Le metriche ed il loro utilizzo nello sviluppo del software

Roberta Sbavaglia
roberta.sbavaglia@technis.info
www.technis.info

Agenda

- La qualità del software e la ISO 9126
  - Panorama, esigenze, definizioni, modello
- Misurare la qualità del software
- Le Metriche di prodotto
  - Metriche tradizionali
  - Metriche OO
  - Mappatura delle Metriche con la ISO 9126
- Misurare la copertura del test

La qualità del software e la ISO 9126

Panorama

- L'evoluzione della tecnologia è continua e veloce
- I Clienti sono più competenti e richiedono prodotti di qualità più elevata in tempi più brevi e a costi più bassi
- L'aggiornamento delle competenze e il loro consolidamento è diventato molto difficile a causa della continua evoluzione tecnologica
- Anche il consolidamento di metodologie e processi da riutilizzare su più progetti è altresì difficile

Esigenze

Fornire al Cliente un prodotto software di qualità in tempi e costi contenuti

Come?

Organizzando un Sistema di Gestione della Qualità

Modelli di Qualità

- I primi modelli di qualità (dei processi di sviluppo del software) sono nati in campo militare, negli anni '80, come "regole" imposte ai fornitori.
- La prima norma in questo settore è la DOD-STD-2167A (US Department of Defense, "Defense System Software Development") del 1988. (poi MIL-STD-498 "Software Development and Documentation" 1994).
- La NATO ha emesso la AQAP-150 "Quality Assurance requirements for software development", Edit. 2 –1997.
- La ISO 9000 il sistema specializzato per la normazione mondiale è la scelta più diffusa in Italia. La norma ISO /IEC 90003 fornisce le linee guida per le organizzazioni per l'acquisizione, la fornitura, lo sviluppo, la gestione operativa e la manutenzione del software per elaboratore

La ISO 9126

- Il principale modello di riferimento per la qualità del software è la normativa ISO/IEC 9126 (*Software engineering – Product quality*), questa si compone di 4 parti:
  - il modello
  - le metriche per la misura della qualità esterna
  - le metriche per la misura della qualità interna
  - le metriche per la misura della qualità in uso

Qualità

- **Percepita (in uso)**
  Esprime l'efficacia ed efficienza, con cui il software serve le esigenze dell'utente, ma anche l'usabilità del prodotto.
- **Interna (intrinseca)**
  Esprime la misura in cui il codice software possiede una serie di attributi, indipendentemente dall'ambiente di utilizzo (analisi statica del codice sorgente).
- **Esterna**
  Esprime le prestazioni del software nel suo ambiente di utilizzo (comportamento del codice in esecuzione).

La ISO 9126
Definizioni

- **La qualità del software:**

  l'insieme delle caratteristiche che incidono sulla capacità del prodotto di soddisfare requisiti espliciti od impliciti.

- **Il prodotto software:**

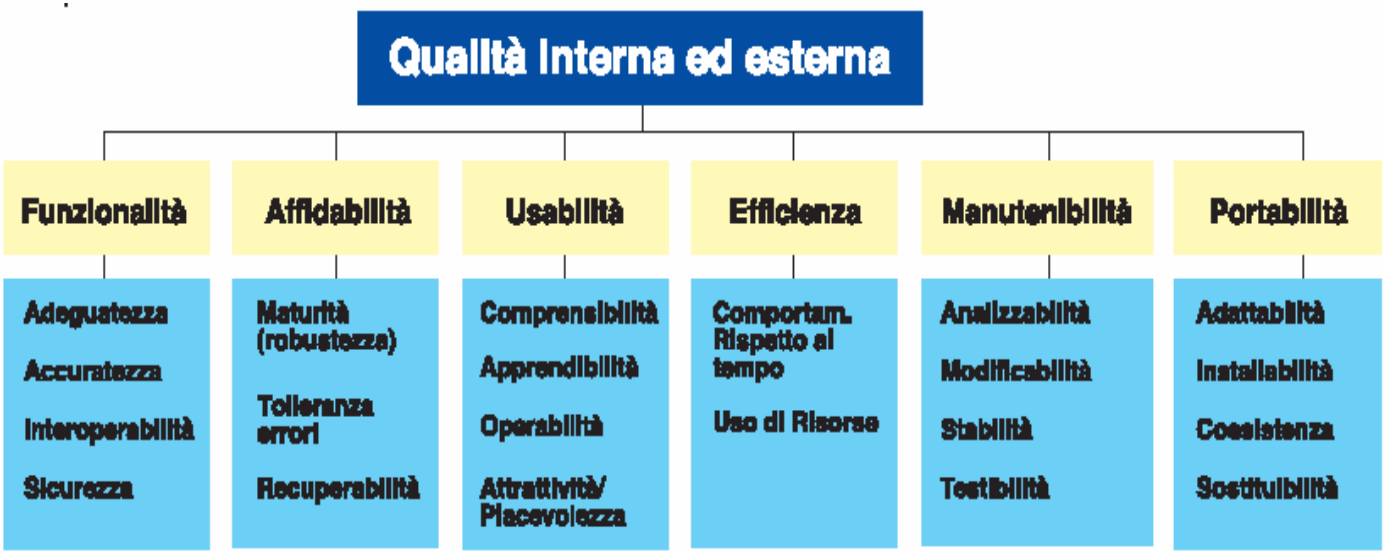
  l'insieme di programmi, procedure, regole, documenti, pertinenti all'utilizzo di un sistema informatico.

La ISO 9126
Definizioni

- **Caratteristica (o attributo):**
  proprietà di un'entità che può essere percepita in modo quantitativo o qualitativo da una persona o uno strumento.
- **Contesto d'uso:**
  l'insieme di utenti, procedure, risorse hardware, software o d'altra natura, l'ambiente fisico e sociale nel quale il prodotto è utilizzato.
- **Indicatore:**
  una misura che fornisce una stima o una valutazione di un attributo sulla base di specifiche regole di misurazione.

Il Modello ISO/IEC 9126

La norma ISO/IEC 9126:2001 definisce sei caratteristiche principali che costituiscono la qualità di un prodotto software



1° Caratteristica: Funzionalità

Capacità di fornire funzioni tali da soddisfare, in determinate condizioni, requisiti funzionali espliciti o impliciti (il software fa ciò per cui è stato sviluppato). Sottocaratteristiche:

- **Adeguatezza**: presenza di funzioni appropriate per compiti specifici
- **Accuratezza**: capacità di fornire risultati od effetti in accordo con i requisiti
- **Interoperabilità**: capacità di interagire con altri sistemi.
- **Sicurezza**: capacità di evitare accessi non autorizzati a programmi e dati

2° Caratteristica: Affidabilità

Capacità di mantenere le prestazioni stabilite nelle condizioni e nei tempi stabiliti. Sottocaratteristiche:

- **Maturità (robustezza):** capacità di evitare fermi della applicazione a seguito di errori nel software
- **Tolleranza errori:** capacità di mantenere livelli di prestazioni predeterminati in caso di fallimenti o di violazione delle regole di interfacciamento
- **Recuperabilità:** capacità di ripristinare livelli di prestazione predeterminati e di recuperare i dati a seguito di errori

3° Caratteristica: Usabilità

Capacità del sw di essere compreso, appreso, usato con soddisfazione dall'utente in determinate condizioni d'uso.
Sottocaratteristiche:

- **Comprensibilità**: impegno richiesto agli utenti per capire il funzionamento del software e la sua applicabilità.
- **Apprendibilità**: impegno richiesto agli utenti per imparare a usare il software.
- **Operabilità**: impegno richiesto agli utenti per usare il software.
- **Attrattività**: capacità del software di essere piacevole per l'utente che ne fa uso.

