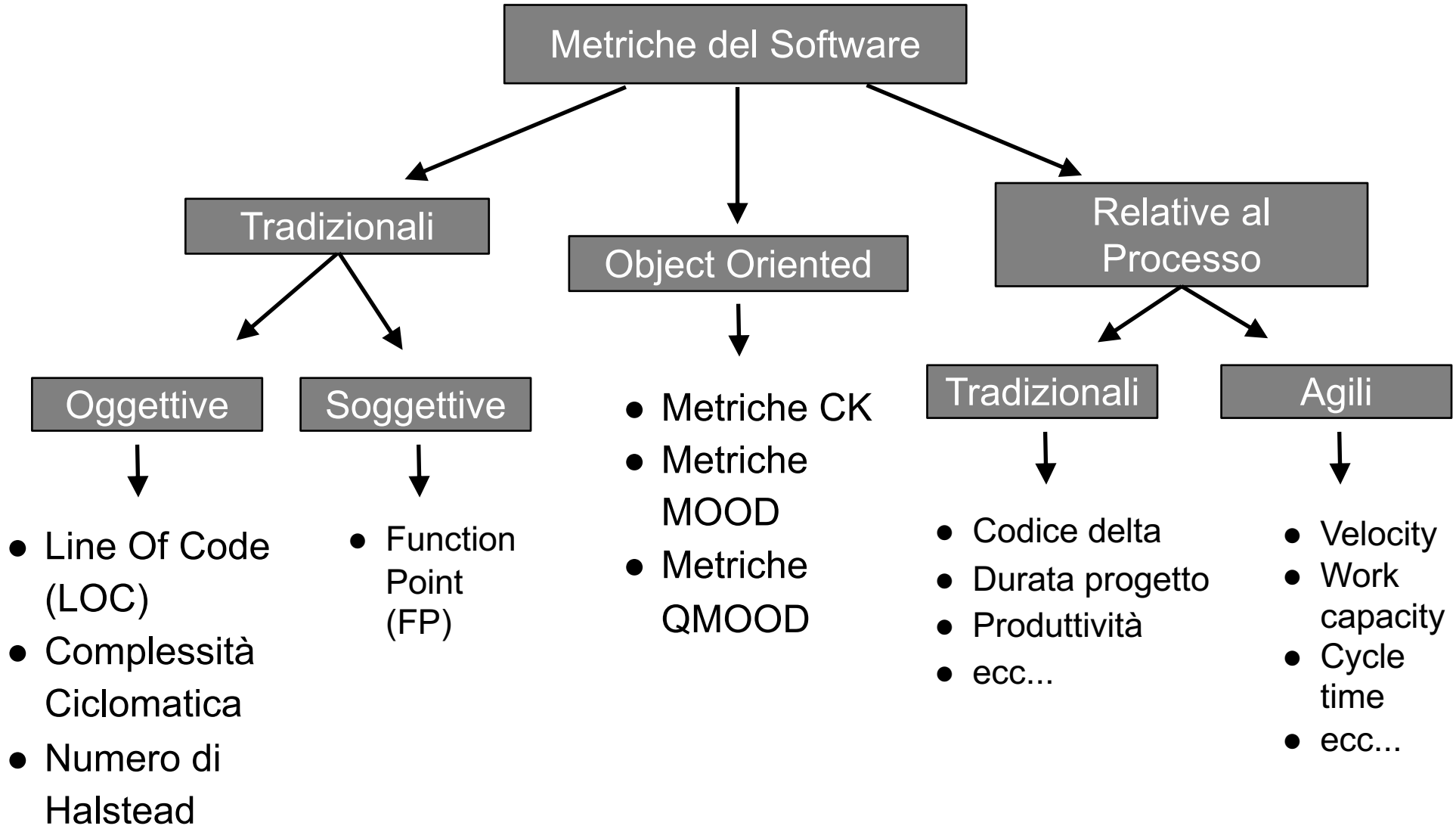
  attributi **esterni**: visibili all'utente

  attributi **interni**: visibili ai costruttori

Esempi di misure usate in Sw Eng

codice	lunghezza	LOC
	funzionalità	Function Points
	complessità	Indice McCabe
specifiche	lunghezza	#pagine
	riuso	#pagine
codifica	sforzo	Mesi/persona
testing	Fase rilevazione	%difetti trovati
	volume	#test schedulati
manutenzione	Costo medio	€/difetto

Classificazione



Tipi di problemi nel software

- **Failure**: comportamento del sw non previsto dalla sua specifica
- **Fault**: difetto del sorgente (detto anche *bug*), causa di un failure
- **Error**: causa di un difetto; esempio: un errore umano d'interpretazione della specifica o nell'uso di un metodo

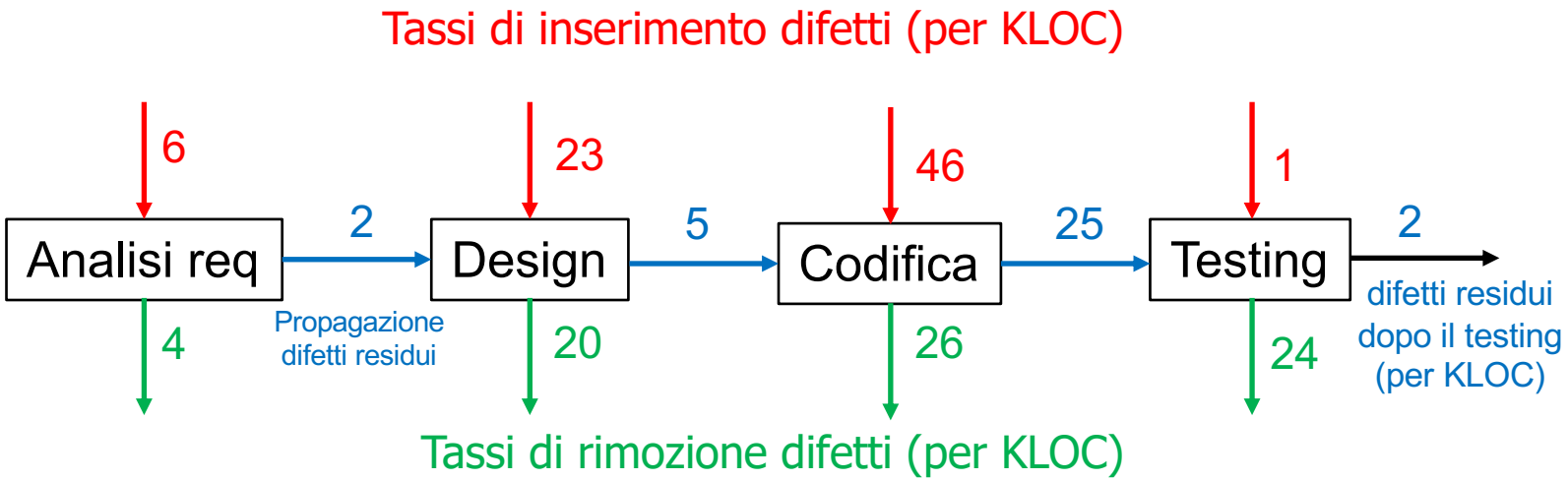
Da IEEE Standard Glossary of SE Terminology

Perché il sw è difettoso?

- Gli esseri umani commettono errori, specie quando eseguono compiti complessi; ciò è inevitabile
- I programmatori esperti commettono in media un errore ogni 10 righe
- Circa il 50% degli errori di codifica vengono catturati a tempo di compilazione
- Altri errori vengono catturati col testing
- Circa il 15% degli errori sono ancora nel sistema quando viene consegnato al cliente

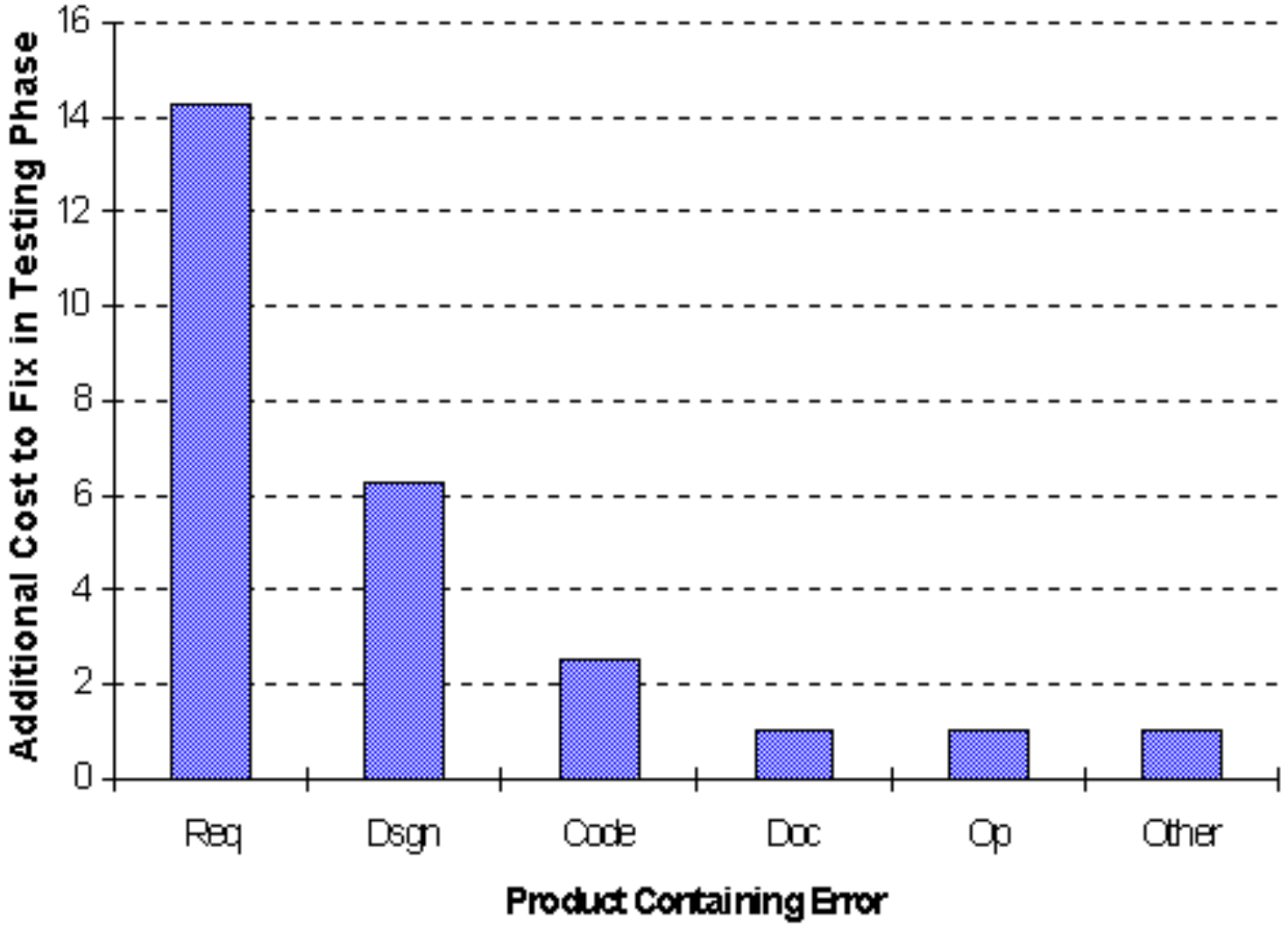
(W. Humphrey: „What if your life depended on software?" EuroSPI conference, Copenhagen, April 2000)

Da dove arrivano i difetti?

