Progettazione SW sicuro

Indice

PROPRIETÀ DEL PROCESSO E PRODOTTO parte 1

PROCESSO: Come realizziamo il prodotto **(PTT)**

- **PRODUTTIVITA'** efficienza, velocità di consegna
- **TEMPESTIVITA'** valutare, rispettare tempi di consegna
- **TRASPARENZA** capire il suo stato attuale e i suoi step

PRODOTTO: Cosa realizziamo (SW) **(IMH)**

- **INTANGIBILE**: difficile da descrivere/valutare
- **MALLEABILE:** Può essere trasformato
- **HUMAN INTENSITIVE:** Non comporta nessun processo manufatturiero tradizionale

QUALITÀ DEL PRODOTTO parte 2

- **INTERNE:** (white box) Non visibile all'utente **(RVF)**
  RIUSABILITA': riusato tutto o parzialmente per costruire nuovi sistemi
  VERIFICABILITA': dimostrare la correttezza con testing , verifiche run time
  FACILITA MANUTENZIONE: nella ricerca degli errori (modifiche correttive 20%), aggiungere nuove funzionalità (perfettive 50%) o ai cambiamenti (adattive 20%)

- **ESTERNE** (blackbox) **(CERA PIU)**
  AFFIDABILITA': Se si comporta "come previsto ", la probabilità di **ASSENZA di fallimenti** in un dato intervallo di tempo; se le specifiche sono "corrette" tutto il SW è affidabile; ma non viceversa!
  CORRETTEZZA: rispetta specifiche funzionali iniziali del progetto
  EFFICIENZA: Se sfrutta al meglio le risorse a disposizione: memoria, tempo, metodi comunicazione
  USABILITA': Facilità d'USO
  ROBUSTEZZA: Resistente agli imprevisti es: input scorretti, rottura dischi
  PORTABILITA': Funziona su + piattaforme es: programmi in java
  INTEROPERABILITA': cooperare con altri sistemi es: word processor con ambienti grafici

***SICUREZZA*: proprietà relativa al contesto (non assoluta) → stabilire policy**

- **SAFETY** Il sistema NON CAUSA danni all'esterno, exploit non accadano DAL sistema
- **SECURITY** Il sistema NON SUBISCE danni all'esterno, exploit non accadano SUL sistema

OBIETTIVO DEL PRODOTTO parte 3 (PS AMA PIT)

- **PREVENZIONE**: anticipare attacchi a lv. Progettazione/implementazione/ uso
- **TRACCIABILITÀ/auditing**: è UN PROCESSO/**VALUTAZIONE INDIPENDENTE VOLTA A OTTENRE PROVE, VALUTARLE CON OBIETTIVITA'** PER STABILIRE in maniera inequivocabile relazione causa/effetto e relativo livello di rischio
- **MONITORAGGIO**/ controllo delle info **con policy**/approcci semplici (check signature note per identificare l'attacco in corso) o complessi (mediante asserzioni). E' un **auditing in tempo reale** che GENERA ALLARMI (possono verificarsi anche falsi allarmi)
- **PRIVATEZZA (**E' il diritto di un individuo di stabilire **se, come, quando e a chi** l'informazione che lo riguarda può essere rilasciata) **/CONFIDENZIALITÀ** (info. Accessibili solo ad utente autorizzati)
- **SICUREZZA A + LV**: classificate in: pubbliche, confidenziali, top-secret…
- **ANONIMATO:** Mantenere segrete l'origine dei dati
- **AUTENTICAZIONE:** conoscere l'identità di chi accede ad un servizio (SSL garantisce solo protezione)
- **INTEGRITÀ**: rimanere lo stesso , non modificato DA QUANDO creato

PRINCIPI delle ARCHITETTURE SICURE parte 4 (AMA DTPM FACS)