4° Caratteristica: Efficienza

Rapporto fra prestazioni e quantità di risorse utilizzate, in condizioni normali di funzionamento. Sottocaratteristiche:

- **Comportamento rispetto al tempo**: tempi di risposta e di elaborazione richiesti per eseguire le funzioni richieste in determinate condizioni.
- **Uso di risorse**: quantità e tipo di risorse usate per seguire le funzioni richieste in determinate condizioni.

5° Caratteristica: Manutenibilità

Capacità del software di essere modificato con un impegno contenuto. Sottocaratteristiche:

- **Analizzabilità**: impegno richiesto per diagnosticare carenze o cause di fallimento, o per identificare parti da modificare.
- **Modificabilità**: impegno richiesto per modificare, rimuovere errori o sostituire componenti.
- **Stabilità**: capacità di ridurre il rischio di comportamenti inaspettati a seguito di modifiche.
- **Provabilità**: impegno richiesto per validare le modifiche apportate al software.

6° Caratteristica: Portabilità

Facilità con cui il software può essere trasferito da un ambiente operativo ad un altro. Sottocaratteristiche:

- **Adattabilità**: capacità adattiva a nuovi ambienti operativi applicando le sole azioni previste per questo motivo da parte del software stesso.
- **Installabilità**: impegno richiesto per installare il software in un particolare ambiente.
- **Coesistenza**: capacità di coesistenza con altri software nel medesimo ambiente, condividendo risorse.
- **Sostituibilità**: capacità di essere utilizzato al posto di un altro software.

Livelli di qualità

- La qualità è un livello di possesso di determinati attributi che soddisfa delle esigenze.

- Il livello di qualità minimo dei vari attributi dipende da una serie di fattori quali: la classe di rischio del software che si deve sviluppare, la tipologia di applicazione software, i requisiti del Cliente, il contesto d'uso dell'utilizzo del software.

Livelli di qualità

Esempio:

Caratteristica	Sottocaratteristica	rilevanza
Funzionalità	Adeguatezza	1
	Accuratezza	2
	Interoperabilità	2
	Sicurezza	1
Affidabilità	robustezza	1
	Tolleranza errori	1
	Recuperabilità	1
Usabilità	Comprensibilità	2
	Apprendibilità	2
	Operabilità	1
	Attrattività	3
Efficienza	Comportamento rispetto al tempo	1
	Uso di risorse	1
Manutenibilità	Analizzabilità	1
	Modificabilità	2
	Stabilità	1
	Provabilità	1
Portabilità	Adattabilità	3
	Installabilità	3
	Coesistenza	3
	Sostituibilità	3

Livelli di qualità

Come scegliere le metriche da rilevare?

le metriche e le relative soglie verranno selezionate secondo il criterio di garantire il livello di qualità derivato dal grado di possesso definito per i vari attributi

Misurare la qualità del software

Misurare la Qualità

- Tramite la misura della qualità dei prodotti e dei processi si verifica l'efficienza e l'efficacia del Sistema di gestione della Qualità al fine di soddisfare gli Stakeholder
- Ciò che non si può misurare non si può controllare
- Utilizzo delle Metriche per controllare la qualità del prodotto e del processo

Metriche Software

Vengono utilizzate per assicurare la qualità del prodotto e per valutare l'efficienza del processo produttivo adottato. Sono suddivise in:

- metriche di processo
- metriche di prodotto

Metriche di Processo

Misurano le caratteristiche principali del processo di sviluppo e di manutenzione del software quali:

- Costi
- Tempi
- Produttività

Metriche di Prodotto

Hanno l'obiettivo di misurare la qualità del prodotto software nelle sue caratteristiche fisiche quali:

- Dimensioni
- Funzionabilità
- Manutenibilità
- Usabilità

Le metriche nel ciclo di vita

- Come inserire le metriche nel ciclo di vita del software?
- In quali momenti è più opportuno rilevarle?
- Le metriche da rilevare nelle varie fasi del ciclo di vita sono sempre le stesse?
- Le metriche da rilevare dipendono dal punto di vista?

Le metriche nel ciclo di vita

La rilevanza degli attributi che determinano le caratteristiche qualitative del prodotto software variano in funzione della fase in cui ci troviamo, di conseguenza

anche i criteri di controllo e misurazione

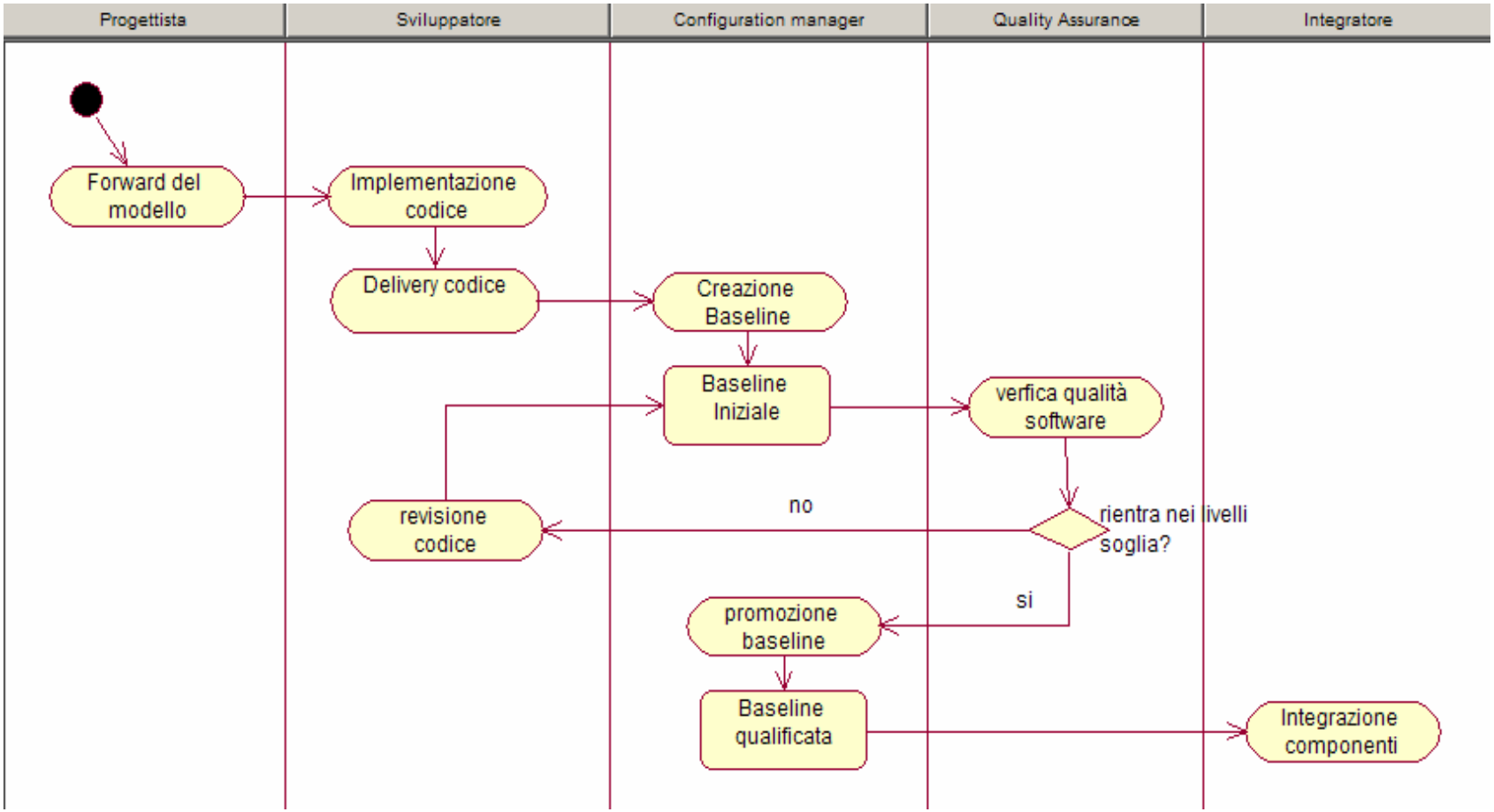
processi:	Usabilità	Portabilità	Funzionalità	Manutenzione	Affidabilità	Efficienza
Sviluppo	X	X	X	X	X	X
Evoluzione			X			
Miglioramento	X			X		
Adeguamento		X				X
Correzione			X		X	X