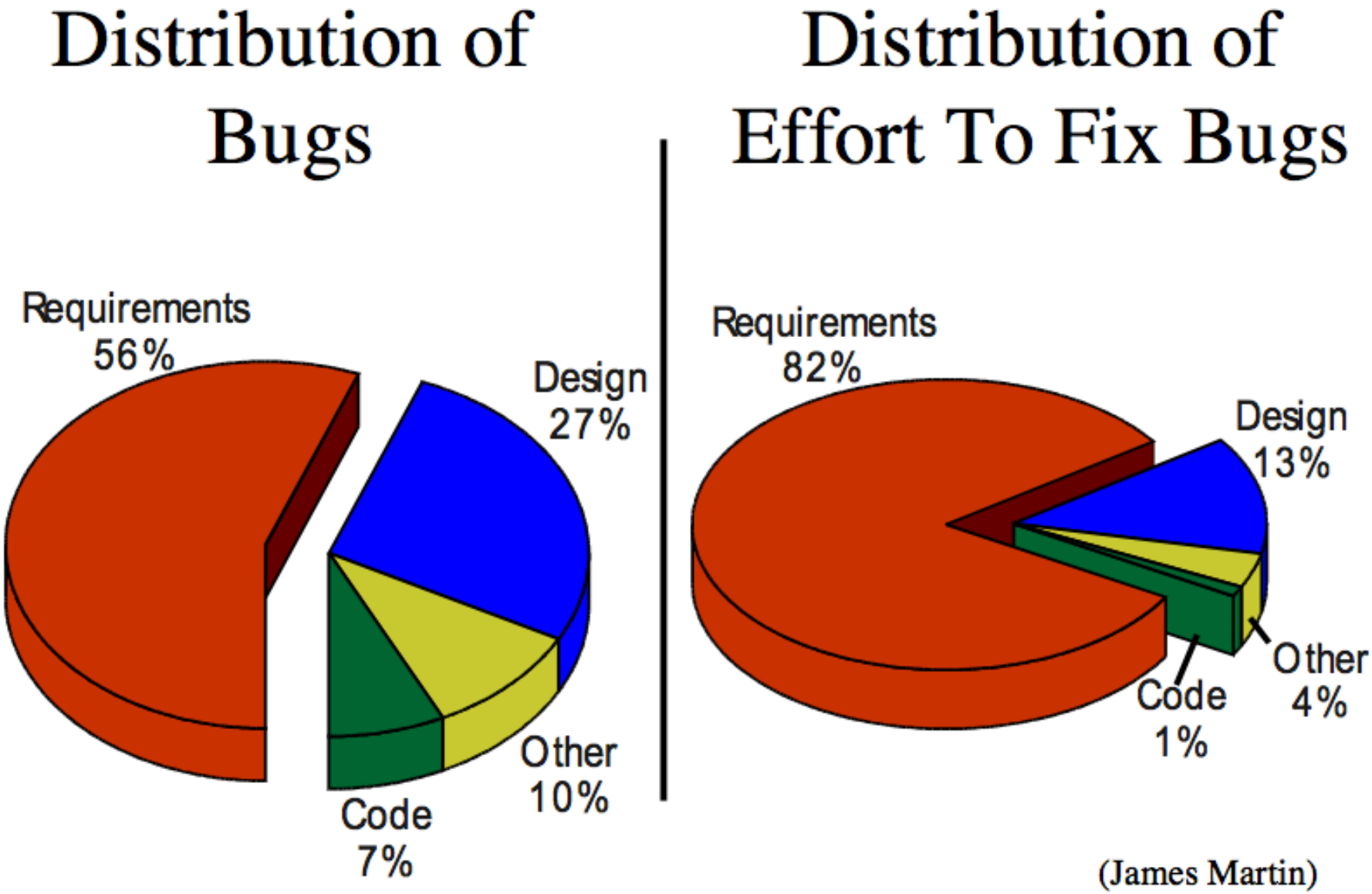


Fonte: Eick & Loader, ICSE 1992

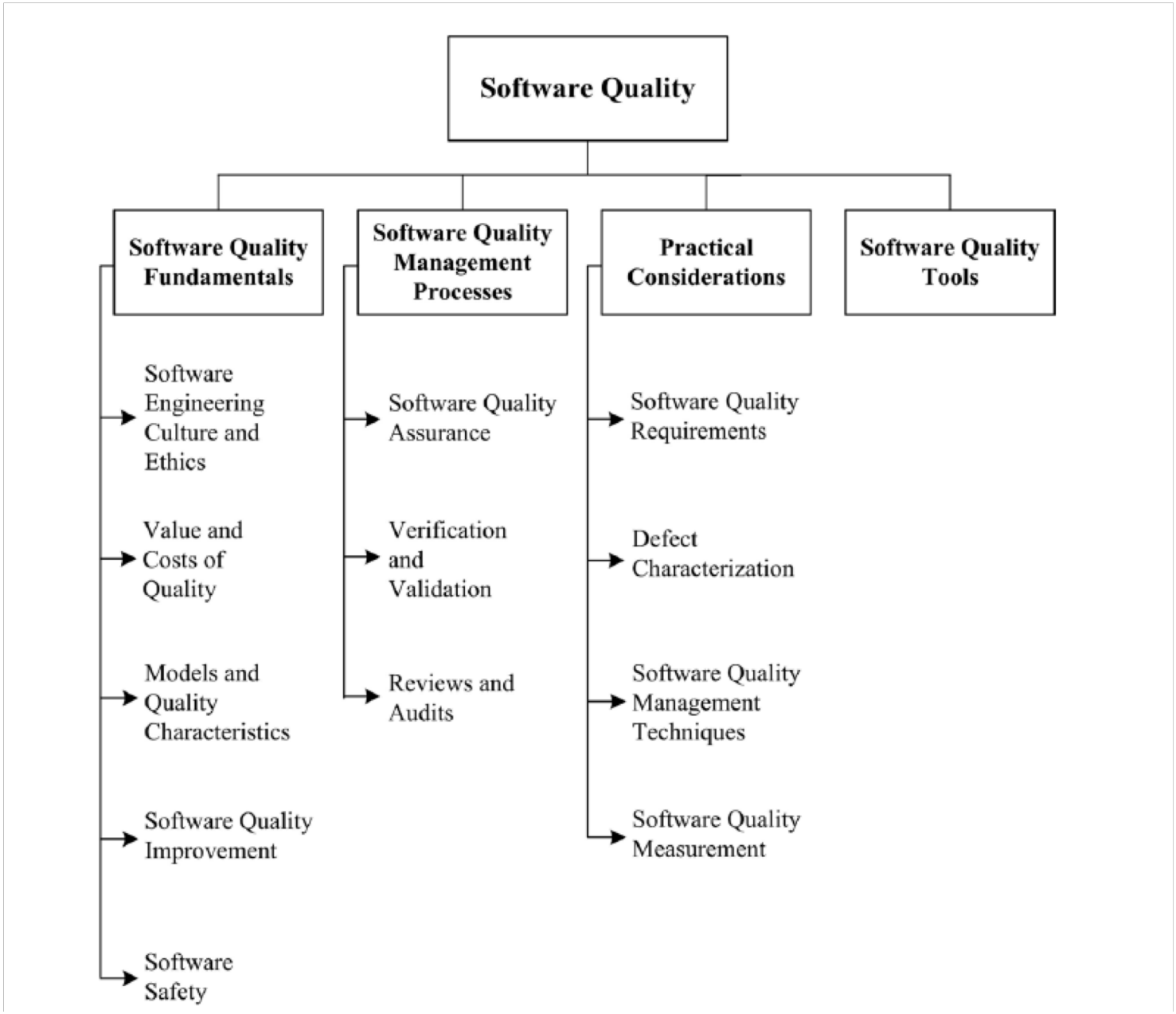
Costo relativo di correzione nella fase di testing di difetti introdotti in altre fasi



Lo sforzo per correggere i difetti



La qualità del sw nel SWEBOK



Il problema della qualità del sw

- Assicurare la **qualità** di un prodotto o servizio è difficile; nel caso del software valutarla o garantirla è particolarmente complesso
- Esistono **attività** legate alla qualità del sw (es. testing), **metodi** orientati alla qualità "di **prodotto**" (es. Cleanroom), **metodi** orientati alla qualità "di **processo**" (es. ISO9000), **metamodelli** di processo orientati alla qualità (es. GQM)

Metodi per aumentare la qualità del software

Esistono almeno tre classi di metodi che promuovono la qualità del sw:

- metodi convenzionali, orientati al prodotto, che cercano di aggiustare gli errori dopo che sono stati fatti e trovati
- metodi standard di *qualità totale*, come ISO 9000 e CMM, orientati alla qualità di processo
- metodi "formali", come Cleanroom e PSP, in cui le fasi del ciclo di sviluppo sono ridefinite una per una orientandole alla creazione di prodotti di qualità

Come introdurre la qualità

- Qualsiasi valutazione di qualità inizia dallo *scopo* che ha chi la vuole valutare
- Chi valuta la qualità di un prodotto o processo dovrebbe
  - avere chiari i propri obiettivi,
  - legarli a domande specifiche sui prodotti o processi oggetto di analisi, e
  - definire metriche capaci di analizzare e quantificare le qualità richieste ai prodotti rispetto agli obiettivi

Goal Question Metric (GQM)

- Metodo di definizione di metriche software
- Sviluppato per valutare i difetti software in progetti NASA, e poi generalizzato
- Passi:
  - Comprendere e analizzare gli obiettivi del progetto o organizzazione
  - Per ciascun obiettivo definire le domande cui occorre dare risposta onde poter capire se gli obiettivi sono stati raggiunti o meno
  - Stabilire quali attributi occorre misurare per poter rispondere alle domande

GQM

Il metodo **Goal-Question-Metric** (GQM) si usa per definire misure di progetto, processo e prodotto sw in modo che

- La misurazione sia semplice
- I dati di misurazione possano essere usati in modo costruttivo e condiviso
- Le metriche e la loro interpretazione riflettano i valori ed i punti di vista delle diverse parti interessate

goldpractice.thedacs.com/practices/gqm/

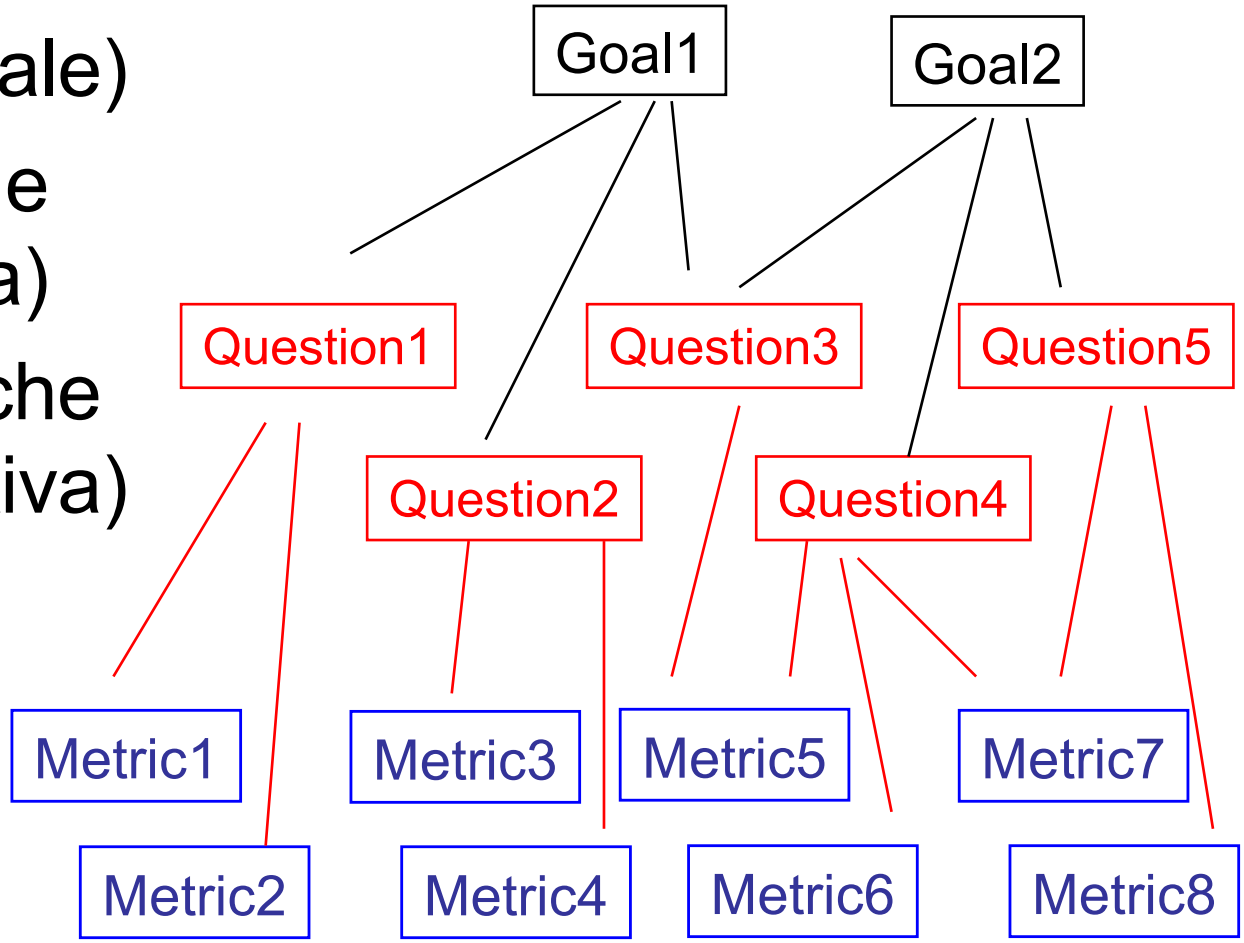
GQM

GQM si basa su tre livelli:

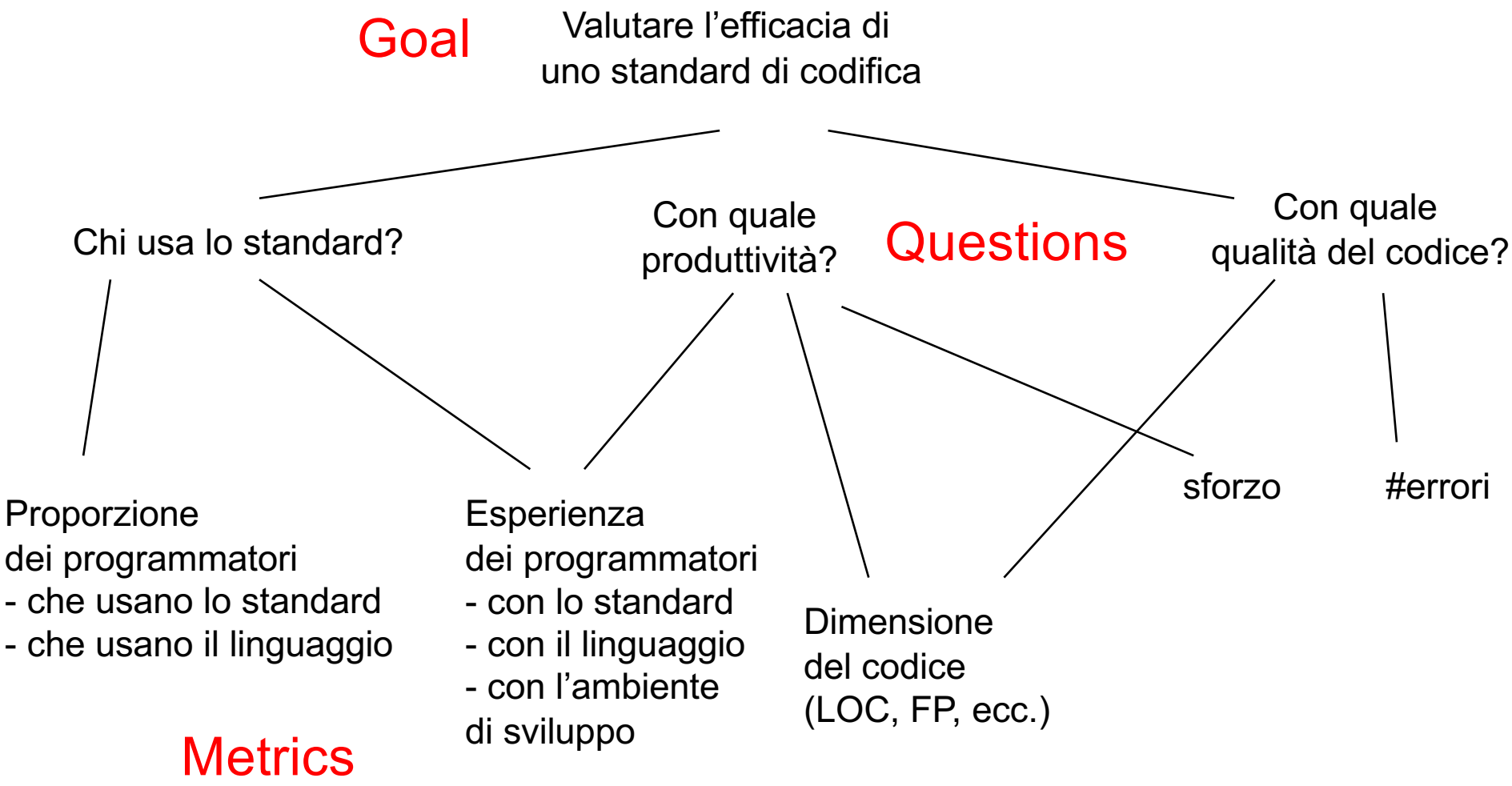
- Livello concettuale (goal): un obiettivo (goal) per un oggetto di studio viene definito rispetto a vari modelli di qualità e da vari punti di vista, relativamente ad un ambiente particolare
- Livello operativo (question): Si usa un insieme di domande per definire modelli dell'oggetto di studio atti a valutare l'obiettivo richiesto
- Livello quantitativo (metric): Un insieme di metriche basate sui modelli viene associato ad ogni domanda in modo da caratterizzare le risposte in modo misurabile

GQM: gerarchia di modellazione

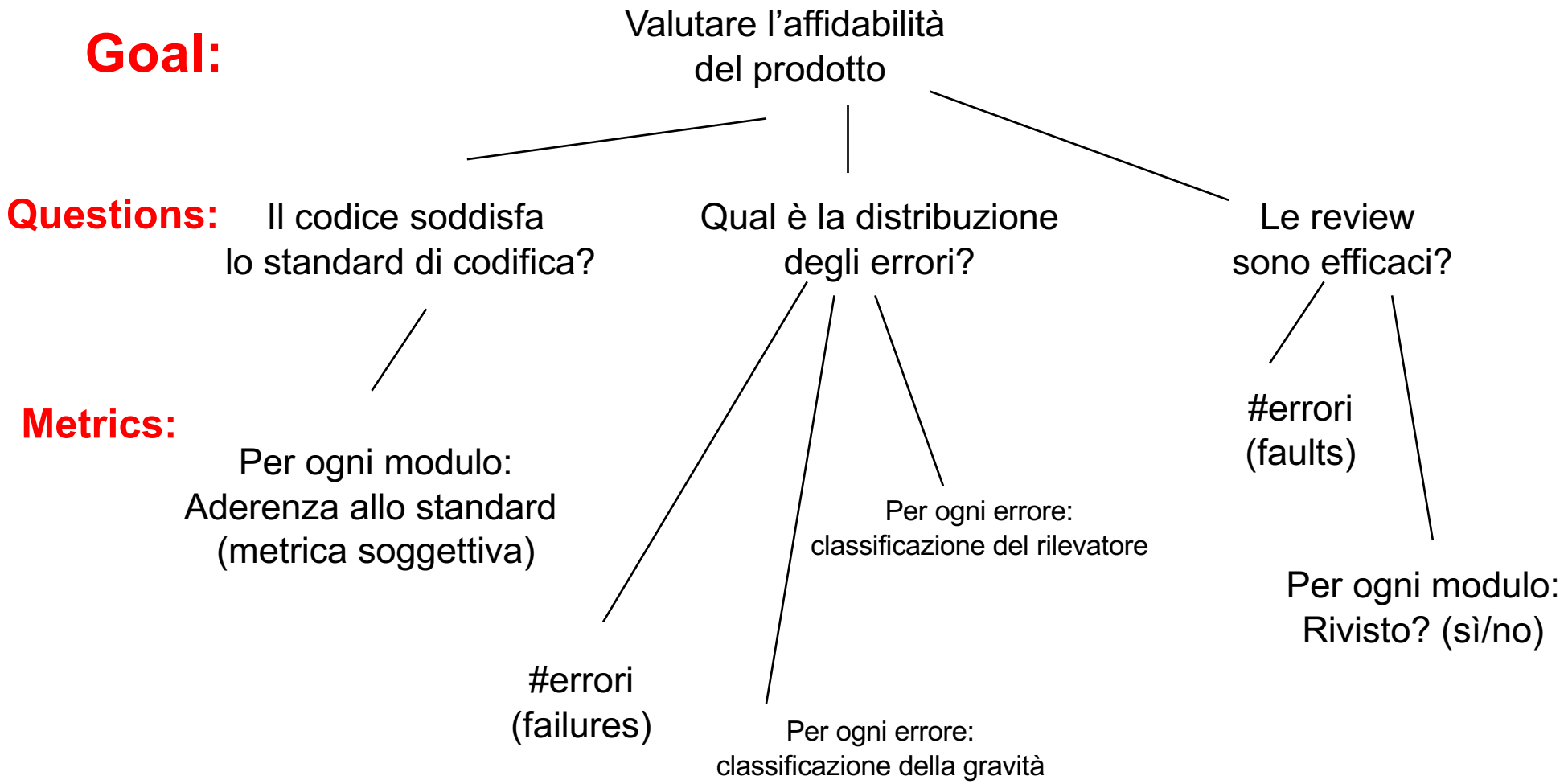
- Prima i goal (fase concettuale)
- Poi le domande (fase operativa)
- Infine le metriche (fase quantitativa)



Usare GQM. Esempio 1



Usare GQM: Esempio 2



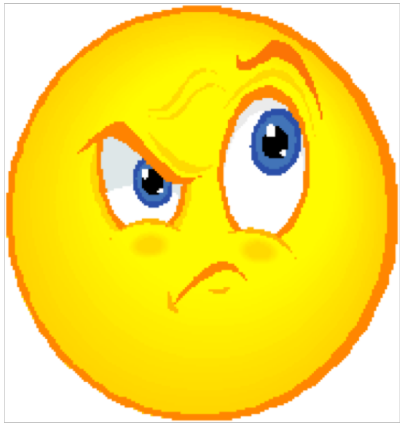
Usare GQM

- GQM si può usare in tutte le le fasi di sviluppo sw
- Si può applicare ai progetti, ai processi ed ai prodotti
- Le metriche che vengono definite debbono correlarsi agli obiettivi dell'organizzazione
  - Le misurazioni dovrebbero essere utili e costruttive, perché l'organizzazione deve imparare analizzandoli
  - Le metriche e le loro definizioni dovrebbero riflettere il punto di vista di diverse parti interessate (es. sviluppatori, utenti, progettisti, ecc.)

Pianificare un modello di qualità con GQM

1. Sviluppo dei goal e delle misure associate di qualità
2. Generazione di domande che definiscono i goal, quantificandoli
3. Specificare le misure da collezionare in conformità ai goal
4. Sviluppare i meccanismi operativi di collezione delle misure
5. Raccogliere i dati e analizzarli onde sviluppare azioni correttive
6. Analizzare i dati postmortem per raccomandazioni sul futuro

Esercizio



Pianificare con GQM

- Le misure di qualità di una festa
- Le misure di qualità di chi organizza una festa
- Le misure di qualità di chi partecipa ad una festa

Valutare una presentazione

- Qualità formale della presentazione
  - Uso di figure originali
  - Errori nel testo
  - Semplicità ed eleganza grafica
- Attinenza coi temi del corso di Ingegneria del sw
  - Presenza di riferimenti ad argomenti del corso
  - Qualità del gergo utilizzato
- Ampiezza dell'ambito della presentazione
  - Pubblicazioni consultate (in bibliografia)
  - Almeno una citata, almeno una citante
- Complessità dell'articolo scelto
  - Provenienza (quale rivista o conferenza)
  - Numero di pagine dell'articolo originale

Discussione

Come usare GQM per valutare la qualità della presentazione che mi manderete?

Cosa si intende per “qualità”?