- **ADVERSARY principle**: progetta il SW come se il nemico più astuto lo attaccherà anticipa attacchi cercando di prevedere sol. **GASSP** Generally Accepted System Security Principles
- Rispettare la **CHAIN OF TRUST:** Non invocare programmi *NON fidati* da programmi fidati
- **Minimo PRIVILEGIO**: se devo leggere un file non lo apro anche con la 'W'
- **MEDIAZIONE completa**: test di verifica contro la policy per verificare la conformità del sistema
- **Minimo STATO**: un programma deve tenere in memoria lo stato minimale → per evitare SYN FLOOD/ **CGI** (Common Gateway Interface risiedente sul server per inviare al client contenuto dinamico) scripts
- **FAULT TOLERANCE: Identificare le funzionalità critiche** tenendo sotto controllo: **Resistance**(Impedire/resistere un attacco), **Recognition**(riconoscere attacco e misura del danno), **Recovery** (ripristinare tutti i servizi dopo l'attacco).
- **GRACEFUL degradation**: Il sistema continua a lavorare in maniera ristretta/ridotta Esempio senza: pacchetti tcp; con: gestione memoria nel S.O: VMS
- **FAULT-SAFELY:** in caso di errore → terminazione in modo sicuro.
- **ACCETTABILITA' PSICOLOGICA:** utilizzare modelli/paradigmi tratti dal mondo reale per non irritare l'utente.
- **AUDITABILIY:** ricostruire la sequenza in base ai log del sistema per ricostruire eventi
- **MODULARIZZAZIONE:** Suddivisione la progettazione in moduli ciascuno con una propria interfaccia che rispettino il minimo privilegio

SICUREZZA IN JAVA parte 1

Confronto con C/C++

ERRORI tipici in c/c++ → gestione della memoria a carico del programmatore (puntatori)

- **DEFEREZIAZIONE**: accesso ad una cella puntata da un puntatore nullo (int *p=0) →segmentation fault
  Accesso ad una posizione di memoria per la quale non gli è permesso accedere.
- **TYPECAST not safe**: conversione di tipo non controllata con possibile perdità info (int a = b) con b double
- **POINTER ARITHMETIC:** puntare a zone di memoria con tipi diversi
- **MEMORY SAFE:** Modificare diritti/causare bufferOverflow(errore spazio) utilizzando puntatori con array

Dangling pointer: (errore temporale) punta ad un zona di memoria che è stata liberata (non più valida) per essere riutilizzata

SOLUZIONI: malloc, uso del garbage collector(**GC**): modalità automatica in cui il S.O. libera porzioni di memoria non + utilizzate da applicazioni evitando **MEMORY LEAKS.**
SW: PURIFY analisi dinamica per scoprire accessi alla memoria, SafeC

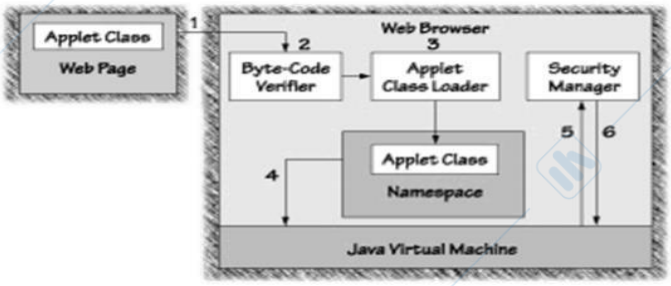
JAVA SVANTAGGI: Prestazioni inferiori in termini di tempo per i controlli GB, impiego maggior memoria per tenere info. Sui tipi : maggior verbosità nella dichiarazione dei tipi,

TECNOLOGIE SICURE: SANDBOX in java parte 2

Meccanismo che offre un set di risorse per testare applicazioni (non attendibili) **in uno spazio limitato** *senza rischiare di infettare il dispositivo dove viene eseguita l'applicazione*

JAVA 1.0

APPLETS: Ogni codice eseguito scaricato da internet considerato malizioso "untrusted", non può accedere ai FS/eseguire programmi locali → eseguito in un sandbox



SANDBOX costituita dalla classi

BYTECODE VERIFIER: controlla che il sorgente non esegua conversioni ILLEGALI , viola restrizioni d'accesso,

SECURITY MANAGER: definisce metodi di controllo dei permessi di sicurezza chiamati dal sistema es: checkRead(stringa)