Le metriche nel ciclo di vita

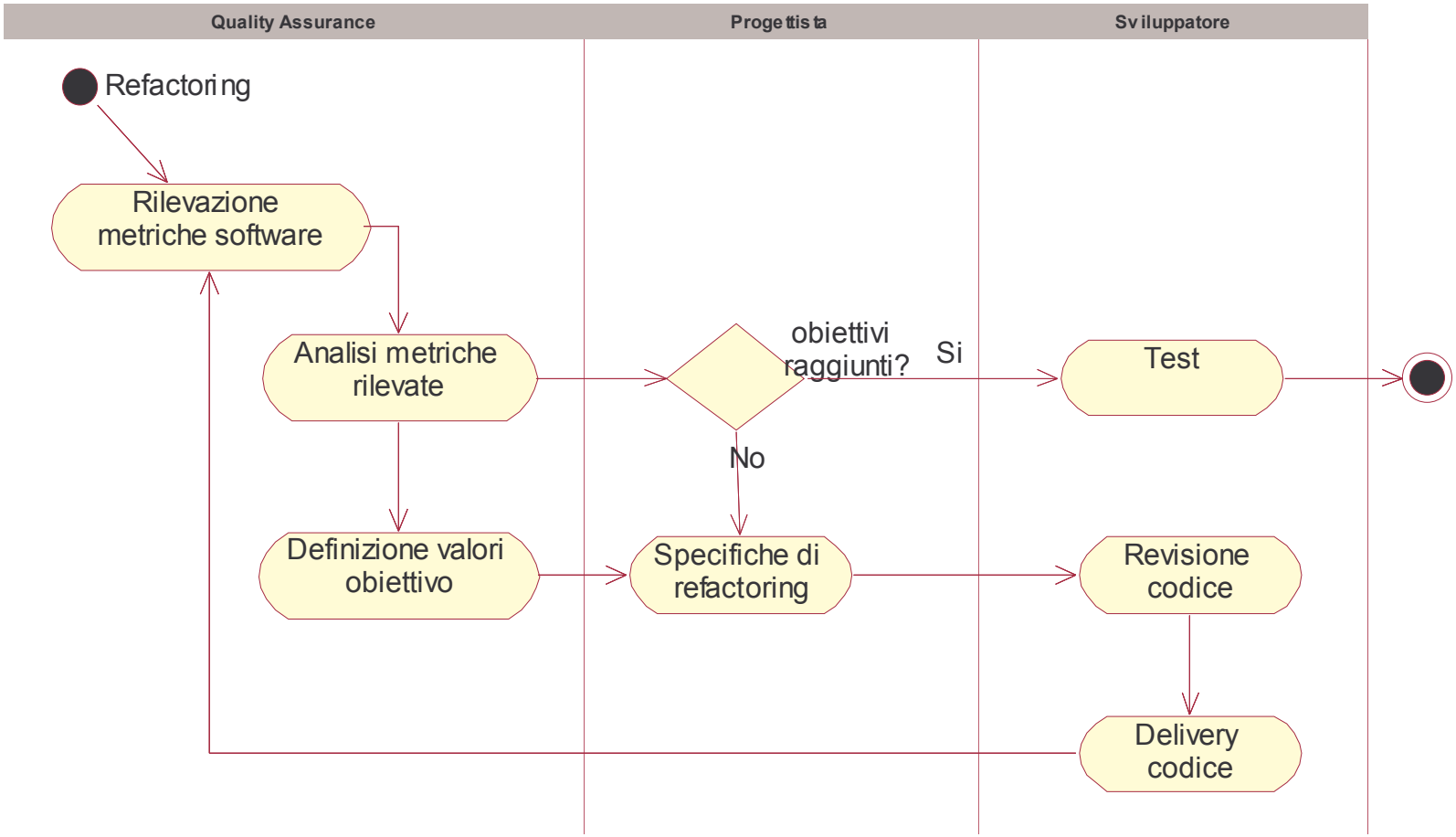
Secondo i punti di vista alcuni fattori di qualità diventano più o meno sentiti (indicati con la lettera Q), così come varia l'esigenza di quantificare il fenomeno con misurazioni (indicata con m).

Punti di vista:	Usabilità	Portabilità	Funzionalità	Manutenzione	Affidabilità	Efficienza
Utente	Q	Q	Q		Q	Q
Committente			Q m	Q m	Q m	Q m
Terza parte				Q m	Q m	Q m
Fornitore	Q	Q	Q m	Q m	Q m	Q m

Le metriche nel ciclo di vita



Le metriche nel ciclo di vita



Caratteristica di una Metrica

Una buona metrica deve essere:

- **Intuitiva**
- **Oggettiva**
- **Indipendente dal linguaggio**

Le Metriche di prodotto

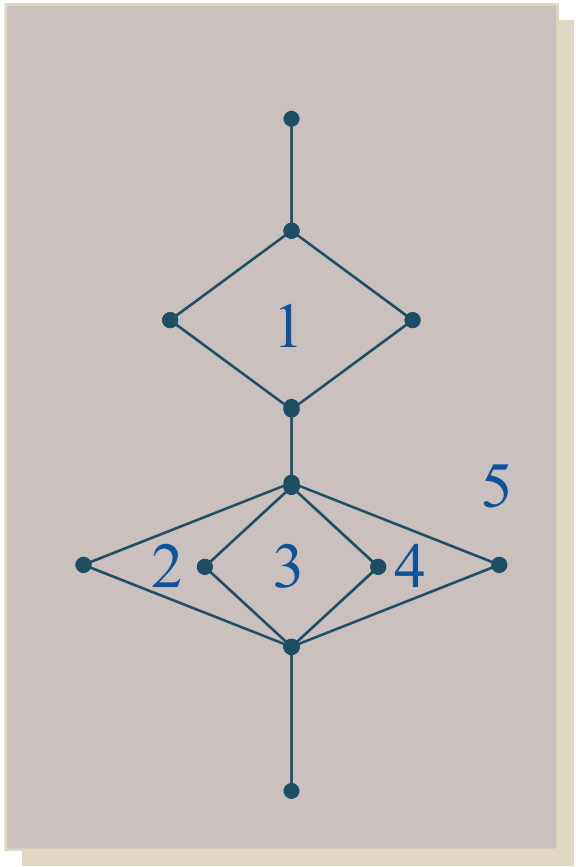
Complessità ciclomatica

La complessità ciclomatica di McCabe **v(g)** è la misura della complessità logica di un modulo.