I punti problematici per un sistema software

- La qualità non è soltanto assenza di difetti del prodotto finale
- Spesso i requisiti sulle qualità esterne (es. efficienza, affidabilità) e quelli sulle qualità interne (es. mantenibilità, riusabilità) sono in antitesi, e vanno bilanciati
- Alcuni requisiti di qualità sono difficili da specificare
- Le specifiche sono spesso **incomplete** e a volte **inconsistenti**, ma non possiamo aspettare che le specifiche migliorino prima di preoccuparci della qualità del prodotto finale

Obiettivi della verifica e della validazione

- **Verifica**:

Confronto di un prodotto con la sua specifica

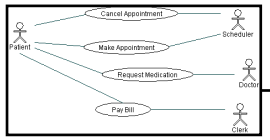
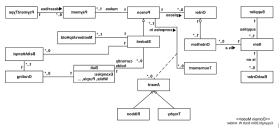
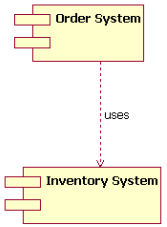
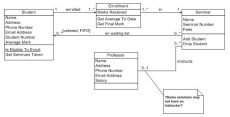
(ovvero, confronto di un prodotto con i suoi requisiti)

- **Validazione**:

Accettazione del prodotto da parte del committente

Nota: Secondo alcuni, la “verifica” determina se una certa attività è stata effettuata correttamente, mentre la “validazione” certifica che un prodotto soddisfa i suoi requisiti; non NON usiamo queste accezioni

Attività di sviluppo del software

Raccolta dei requisiti	Analisi dei requisiti	System Design	Object design	Implementazione	Testing
	espressi mediante	mappati su	realizzati con	definiti da	verificati con
				Class A… Class B… Class C…	
Modello dei casi d'uso	Oggetti del dominio	Sotto-sistemi	Oggetti del sistema	Sorgente	Casi di test

Verifiche

- L'attività di *verifica*, così come quella di *documentazione*, non è una fase separata del processo
- Tuttavia è opportuno che sia effettuata da persone diverse da quelle coinvolte nel design o nella codifica
- Ogni documento prodotto dovrebbe essere controllato (possibilmente da persone diverse dagli autori del documento) e sistematicamente documentato esso stesso
- Esistono due tipi di verifica: quella basata sull'analisi (ispezione) e quello basata sull'esecuzione (testing)

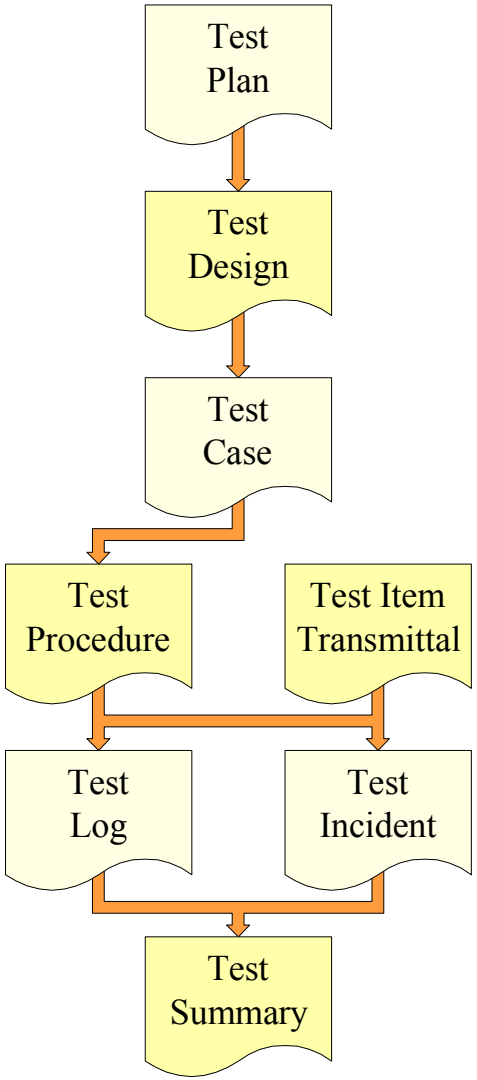
Attività legate alla qualità

- **Testing**: processo di investigazione sui rischi connessi all'esecuzione di un sistema software
- **Misurazione**: di indicatori di qualità, sia mediante ispezione sia mediante esecuzione
- **Verifica**: analisi delle funzioni rispetto alla specifica
- **Validazione**: accettazione da parte degli stakeholder
- **Certificazione**: analisi delle funzioni rispetto ai requisiti di legge da certificare

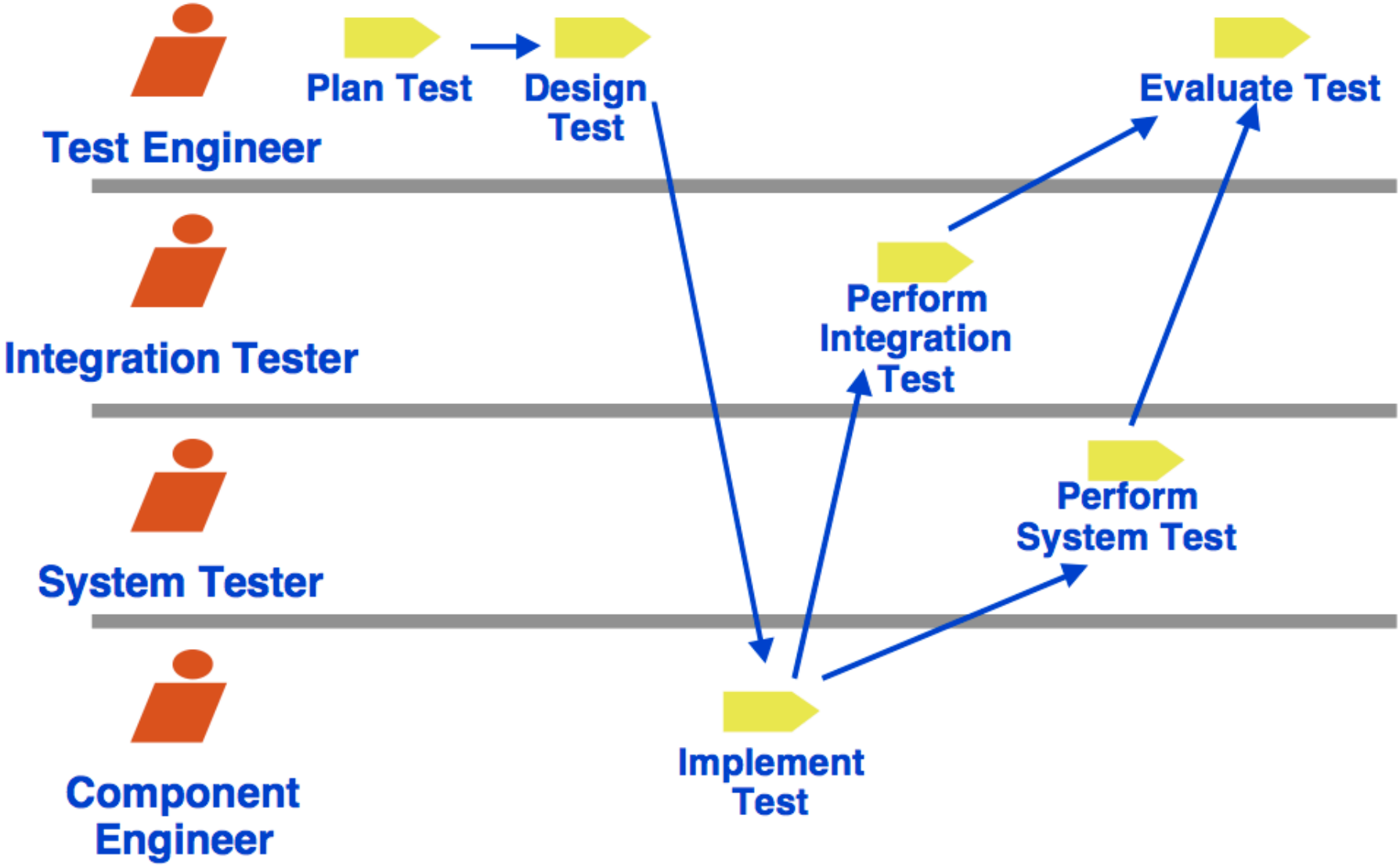
Testing

- Il testing di un prodotto software è un'attività di processo che ha lo scopo di misurare i rischi connessi all'uso di un prodotto software in esecuzione
- Gli artefatti prodotti dal testing sono i piani di test che includono i casi di test, le suite di test che automatizzano il testing, e i rapporti di test che contengono i risultati dell'attività