CLASS LOADER: Carica effettivamente le classi

JAVA 2.0 Introduce il **controllo di accessi** basato su **SECURITY POLICIES** e **PERMESSI Politica**: flexible access control e fine-grained access control

➔ Applet può accedere ad alcune risorse fuori dalla sandbox

CLASSI:
code source: URL dove proviene il codice: IMMUTABILE
Permission: ai file ai socket…
Protected: definisce l'insieme dei permessi-URL dati alla classe
Policy: Dato un codeSource restituisce un PermissionCollection

CARICAMENTO CLASSE E CONTROLLO RUN TIME

VULNERABILITÀ DEL SOFTWARE parte 1

POLITICA: Definisce le **REGOLE**: "Cosa/da chi è consentito o no" e stabilisce **procedura di TESTING** (necessarie a certificare che l'applicazione può girare in rete in modo sicuro) garantita e implementata dal **meccanismo.**
Per non esserci vulnerabilità le politiche non devono entrare in conflitto (**composizione di politiche)**

ANALISI DI SICUREZZA: Bilancia sicurezza e funzionalità;
È indispensabile per decidere quali obietti realizzare e quali proprietà e qualità garantire
Processo **Ortogonale** rispetto al processo di sviluppo SW viene svolta dalla figura **INGEGNERE** DEL SW

Modello ottimale: **A SPIRALE**
Al suo interno prevede l'analisi dei rischi
Flessibile , consente il raffinamento successivo e integrazione nelle fasi successive.

ANALISI DI SICUREZZA: POLICY → SPECIFICATION → DESIGN → IMPLEMENTATION → OPERATION

- Definizione **SECURITY PLAN:** dettagliato, formale, eseguibile **integrato dall'analisi dei rischi**
  proteggere COSA(Asset), da CHI, per QUANTO tempo (requisiti NON funzionali), cosa/perché il sistema fa? Es "I numeri delle carte di credito vanno protette contro potenziali furti perchè sono informazioni delicate"

- **ANALISI RISCHI** Identificare i possibili rischi, classificarli e testarli (allocando risorse) in base alla loro severità (context-sensitive) e valutare strategie per prevenirli/affrontarli. Influisce sul processo di auditing. Non esiste una metrica; solo linee guida per la valutazione e comparazione dei sistemi. **(Lv. Protezione)**

- Fase di **DESIGN:** attuazione del security plan (specifiche) nella comunicazione tra componenti per garantire integrità, confidenzialità, privacy.

- **Stesura CODICE** applicare metodi formale per la corretta del SW, predisporre tecniche per controllare il codice.

- **TESTING: funzionale** "fa ciò che suppone?" , **sicurezza** "immedesimarsi in un cracker tenendo presente ***il CODE COVERAGE:*** se esistono parti di codice non eseguite potenzialmente sfruttabili."

CICLO VULNERABILITA': solitamente dopo la distribuzione del SW. Ortogonale rispetto ad alla vita SW.

Sequenza di eventi: Determinazione, analisi, individuazione soluzione e testate in ambiente locale controllato, distribuzione **del patch** (update automatico) per l'eliminazione della vulnerabilità.

FASI DI VITA DEL SOFTWARE parte 1

PROCESSO DI PRODUZIONE: **Sequenza di attività** svolte per progettare,realizzare,consegnare, modificare un prodotto
CICLO DI VITA: Insieme di **stadi/fasi** in cui il sistema viene a trovarsi

FASI DI SVILUPPO

- **STUDIO FATTIBILITA':** produzione **FSD** (Feasibility Study Document) contenente definizione del problema e relative soluzioni con rapporti costi/benefici (risorse, tempo…)

- **SPECIFICA/ANALISI:** stabilire le qualità che devono essere raggiunte, accordo tra produttore/committente

  Necessario prima comprendere **la CLASSIFICAZIONE DEI REQUISITI:**
  - **FUNZIONALI:** l'**interfaccia** tra applicazione e ambiente esterno, il **Dominio** dell'applicazione, identificare **stakeholder** comprendendo le loro aspettative;
  - **NON funzionali:**
  - **Del processo/manutenzione.**

  QUALITA'/PROPRIETA':
  - **CHIAREZZA, NON AMBIGUITA':**
  - **CONSISTENZA:** Non deve contenere contraddizioni
  - **COMPLETEZZA:** Dettagliate/complete
  - **COMPRENSIBILITA':** intuita/visuale [Assicurarsi di aver compreso bene specifiche]
  - **INCREMENTABILITA':** Feedback, sviluppata in + passi, **E' un processo incrementale:** continua interazione con l'utente **5W: What** will it provide? Where? When?