La v(g) è il numero di path linearmente indipendenti e conseguentemente il numero path che dovrebbero essere testati.

Questa misura da indicazioni sulla complessità logica dei moduli e sull'effort richiesto per testare adeguatamente i moduli

Complessità ciclomatica



3 metodi per calcolarla

1°: v(g) = r-n+2

Numero dei rami – il numero dei nodi + 2

2°: v(g) = p+1

3°: v(g) = n dove n è il numero di aree in cui è diviso il foglio dal grafico

Soglia: v(g) <= 10

Loc (Line Of Code)

Questa metrica misura il numero di linee di codice di un modulo esclusi i commenti e le linee vuote.

È una metrica che dà unicamente informazioni dimensionali e non viene mai utilizzata da sola.

Soglia: Loc <= 30

Densità di commenti

La metrica **% Comm** misura la percentuale delle righe di commento sul totale delle righe di codice presenti in un modulo.

Questa metrica è uno degli indicatori che contribuisce a valutare la manutenibilità di un modulo.

Soglia: 20<= %comm <=35

Valori più alti spesso indicano la presenza di codice commentato

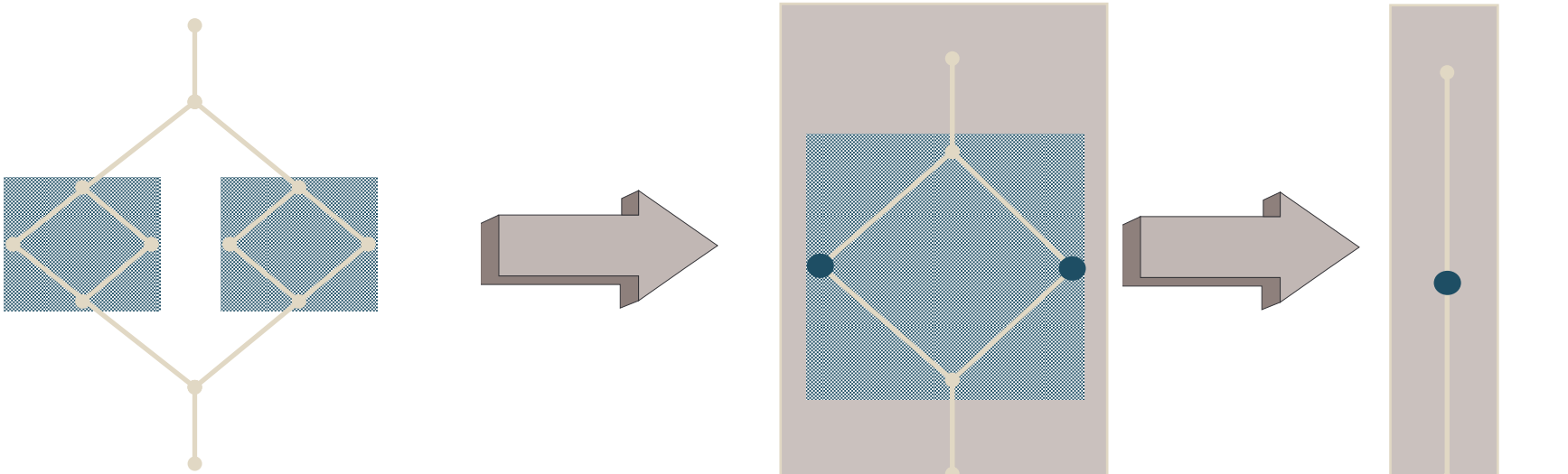
Complessità essenziale

La complessità essenziale **ev(g)** è la misura di quanto un modulo contenga costrutti destrutturati. Questa metrica quantifica il grado di destrutturazione di un modulo ed aiuta a valutare l'effort necessario per la manutenzione

Soglia: ev(g) <= 4

Complessità essenziale

La complessità essenziale ev(g), è la misura della complessità ciclomatica di un flow ridotto dal quale sono stati eliminati i path strutturati.



Cyclomatic Complexity = 4

Essential Complexity = 1

Complessità d'integrazione

La complessità d'integrazione **iv(g)** di un modulo è la misura dei path che contengono chiamate ad altri moduli.

La iv(g) è calcolata come la complessità ciclomatica di un grafo ridotto, ottenuto eliminando i path e nodi che non contengono chiamate ad altri moduli.