Artefatti di testing



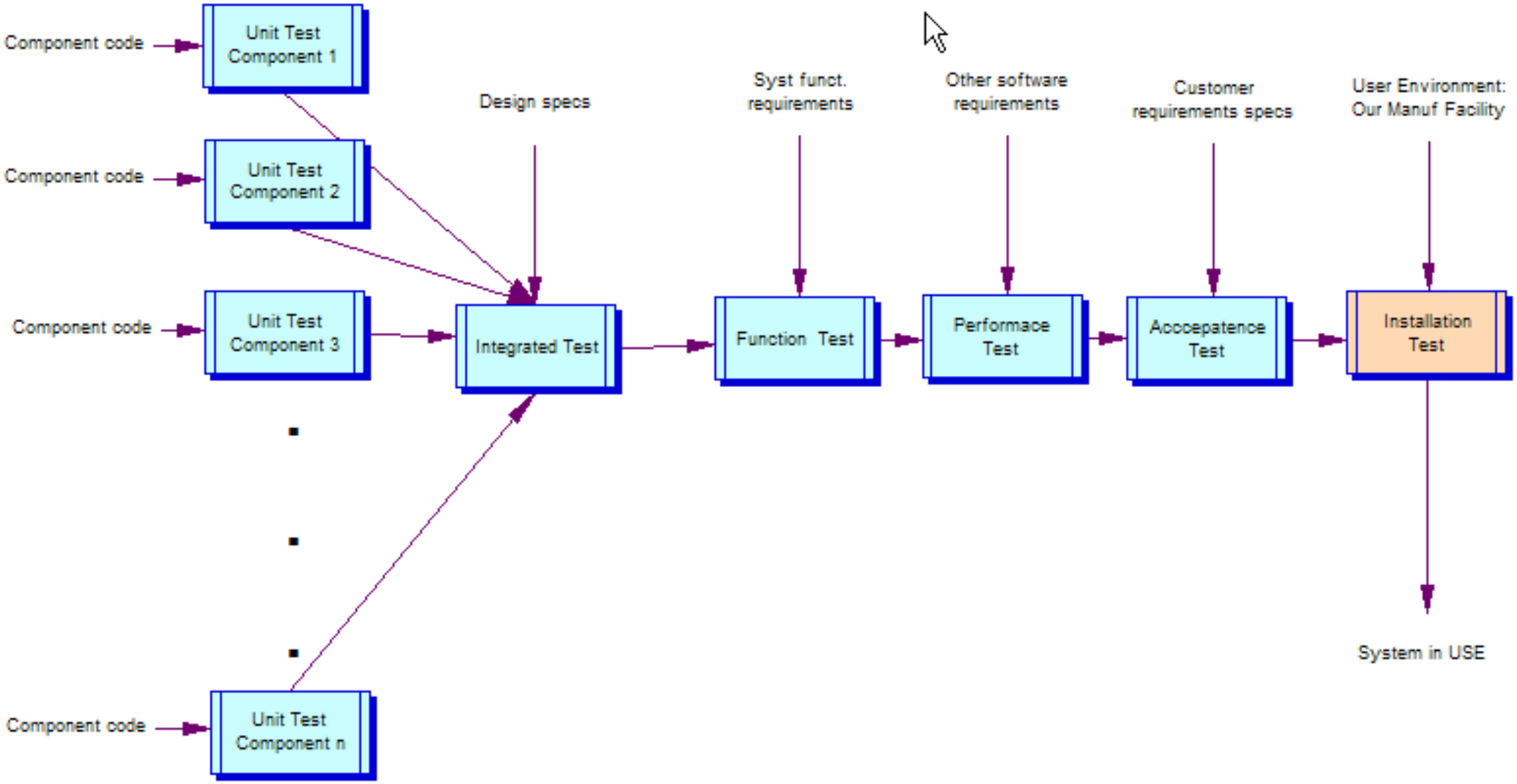
Workflow di testing



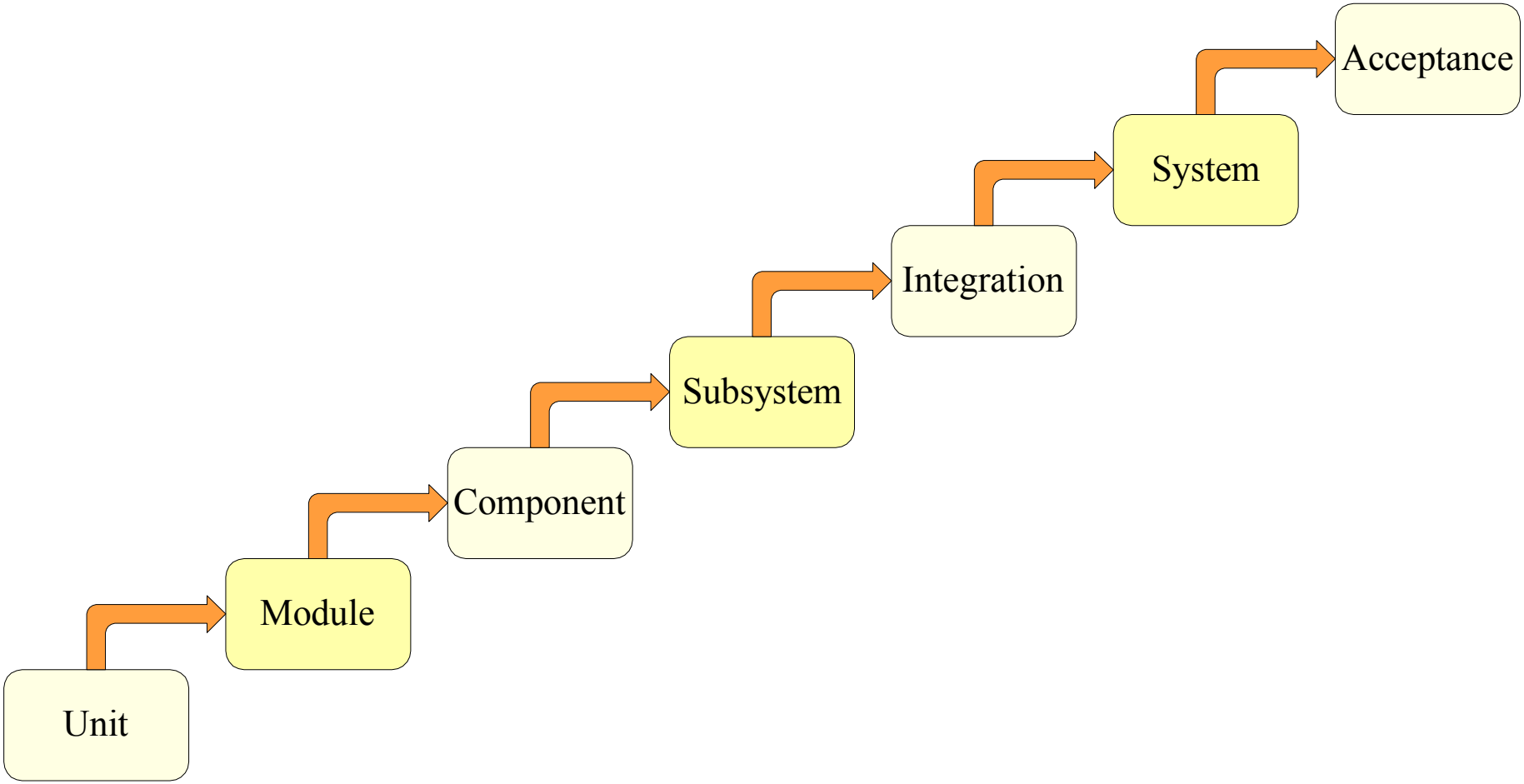
Fasi di testing

- **Unit testing**
  - Controllo di componenti individuali
- **Module testing**
  - Controllo di collezioni di componenti correlati
- **Sub-system testing**
  - Controllo di moduli integrati: attenzione alle interfacce
- **System testing**
  - Controllo dell'intero sistema. Controllo delle proprietà "emergenti" (non attribuibili a singole componenti)
- **Acceptance testing**
  - Testing con dati (e presenza) del cliente per controllare che il comportamento del sistema sia accettabile

Tipi di testing



Tipi di testing



Testing: c’è sempre un altro errore

“Il testing si può usare per provare la presenza di errori in un programma, mai per dimostrarne l’assenza.”

—Edsger W. Dijkstra, 1972

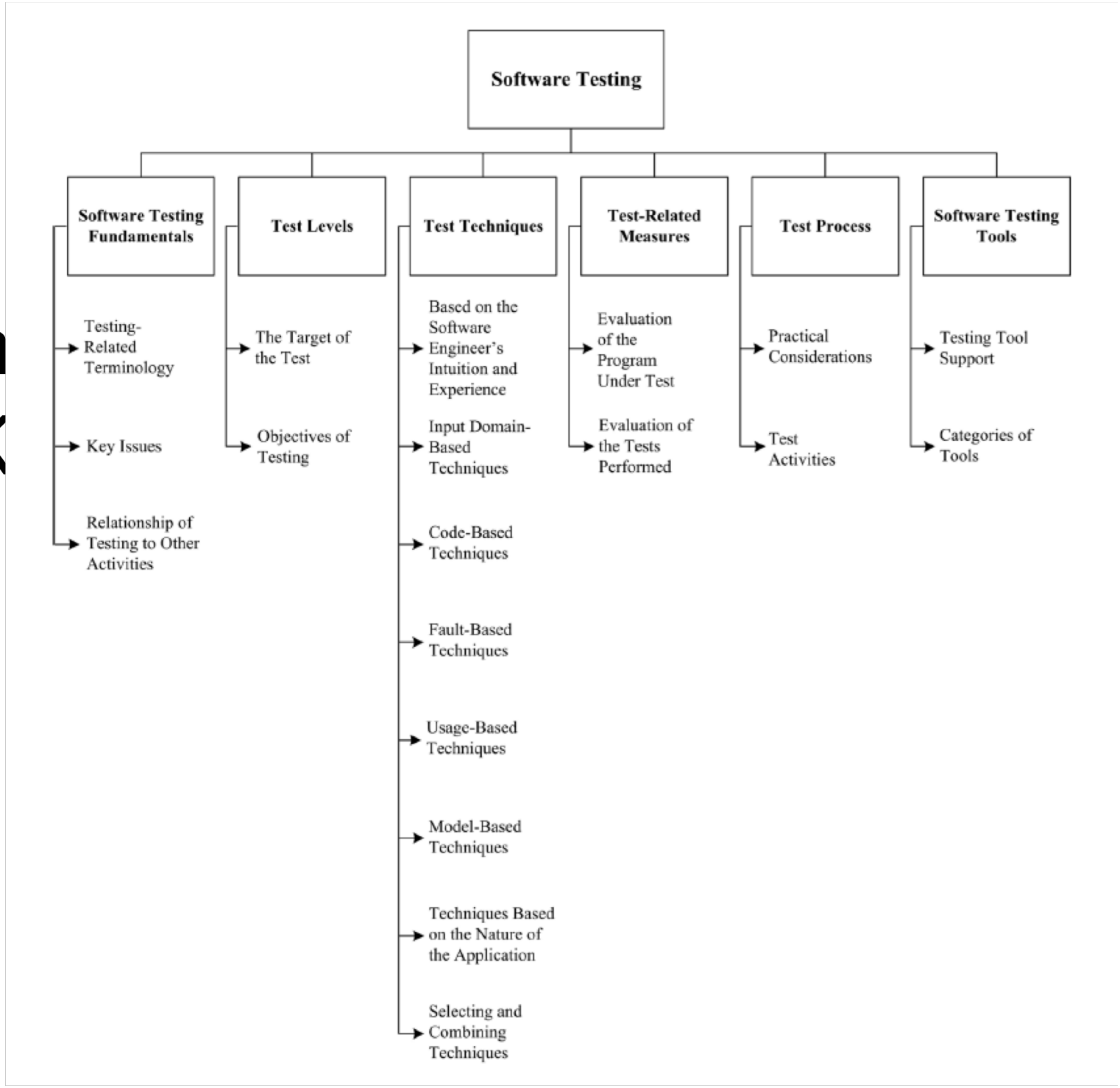
Testing

"Se l'obiettivo è mostrare l'**assenza** di errori, ne troveremo pochi;

Se l'obiettivo è mostrare la **presenza** di errori, ne troveremo molti."

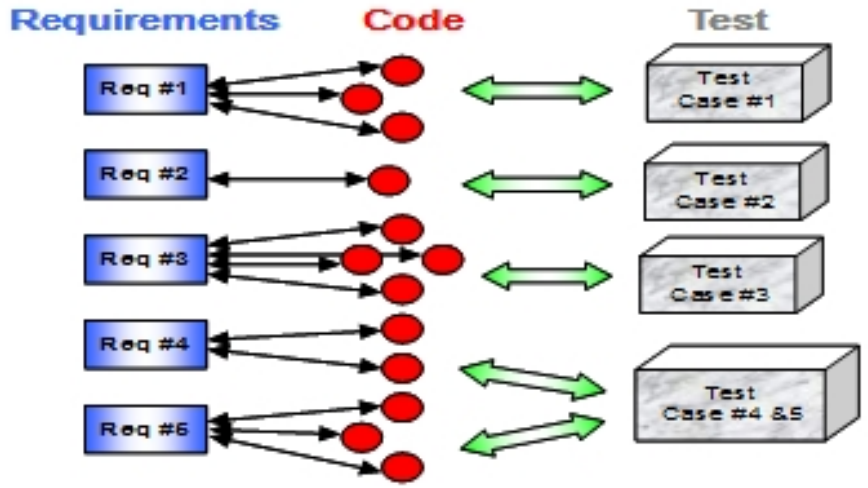
—G. J. Meyers, 1979

Testing in SWEBOK (cap 5)



Requisiti e test

- Il testing ha lo scopo di verificare che il codice soddisfi i requisiti



Il testing è difficile

- Non sempre i requisiti sono chiari
- Molti sviluppatori non conoscono né le tecniche di testing né i relativi strumenti
- Il testing viene ritenuto “noioso”
- Spesso non c’è tempo per testare bene “tutto”

Nota bene: Il testing sistematico ed esaustivo è computazionalmente *intrattabile;* dobbiamo però fare ogni sforzo per eliminare ogni possibile difetto o fattore di rischio

Definire un piano di test

- Completo (ma non eccessivo)
  - Usare una matrice di test
  - Includere test positivi e negativi
- Trovare i casi di test “interessanti”
- Eseguire ogni test almeno due volte
- Aggiornare il piano durante lo sviluppo
  - Aggiungere test di regressione
  - Sfruttare il feedback degli utenti
- Guardarsi dall’obsolescenza del prodotto e dei suoi test

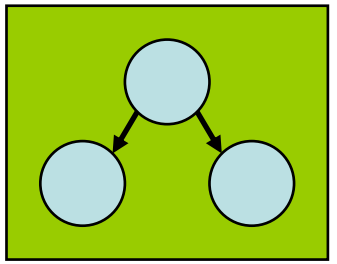
Sample traceability matrix

Requirement Identifiers	Reqs Tested	REQ1 UC 1.1	REQ1 UC 1.2	REQ1 UC 1.3	REQ1 UC 2.1	REQ1 UC 2.2	REQ1 UC 2.3.1	REQ1 UC 2.3.2	REQ1 UC 2.3.3	REQ1 UC 2.4	REQ1 UC 3.1	REQ1 UC 3.2	REQ1 TECH 1.1	REQ1 TECH 1.2	REQ1 TECH 1.3
Test Cases	321	3	2	3	1	1	1	1	1	1	2	3	1	1	1
Tested Implicitly	77														
TC1.1.1	1	×													
TC1.1.2	2		×	×											
TC1.1.3	2	×											×		
TC1.1.4	1			×											
TC1.1.5	2	×												×	
TC1.1.6	1		×												
TC1.1.7	1			×											
TC1.2.1	2				×		×								
TC1.2.2	2					×		×							
TC1.2.3	2								×	×					
TC1.3.1	1										×				
TC1.3.2	1										×				
TC1.3.3	1											×			
TC1.3.4	1											×			
TC1.3.5	1											×			
etc															
TC5.6.2	1														×

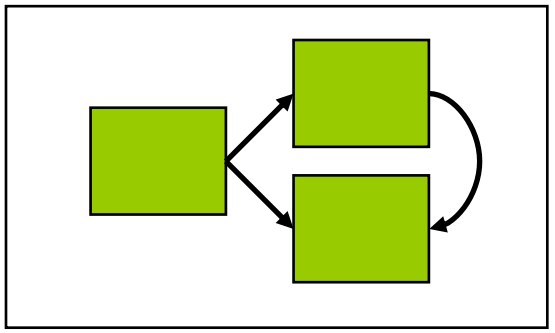
Tipi di test

- Unità o modulo
  - Classi o tipi individuali

- Componente
  - Gruppo di classi correlate



- Integrazione
  - Interazione tra componenti

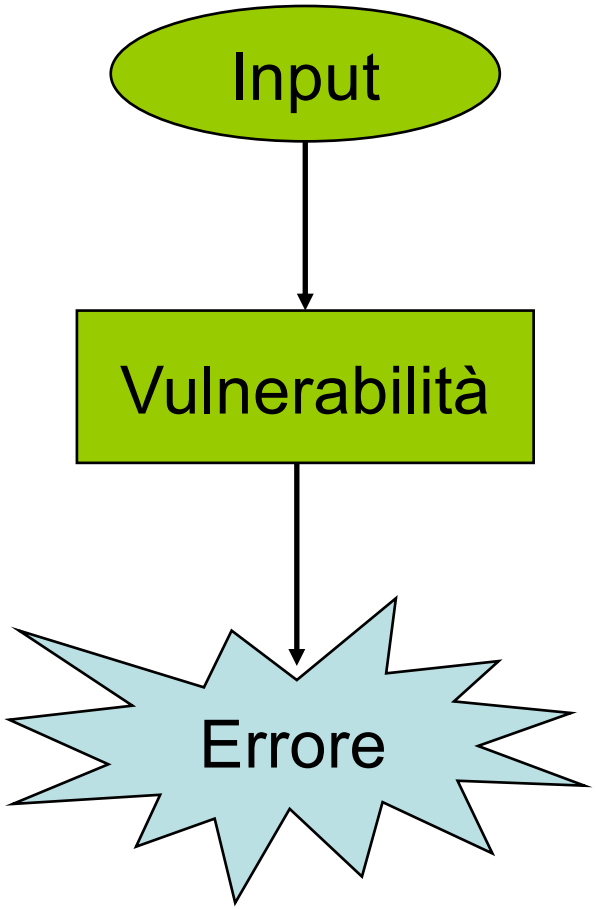


Progettare i test

- I test sono “attacchi” al software condotti per vedere se ci sono difetti o rischi
- Metodi di test
  - Diretti
  - Indiretti
- Cercare e colpire i punti vulnerabili
  - Black box
  - White box