Soglia: iv(g) <= 7

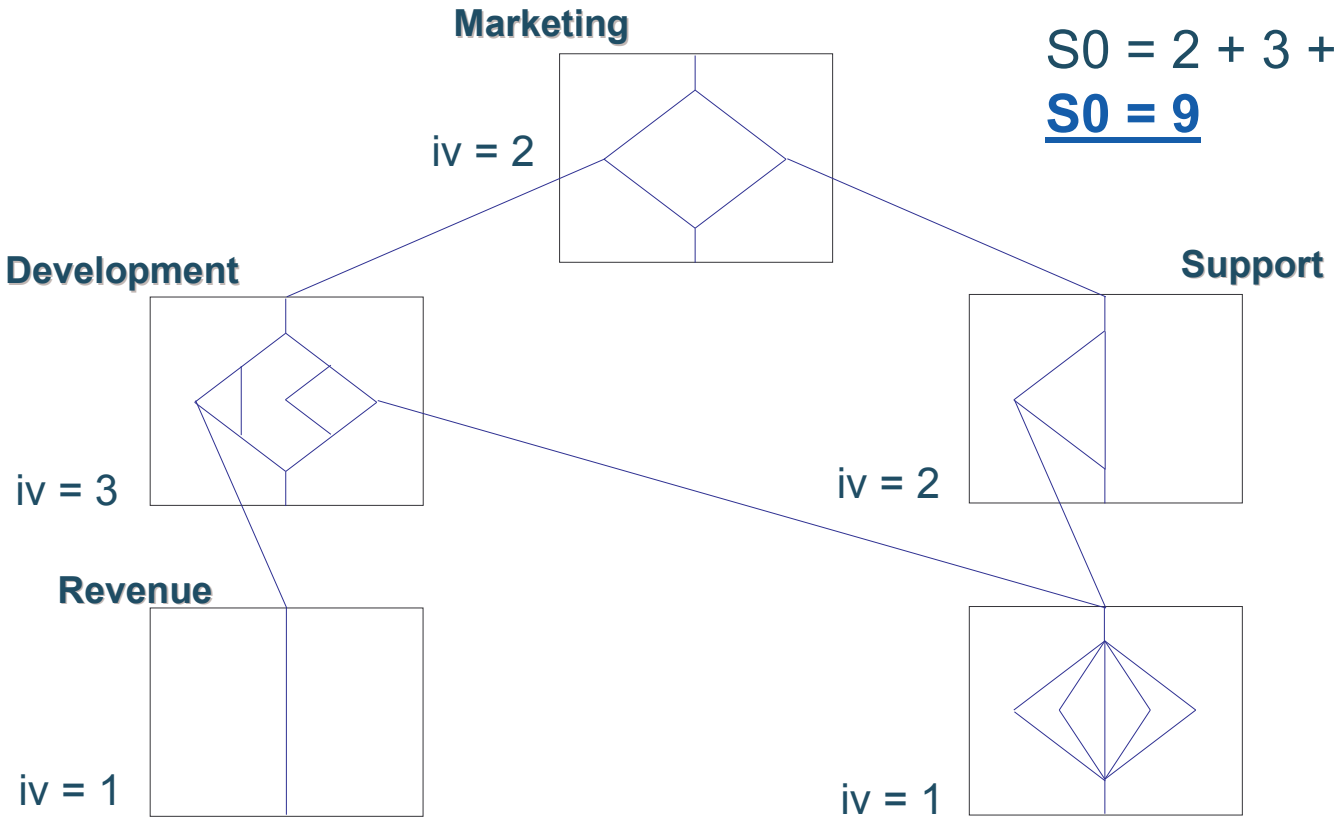
Design Complexity

La metrica Design Complexity **S_0** è la misura a livello di progetto della complessità d'integrazione

$S_0 = \sum iv(g)$

Più grande è questo valore più grande sarà l'effort necessario a garantire un adeguato test d'integrazione dell'intera applicazione

Design Complexity



S0 = 2 + 3 + 2 + 1 + 1

S0 = 9

Metriche OO

Paradigma OO

- **Incapsulamento**
- **Ereditarietà**
- **Coesione**
- **Polimorfismo**

Incapsulamento

PUB_ACCESS

Misura , il numero dei metodi che accedono a dati definiti in un'altra Classe, violando il principio dell'incapsulamento.

$$Pub\_Access = \sum_{i=1}^{N} Mi$$

dove

M = Metodi che accedono a dati Pubblici o Protetti

N = Numero di Metodi nella Classe

Soglia: PubAccess = 0

Questo valore dovrebbe essere il più basso possibile in quanto è una esplicita violazione del paradigma dell'incapsulamento. Tale violazione implica una dipendenza tra l'oggetto e dati/attributi al di fuori della gerarchia di ereditarietà.

Incapsulamento

PUB_DATA

Misura il numero di dati definiti pubblici e protetti in una Classe, in percentuale sul totale dei dati stessi.

$$PUB\_DATA = \left(\left(\sum_{i=1}^{N} Ai \right) / N \right) \bullet 100$$

N = Numero di Attributi complessivi

A = Attributo pubblico/privato definito nella Classe

Soglia: PUB_DATA <= 0

Questo valore dovrebbe essere il più basso possibile in quanto definire dati pubblici o protetti è una "dichiarazione di intenti" di violare il paradigma dell'incapsulamento.

Incapsulamento

CBO Coupling Between Objects

Misura il numero di classi correlate ad una classe al di fuori alla gerarchia di ereditarietà.

$$CBO = \sum_{i=1}^{N} Ci$$

N = numero delle Classi non appartenenti alla gerarchia di ereditarietà

C = Classe correlata

Soglia: **CBO <=** 2

Funzionalità del sistema fortemente connesse tra loro (in termini di dati e di classi) hanno un elevato grado di *coupling* e sono più difficili da mantenere. La modifica di una classe impatterà su tutte le classi che hanno un "accoppiamento" con questo. Valori alti indicano una difficoltà di riuso del componente.

Coesione

LOCM/LCOM Lack Of Cohesion of Methods

Misura il numero di metodi che condividono uno o più attributi.

*LOCM = 1 - sum(mA)/(m*a)* dove

m = Numero dei Metodi in una Classe

a = Numero di Variabili (Attributi) di una Classe

mA = Numero di Metodi che accedono ad una Variabile (Attributo)

sum(mA) = Somma di mA sugli attributi di una Classe

Soglia LOCM >= 75%

LOCM viene calcolato come percentuale dei metodi che non accedono a specifici attributi definiti in altri metodi della stessa classe

Coesione

LOCM/LCOM Lack Of Cohesion of Methods

Classi con bassa coesione possono esser probabilmente suddivise in più sottoclassi. Più il valore è alto e maggiore è la coesione tra dati e metodi. Il valore di 100 % è comune in classi che sono usate per contenere i dati ma queste non devono avere un valore di Sum v(G) superiore a 7 * WMC.

Polimorfismo

WMC Weighted Methods per Class

Misura i metodi pesati implementati all'interno di una classe.

$$WMC = \sum_{i=1}^{N} Ci$$

N = numero dei metodi implementati all'interno della Classe

C = complessità ciclomatica

Se tutte le complessità sono considerate essere unitarie, allora WMC = n, dove n è il numero dei metodi.

Soglia WMC <= 14

Questa metrica indica lo sforzo necessario per sviluppare e/o manutenere la classe. Troppi metodi rendono la classe di difficile comprensione, incrementando il rischio d'errori a fronte di una modifica e ne limitano la possibilità di riuso.

Polimorfismo

RFC Response for a Class

Misura il numero dei metodi invocato nella risposta ad un messaggio.

$RFC = M + R$

M = numero dei metodi della Classe

R = Numero dei metodi remoti chiamati, attraverso l'intero albero delle chiamate

Soglia: RFC <= *(WMC * Depth) + 1*

Valori grandi indicano una grande complessità e una peggiore comprensibilità del codice. Valori molto bassi indicano un uso massiccio del polimorfismo.

Ereditarietà

FAN IN (OO FAN IN)

Indica la presenza di "ereditarietà multiple" in una Classe ed il numero di classi da cui sono ereditati dati e attributi.

$$OOFanIn = \sum_{i=1}^{N} Ci$$

C = Classe dalla quale si ereditano dati ed attributi

N = Numero di Classi Root

Soglia FAN IN <= 1

FAN IN > 1 indica ereditarietà multipla, fortemente sconsigliata in quanto incide sulla manutenibilità del disegno object oriented incrementando le "interconnessioni" tra classi incrementando quindi l'impatto potenziale a fronte di una modifica e incrementando la complessità del test in quanto l'oggetto dovrà essere testato "n" volte quante sono le gerarchie che lo ereditano.

Ereditarietà

Depth

Misura il livello di profondità della classe interna alla gerarchia di ereditarietà ed è misurata dal numero di classi tra il nodo e la radice della gerarchia

$$Depth = \sum_{i=1}^{N} Li$$

N = Numero massimo di livelli dell'albero gerarchico delle Classi

L = Livelli tra la Classe oggetto di misurazione e il livello precedente

Soglia DEPTH <= 7

Più profonda è la gerarchia, maggiore sarà la complessità del disegno OO poichè verrà coinvolto un maggior numero di metodi pur comportando un maggiore riuso del codice.

Ereditarietà

CDC Class Dependent on a Children

Il flag CDC indica la dipendenza di una Classe da una classe "child" (dipendenza di dati ed attributi non ereditati).

$$CDC = \sum_{i=1}^{N} Ci$$

C = Classe che dipende da un suo Child

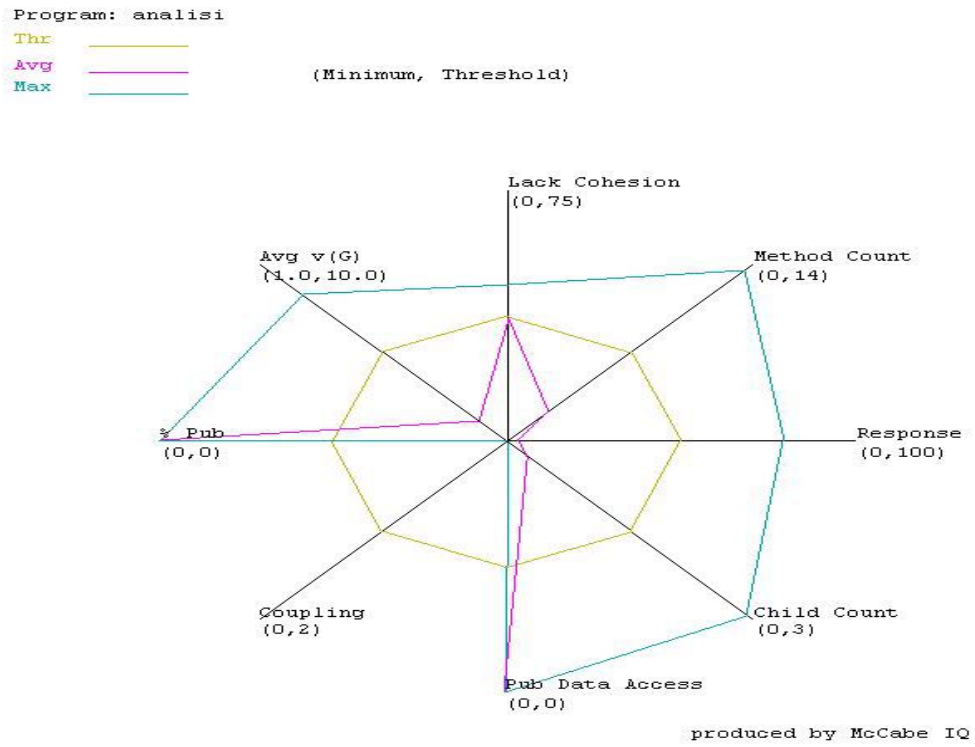
N = Numero delle Classi Child

Soglia: CDC= *FALSE*

Una Classe che dipenda da un suo Child è simile al concetto di ricorsività nei sistemi tradizionali e dovrebbe essere evitata, poiché deteriora la comprensibilità del codice e rende difficile la creazione della classe di test che dovrà essere "ricorsiva" a sua volta.

Diagramma di Kiviat

uno strumento semplice e molto efficace è la rappresentazione di queste metriche tramite un diagramma di Kiviat



Le Metriche e la ISO 9126

Manutenibilità

- Class Depending Child (CDC)
- Class Depth (DEPTH)
- Essential Complexity (Ev(G))
- Multiple Inheritance (FAN IN)
- Access to Protect or Public Data (PUB_ACCESS)
- Protect or Public Data Definition (PUB_DATA)
- Response for Class (RFC)
- Cyclomatic Complexity (v(G))
- Sum of Cyclomatic Complexity of a Class (SUM v(G))
- Module design complexity (Iv(g))
- Design complexity (S_0)

Le Metriche e la ISO 9126

Portabilità

- Class Depending Child (CDC)
- Class Depth (DEPTH)
- Coupling Between Object (CBO)
- Multiple Inheritance (FAN IN)
- Lack of Cohesion of Methods (LOCM/LCOM)
- Response for Class (RFC)
- Weighted Methods for Class (WMC)

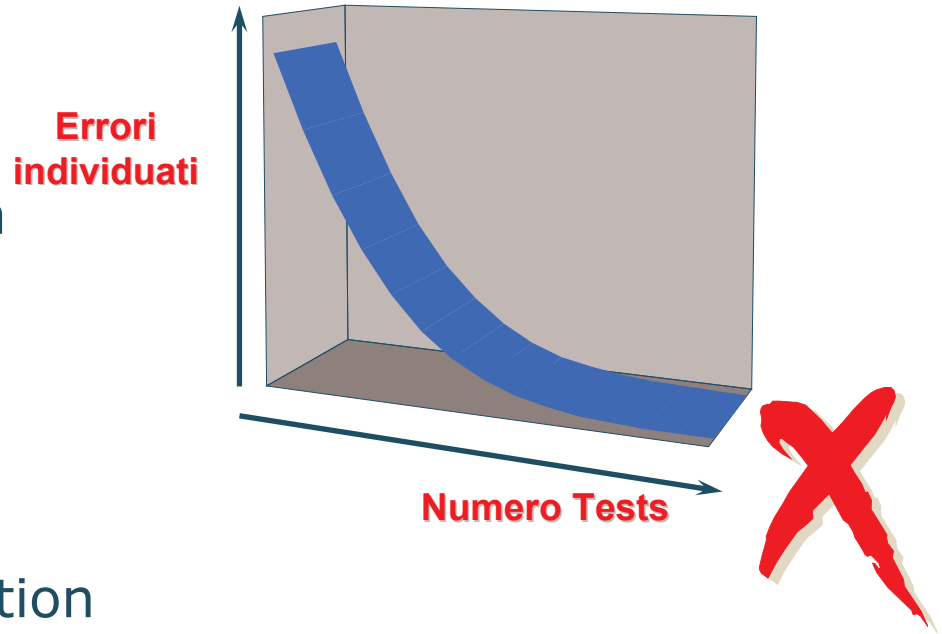
Misurare la copertura del test

Scenario

La Copertura Topologica del Test, è uno strumento estremamente importante per garantire che il prodotto software sia il più affidabile possibile, riducendo sia i costi di manutenzione, sia i costi dovuti ai fermi delle applicazioni rendendo oggettivi alcuni aspetti delle fasi di Collaudo difficilmente evidenziabili (p.e. codice inerte)

La realtà del Test

- Le attività di Test finiscono sempre troppo presto
- Il 100% del Test non è mai possibile
- Si necessita di identificare un minimo set di test possibili
- Stabilire un tempo limite che NON sia basato su:
  - La mancanza di tempo
  - I Test casuali
- Gli strumenti di Test Automation sono fondamentali ma non danno una risposta "completa"



Misurare la copertura topologica del test

- Effettuare l'assessment del test eseguito
  - Determina quanta "logica di codice" è stata realmente percorsa dai Test
  - Aiuta nella determinazione del Rischio nel rilasciare il Software
- Indicare le migliorie necessarie alle fasi di test
  - Aggiunge nuovi test per coprire la maggiore quantità di logica di codice possibile
  - Trovare i percorsi non testati
- Definire le priorità del Test
  - Assicurarsi che il codice critico o modificato sia testato per primo
- Ridurre la duplicazione del test
  - Identificare i Test simili che non danno valore aggiunto e rimuoverli