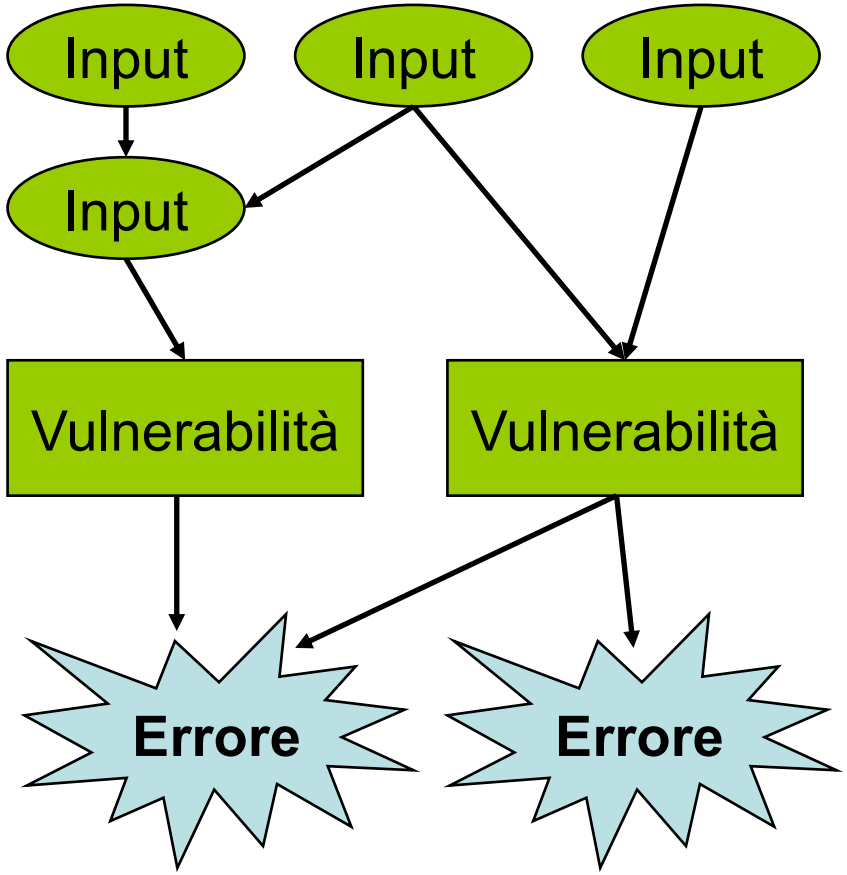
Testing diretto

- Di solito automatico
- Di basso livello
- Solo funzionalità base
- Sfrutta le specifiche



Testing indiretto

- Di solito manuale
- Di alto livello
- Scenari “realistici”
- Scopre molti errori imprevisti



Testing di un modulo

- **Black-box testing** (funzionale, data-driven, I/O-driven): i casi di test sono definiti nel documento di specifica; il codice del prodotto viene ignorato durante il testing.
- **White-box testing** (glass-box, logic-driven, path-oriented): i casi di test sono definiti sul codice sorgente.
- Nota bene: *Eseguire il testing esaustivamente (cioè in tutti i casi possibili) è impossibile*

Esempio (I/O-driven testing): se l'applicazione da testare ha 10 parametri di ingresso, e ciascuno può assumere 5 valori, occorre testare 10^5 casi di input.

Black Box

- Si sfrutta la semplicità apparente del software
  - Si fanno assunzioni sull'implementazione
  - Adatto per testare le interazioni tra componenti
- Testa le interfacce ed il comportamento

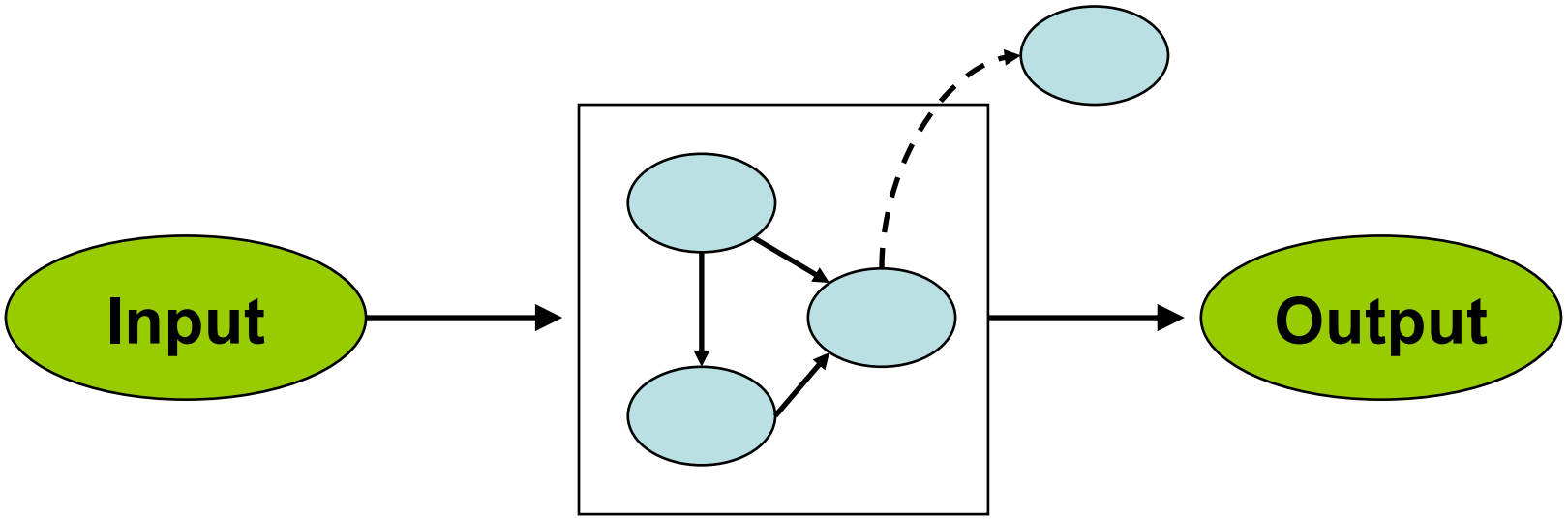
Tecniche di black-box testing

- Il black-box testing si effettua **senza** conoscere il sorgente.
- Consiste nell'usare la specifica del prodotto per predisporre un insieme di casi di test che massimizzi la probabilità di trovare un errore e minimizzi quella che due test diversi trovino lo stesso errore

- **Equivalence testing with boundary value analysis:** i possibili dati di input e/o di output (definiti dalla specifica) possono talvolta essere partizionati in classi di equivalenza; in questo modo si diminuiscono i casi di test necessari e si possono usare i "boundary values" (valori di confine di partizione) per guidare il test.

- **Functional testing:** dopo aver identificato nella specifica tutte le funzioni di un modulo, si definiscono i dati di test necessari per controllare ciascuna funzione separatamente.

White Box

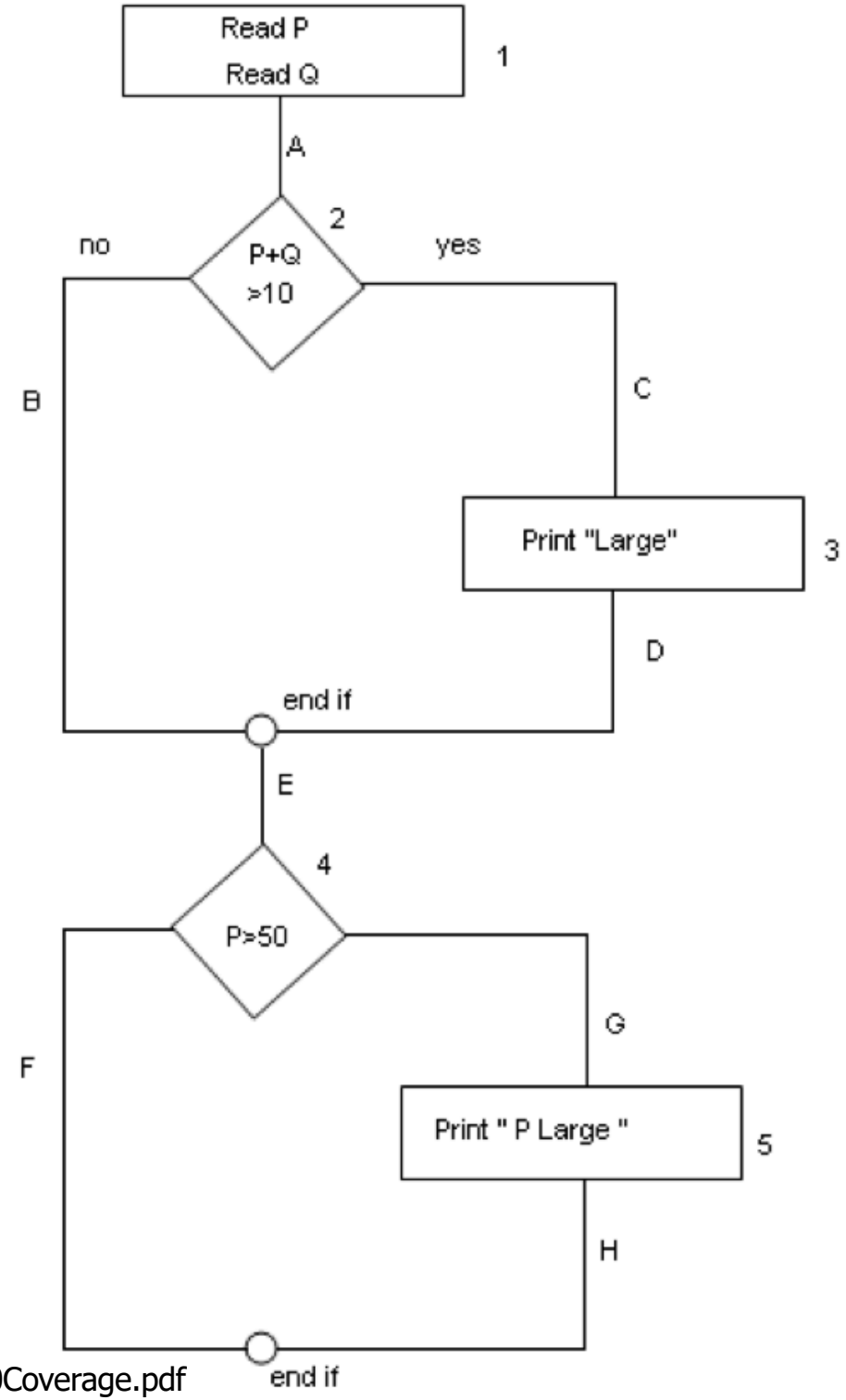


- Sfrutta la complessità interna del software
  - Occorre completa conoscenza dell'implementazione
  - Adatta a testare funzioni singole
- Testa l'implementazione ed il progetto

Tecniche di white-box testing

- **Statement coverage**: ogni comando viene eseguito almeno una volta (ma non c'è garanzia che ogni condizione venga testata).
- **Branch coverage**: ogni comando viene eseguito almeno una volta ed ogni condizione viene testata.
- **Path coverage**: ogni cammino possibile viene testato almeno una volta.

```
read P
read Q
if P+Q > 10 then
print "Large"
endif
if P > 50 then
print "P Large"
endif
```



Teorema di Weyuker

Dato un generico programma P sono indecidibili:

- Esiste un dato di ingresso di P che causa l'esecuzione di un particolare comando? indecidibile
- Esiste un dato di ingresso che causa l'esecuzione di una particolare condizione (branch)? indecidibile
- Esiste un dato di ingresso che causa l'esecuzione di un particolare cammino in P? indecidibile

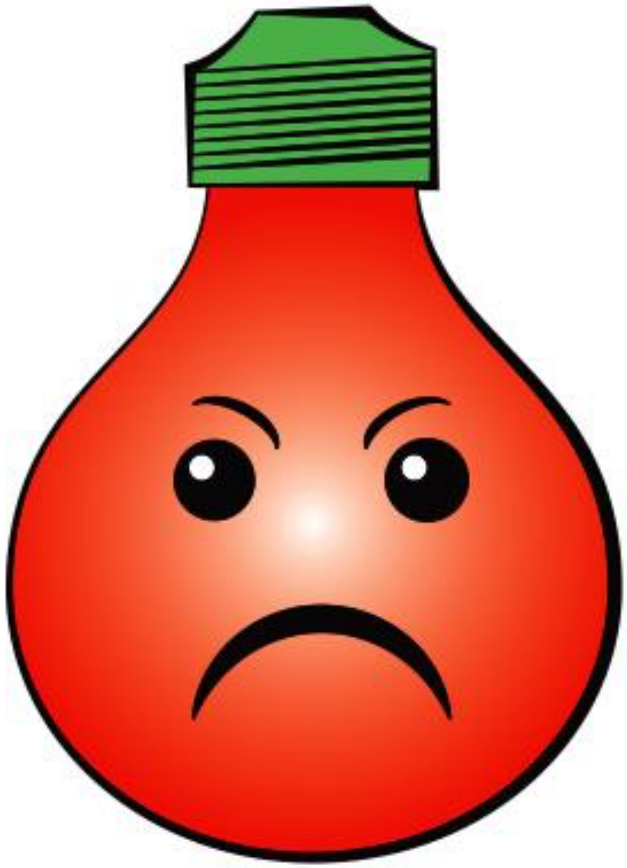
È però possibile individuare sottoproblemi decidibili

Processi orientati al testing: i metodi agili

I metodi “agili” mettono il testing prima della codifica (*Test Driven Development: TDD*): nessun modulo viene sviluppato prima che sia stato definito almeno un test di correttezza

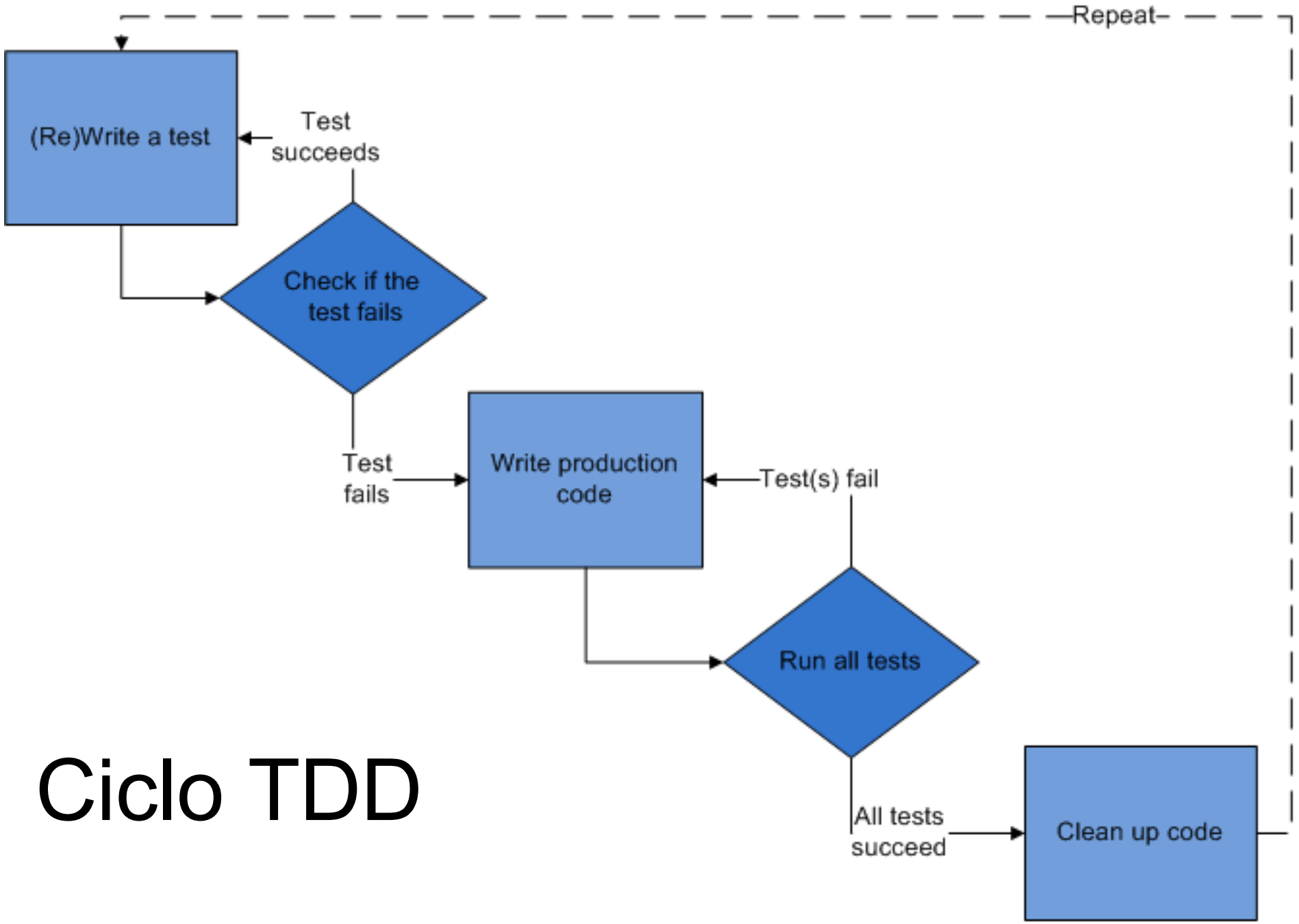
Ciclo:

- Test **rosso**: test di nuova funzione vuota; test che fallisce perché la funzione non esiste;
- Test **verde**: codice necessario per passare il test rosso;
- **Refactoring**: modifiche al codice di test verde per semplificarlo



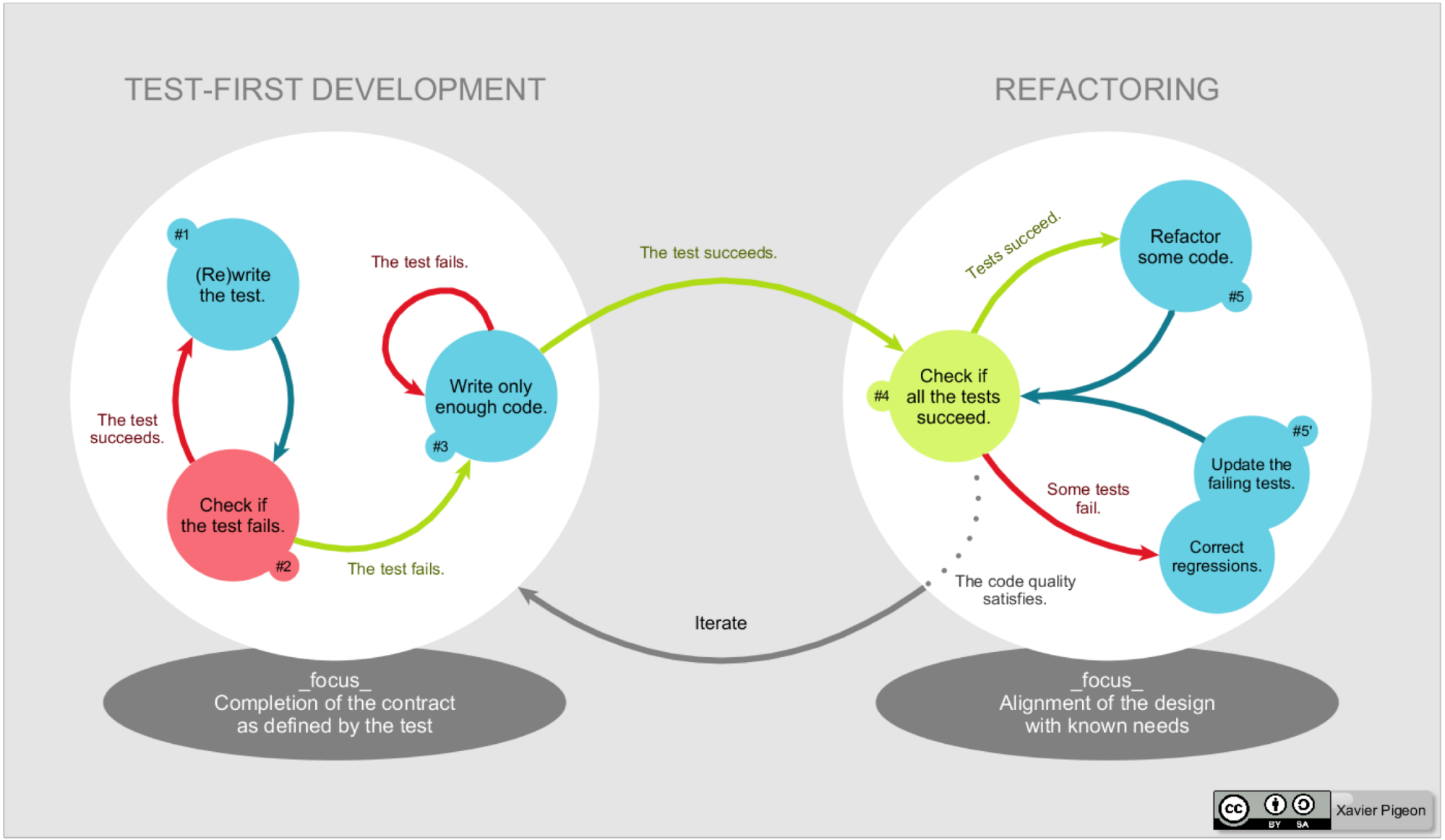
Debugging
Sucks!

Testing
Rocks!


Repeat
(Re)Write a test
Test succeeds
Check if the test fails
Test fails
Write production code
Test(s) fail
Run all tests
All tests succeed
Clean up code

Ciclo TDD

Test-first + refactoring



Crowdsourced testing

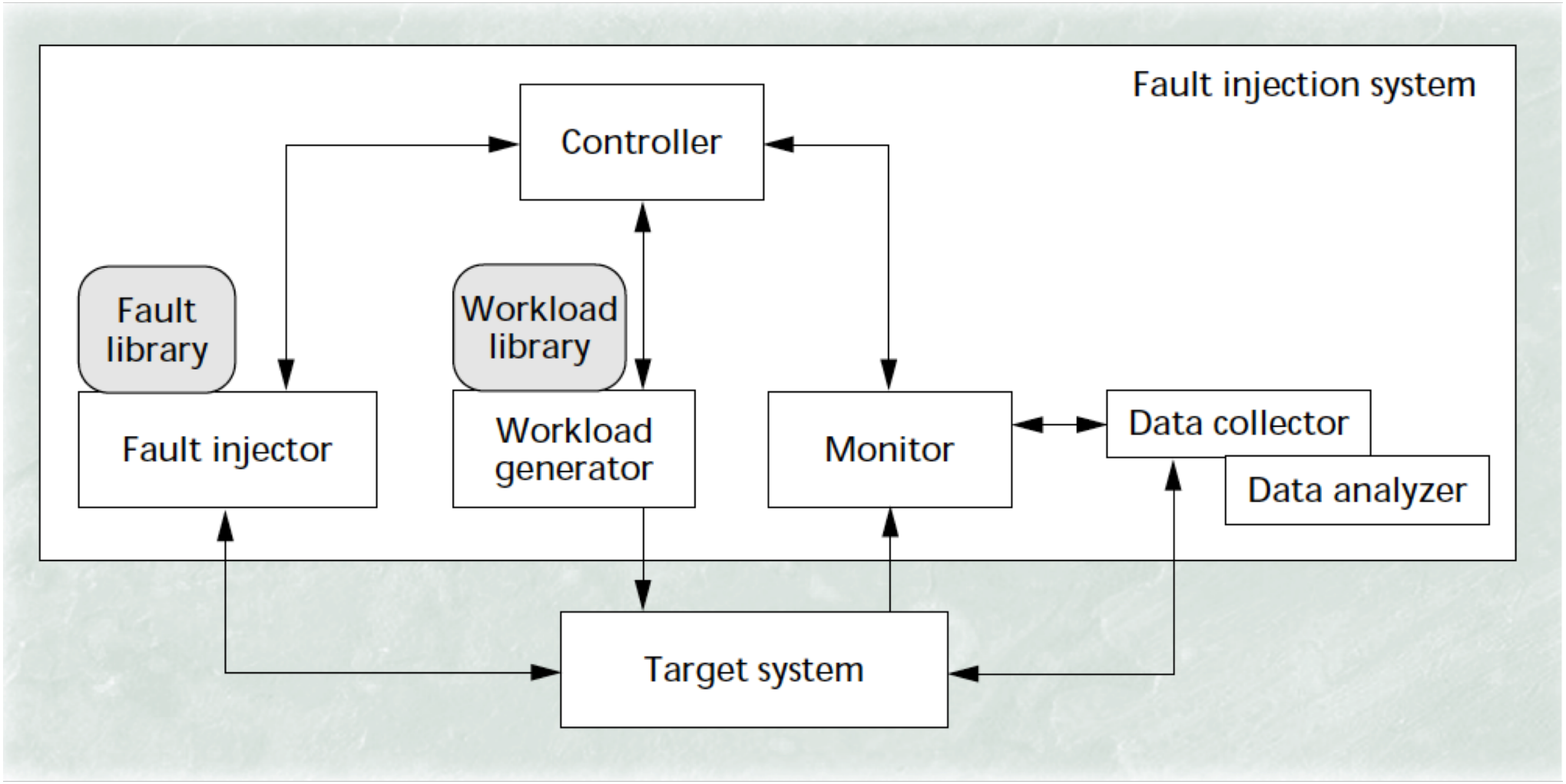
- Test gestito da non specialisti mediante infrastruttura cloud
- Utilizzato per software user-centric specie per valutarne l'usabilità
- Si pagano solo gli errori trovati

https://crowdsourcedtesting.com

Fault injection

- Una tecnica per migliorare il test coverage introducendo errori per testare in particolare il codice che gestisce tali errori, che altrimenti non sarebbe eseguito
- Esempio: nel test di un kernel di sistema operativo si può inserire un driver che intercetta le system calls e ritorna a caso un errore per alcune delle calls.