Criteri di copertura

- Code coverage
- Branch coverage
- Path coverage
- Boolean coverage

Code Coverage

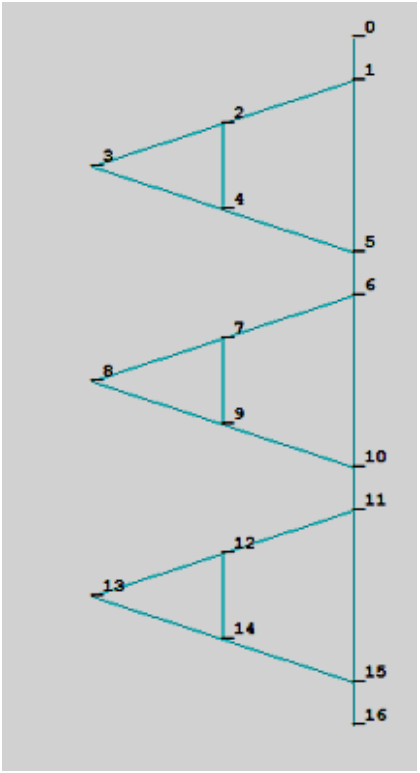
- Misura la capacità di coprire mediante esecuzione di test tutte le linee di codice di un modulo.

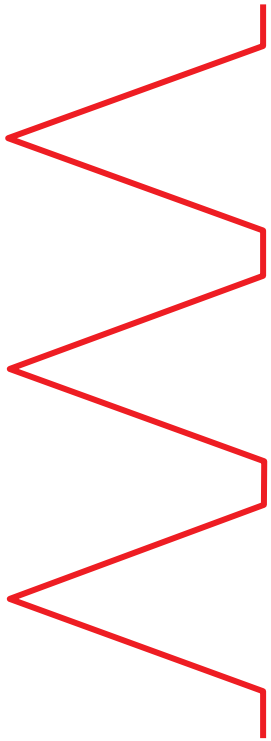
- Una copertura topologica del test del 100% di tipo Code Coverage garantisce di aver eseguito almeno una volta tutte le istruzioni, ma non tutti i rami.

Code Coverage

```
int compare(int a, int b, int c)
{
    int ret = 0;

    if (a > 10)
    {
        if (a > c)
            ret += a/b;
    }
    if (b > 10)
    {
        if (b > c)
            ret += b/c;
    }
    if (c > 10)
    {
        if (c > a)
            ret += c/a;
    }
    return(ret);
}
```





1 Execution ⟶ 100% Line Coverage

Branch Coverage

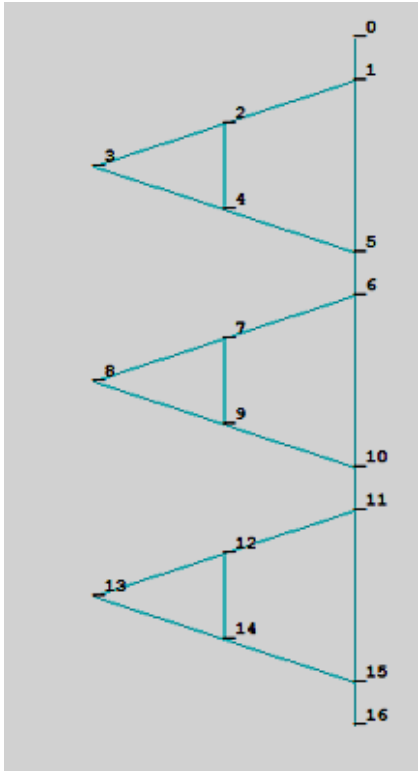
- Misura la capacità di coprire mediante esecuzione di test tutti i rami di un modulo (ogni arco del grafo di flusso deve appartenere almeno ad un cammino di esecuzione).

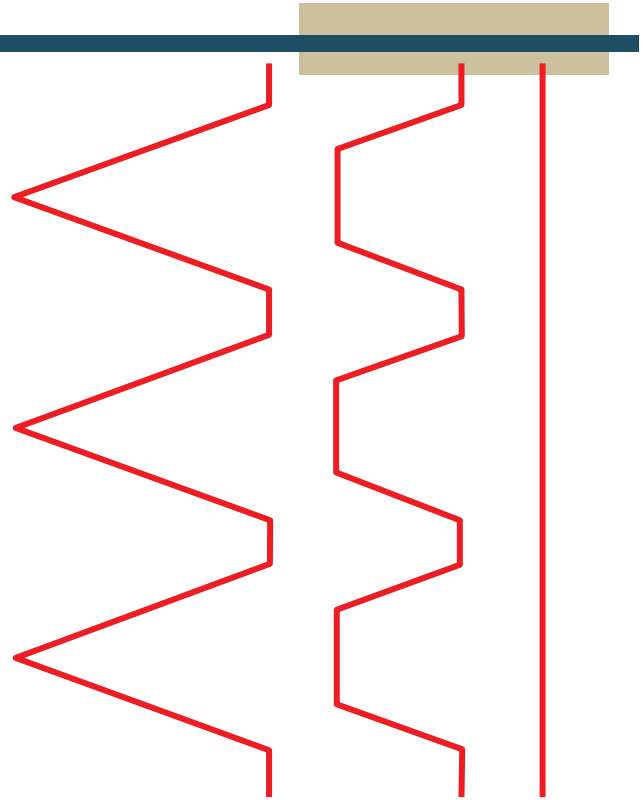
- Una copertura topologica del test del 100% di tipo Branch Coverage garantisce di aver eseguito almeno una volta tutti i rami, ma non di aver esercitato tutte le condizioni.

Branch Coverage

```
int compare(int a, int b, int c)
{
    int ret = 0;

    if (a > 10)
    {
        if (a > c)
            ret += a/b;
    }
    if (b > 10)
    {
        if (b > c)
            ret += b/c;
    }
    if (c > 10)
    {
        if (c > a)
            ret += c/a;
    }
    return(ret);
}
```





1 Execution ⟶ 100% Line Coverage

3 Executions ⟶ 100% Branch Coverage

Path Coverage

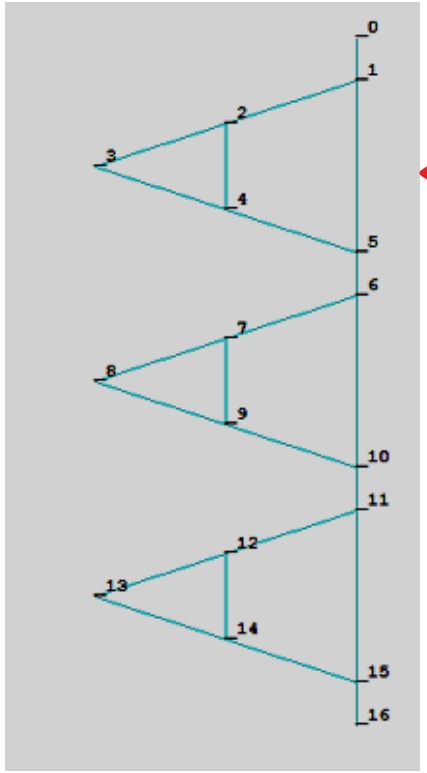
- Misura la capacità di coprire mediante esecuzione di test tutti i path di un modulo (ogni path linearmente indipendente del grafo di flusso deve appartenere almeno ad un cammino di esecuzione).