Fault injection



Hsueh, Mei-Chen, Timothy K. Tsai, and Ravishankar K. Iyer. "Fault injection techniques and tools." *Computer* 30.4 (1997): 75-82.

Strumenti di testing

- Bugzilla e Scarab
- Clover
- Junit
- sonarqube
- xUnit
- SilkTest (Borland)
- Selenium (per applicazioni web)

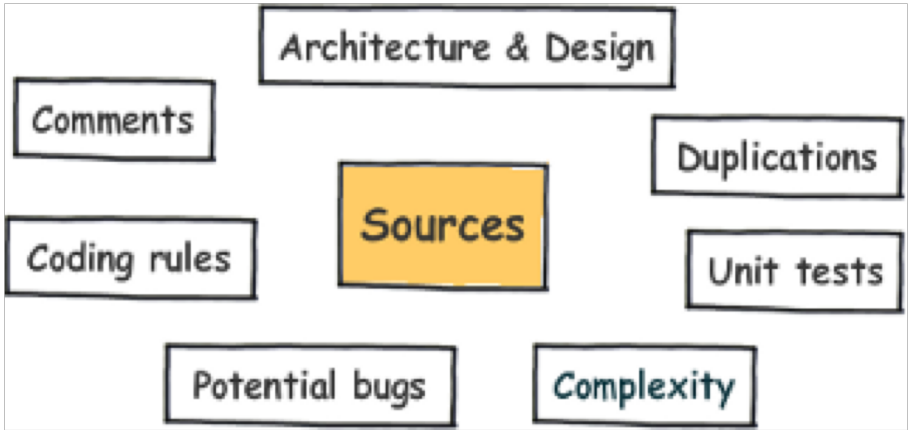
www.opensourcetesting.org

SonarQube

SonarQube è una piattaforma aperta, estendibile con plugin

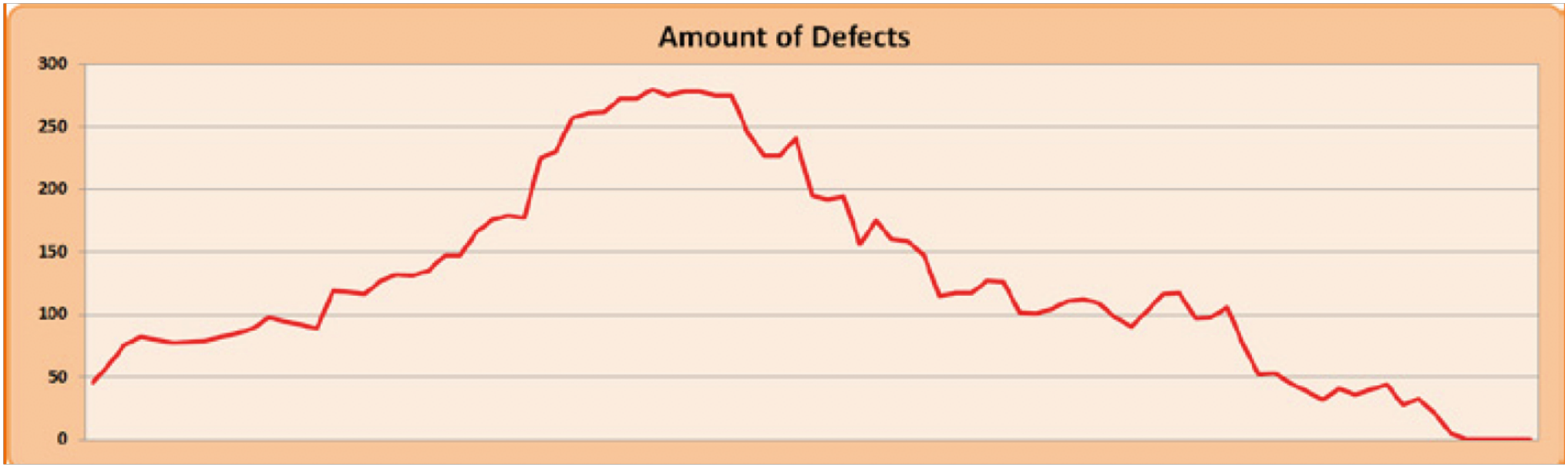
Serve per analizzare grosse codebase, segnalando bugs e smells

Copre più di 20 diversi linguaggi



Il debito tecnico

Debito tecnico è un'espressione che designa le conseguenze di accettare deliberatamente una soluzione di sviluppo non ottimale per risparmiare tempo, che però sarà necessario "spendere" prima o poi, dopo il primo deployment, magari con gli interessi



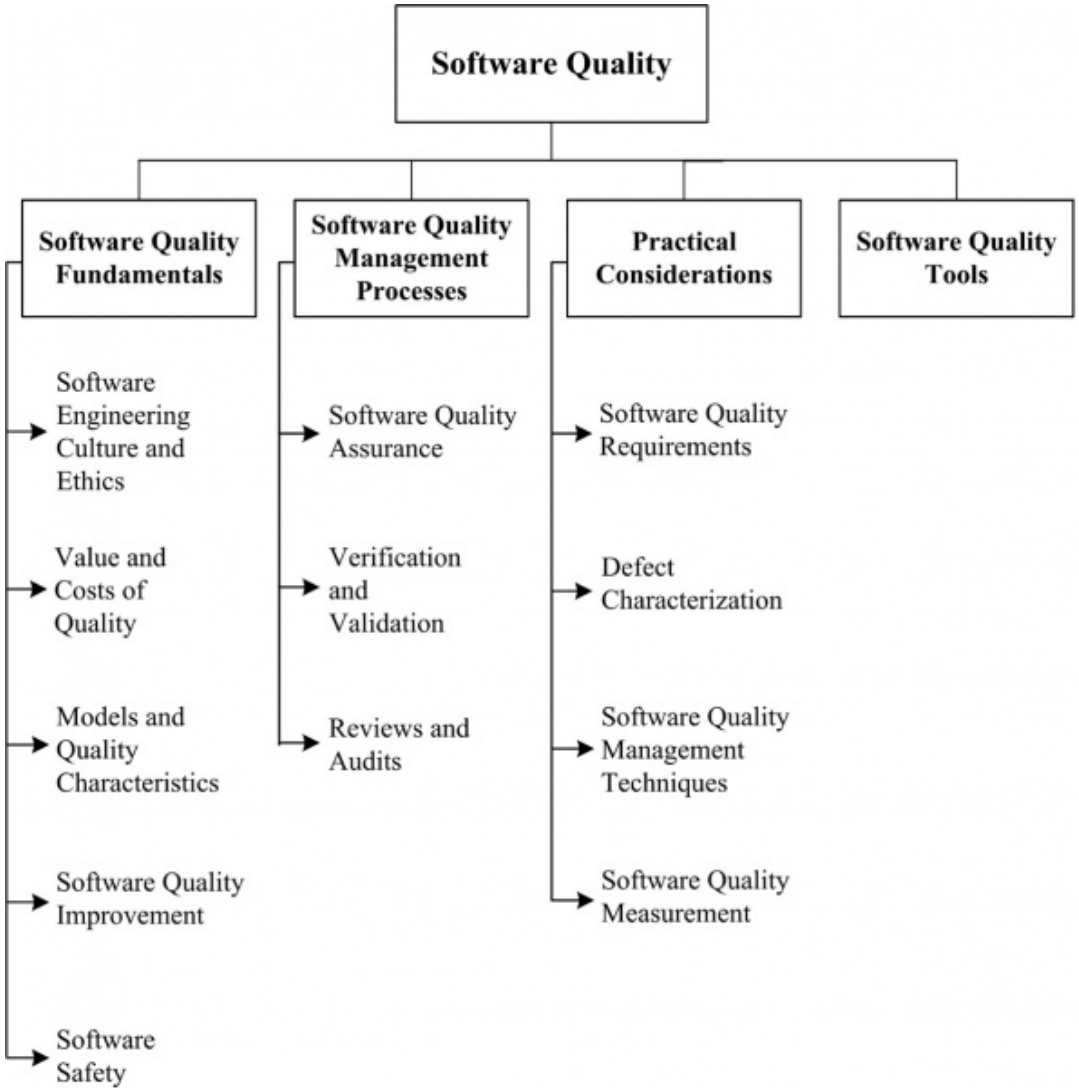
Esempio di andamento del debito tecnico in un anno di sviluppo

https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Conclusioni

- La qualità del software si misura sia rispetto ai requisiti funzionali sia a quelli extrafunzionali, inclusa la qualità del sorgente
- Il testing misura soprattutto la qualità funzionale rispetto ai requisiti
- Le qualità extrafunzionali misurabili sono moltissime: includono per es. le prestazioni, la modificabilità, la manutenibilità, ecc.
- La qualità del codice (es. leggibilità, modificabilità, ecc.) si misura con strumenti appositi, capaci anche di calcolare il debito tecnico

Sw quality in SWEBOK



Domande di autotest

- Qual è la prima cosa da fare quando qualcuno segnala che un prodotto software ha un bug?
- Come si definisce la “correttezza” di un prodotto software?
- Come si possono misurare gli attributi di qualità del software, quali la correttezza, la manutenibilità, l’affidabilità?
- Che differenza c’è tra verifica e validazione?
- Usando l’approccio GQM, come si potrebbe controllare la qualità di un processo di sviluppo? e quella di una specifica UML?
- Quali sono le tecniche di “white box” testing?

Riferimenti

- Capitolo 5 del SWEBOK. “Software testing”, 2014
- Capitolo 11 del SWEBOK. “Software quality”, 2014
- Hutcheson, *Software Testing Fundamentals*, Wiley, 2003
- McConnell, *Code Complete*, MS Press 2004
- McGregor & Sikes, *A Practical guide to Testing OO Software*, Addison Wesley, 2001
- V.Basili, G.Caldera, D.Rombach: The Goal Question Metric Approach, *The Encyclopedia of Software Engineering*, 1994
- N.Davis, W.Humphrey et al.: "Processes for producing secure software”, *IEEE Security and Privacy Magazine*, 2004

Siti utili

- www.asq.org
- www.swquality.com/users/pustaver/index.shtml
- www.ifpug.org/certification/software.htm
- xunitpatterns.com
- www.dmoz.org/Computers/Programming/Software_Testing/Products_and_Tools/
- www.otal.umd.edu/guse/testing.html
- www.aivosto.com strumenti di analisi di errori
- cwe.mitre.org/top25 i 25 errori più comuni
- docs.seleniumhq.org/ testing automatico per applicazioni web

Pubblicazioni di ricerca sul testing e la qualità del software

- IEEE Int. Conf. on Empirical Software Engineering and Measurement
- IEEE Int. Conf. on Software Testing, Verification and Validation
- Int. Conf. on Software Quality
- Software Quality Journal

Domande?


WE FOUND A DOZEN BUGS IN THE SOFT-WARE YOU SOLD US.
WE CAN FIX THESE BUGS FOR $20,000.
WHAT?!
E-mail: SCOTTADAMS@AOL.COM
YOU CAN'T CHARGE US TO FIX YOUR OWN DEFECTIVE PRODUCT!
BUWAHAHA!!
SORRY...I WAS OVERCOME BY AN EVIL EUPHORIA.
I GUESS WE HAVE TO PAY. WE HAVE NO CHOICE!
EXCUSE ME.
3|1|00 © 2000 United Feature Syndicate, Inc.
PUT MORE BUGS IN THE SOFTWARE! I'M MAKING A FORTUNE OUT HERE!
I'M STARTING TO QUESTION OUR SINGLE SOURCE STRATEGY.
PHBLLLT!
www.dilbert.com