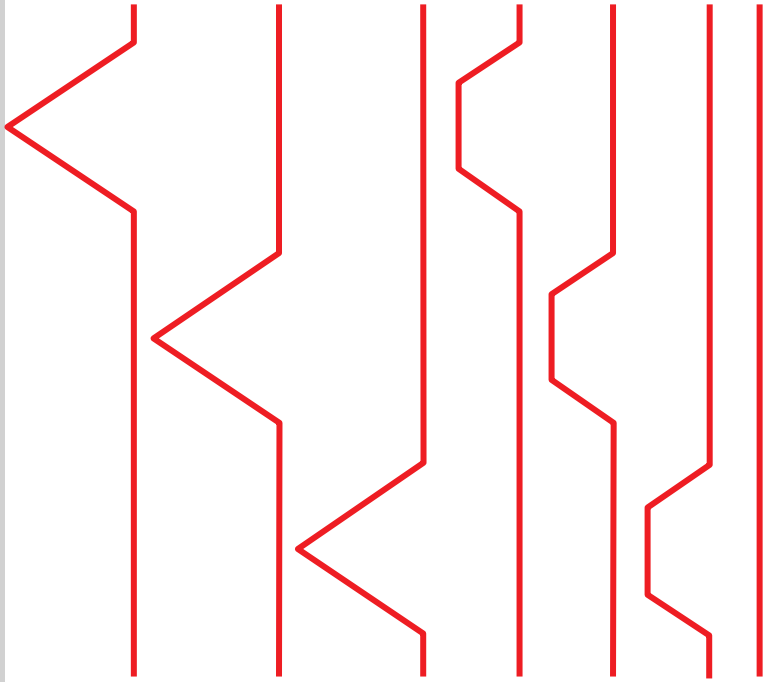
- Una copertura topologica del test del 100% di tipo Path Coverage garantisce di aver eseguito almeno una volta tutti i path linearmente indipendenti, ma non di aver esercitato tutte le combinazioni delle condizioni.

Path Coverage

```
int compare(int a, int b, int c)
{
    int ret = 0;

    if (a > 10)
    {
        if (a > c)
            ret += a/b;
    }
    if (b > 10)
    {
        if (b > c)
            ret += b/c;
    }
    if (c > 10)
    {
        if (c > a)
            ret += c/a;
    }
    return(ret);
}
```





1 Execution ⟶ 100% Line Coverage
3 Executions ⟶ 100% Branch Coverage
7 Executions ⟶ 100% Path Coverage

Boolean Coverage

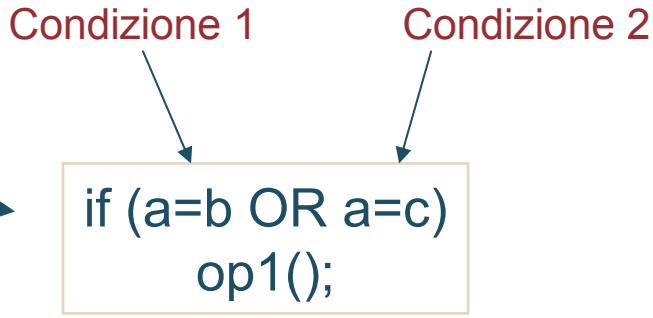
- Misura la capacità di coprire mediante esecuzione di test tutte le condizioni di un'espressione booleana di un modulo.

- Una copertura topologica del test del 100% di tipo Boolean Coverage garantisce di aver eseguito almeno una volta tutte le condizioni presenti in un'espressione booleana.

Boolean Coverage

```
int compare(int a, int b, int c)
{
    int ret = 0;

    if (a = b or a = c)
    {
        ret += a/b;
    }
    if (b > 10)
    {
        if (b > c)
            ret += b/c;
    }
    if (c > 10)
    {
        if (c > a)
            ret += c/a;
    }
    return(ret);
}
```



Test #	C0	C1	Result
1	T	T	T
2	F	T	T
3	F	F	F

Coverage Comparison

Code Coverage

Si ottengono valori alti di copertura con poco effort ma è molto poco significativa

Branch Coverage

È correlata più linearmente con il numero di test eseguiti ed incrementa l'efficacia del Test

Expanded Branch Coverage

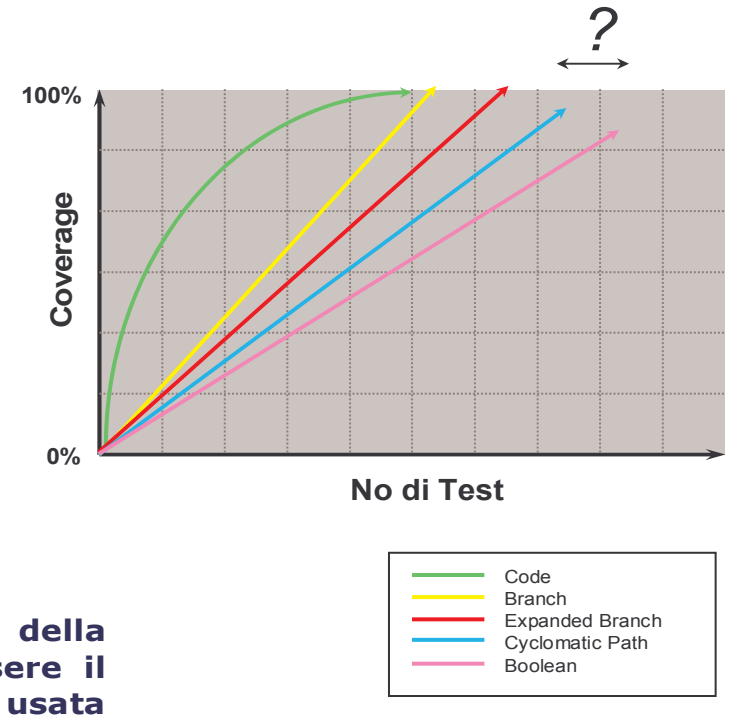
Estende il Branch Coverage includendo la "compound logic" all'interno del codice

Path Coverage

È molto più rigorosa, richiede un grande effort e non può essere il 100%

Boolean Coverage

Aggiunge rigore al testing della "compound logic" e non può essere il 100%. Dovrebbe essere usata combinata con un altra tipologia di copertura



Misure di copertura

- Il concetto di copertura può quindi essere utilizzato come metrica per valutare l'effettività dei test eseguiti
- Un criterio quantitativo di copertura viene spesso utilizzato come soglia che va raggiunta durante l'attività di test, cioè definisce il criterio di uscita dalla fase di test
- Può venire utilizzato anche per individuare eventuali porzioni di "codice morto", quel codice cioè, che non verrà mai eseguito

Riferimenti

- ISO/IEC 9126:2001 "Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use" 1991
- ISO/IEC 14598-1:2000 "Ingegneria del Software – Tecnologia dell'informazione- Valutazione del prodotto software – Visione generale"
- Roger S. Pressman, "Principi di Ingegneria del Software", McGraw Hill, Luglio 2000
- S.R. Chidamber, C.F. Kemerer, A metrics suite for Object Oriented Design, MIT Sloan School of Management December 1993
- Arthur H. Watson, Thomas J. McCabe, "Structured Testing: A testing methodology using the cyclomatic complexity metric", NIST
- http://www.mondomatica.it/professioni.htm