Produzione del **RASD** (Requirements Analysis and Specification Document) contenente funzionalità/proprietà in termini di performance, facilità d'uso, comprensibile, completo, coerente, modificabile

- **PROGETTAZIONE:** produzione di un documento contenente la descrizione dell'architettura SW(globale/singoli moduli)
  Fase **Design Document**: Def. Moduli e relazioni/interazioni tra componenti con separazione delle responsabilità

- **IMPLEMENTAZIONE:** dai programmatori: CODIFICA + TEST + SPECIFICA TEST con **INTEGRAZIONE** dei vari codici

- **CONSEGNA:** **BETA TESTING** cerchia ristetta utenti "in casi reali"per correggere eventuali errori prima della distribuzione Effettiva nella quale per migliorare il SW va rilasciato un patch specifico.

- **MANUTENZIONE** Correzione + evoluzione

MODELLI DI PROCESSO determinano l'**organizzazione** delle diverse fasi di sviluppo

- **CASCATA**/Waterfall: Output di ogni fase viene **COMPLETATA** e data in input alla fase successiva (processo black box). Non tiene in considerazione che i requisiti possano cambiare.

- **EVOLUTIVO** INCREMENTALE : caratterizzato dalla **NON SEQUENZIALITA'** delle fasi; basato sulla **Prototipazione:** realizzazione di un versione semplificata con le quali sperimentare funzionalità , Oppure **fasi di release** (BETA testing before release before feedback)
  "DO IT TWICE": modifico il progetto e riscrivo il codice. **O metodi agili** (obiettivo utente/tempo)

- **SPIRALE** ITERATIVA Ingloba la prototipizzazione: un processo ciclico di tutte le fasi (PIANIFICAZIONE → **ANALISI RISCHI** → SVILUPPO → VERIFICA) ad ogni ciclo il costo aumenta ma migliora sempre di più il SW incrementale in quanto il vero prodotto si ottiene solo alla fine di tutta l'aspirale.

STAKEHOLDER Qualsiasi soggetto **interessato** nel progetto in corso quali fornitori/clienti/finanziatori/collaboratori/esperti
ARTIFATTO **Sottoprodotto** realizzato durante lo sviluppo SW es:codice sorgente, documentazione che la progettazione.

CRITERI E CLASSI DI VALUTAZIONE parte 1

***Criteri* di valutazione**:

- Def. Una metodologia, complesso di artifatti per misurare l'affidabilità del **prodotti** (che deve soddisfare requisiti security class dell'ORANGE BOOK USA e protection file dei COMMON CRITERIA), che formano un **sistema** valutato da un attore indipendente dal costruttore.

***Classi* di valutazioni**

- **Politiche di sicurezza**: **mandatorie** (vincoli es Bell-LaPadula) / **discrezionali** (tramissione permessi)
- **Etichettatura oggetti**
- **Identificazione soggetti** (autenticazione)
- **AUDITING**
- **Documentazione** (test)
- **Affidabilità** (meccanismi attivi per evitare il blocco del sistema)

Common Criteria Definiscono un insieme di classi e di componenti progettate per essere opportunamente combinate per definire profili di protezione per ogni tipo di prodotto IT, incluso hardware, firmware, software

PROCESSO VALUTAZIONE per la certificazione in base ai common criteria
Produzione profilo di protezione in base a determinate specifiche: questo viene pubblicato (target di sicurezza)che viene utilizzato da un vendor realizzando un target di valutazione che verrà inviato e approvato in base alle common criteria da un istituzione accreditata ed infine dal **NVLAP** (National Volontary Lavoratory Accreditation Program)

AUTOMA FMS (finite state machine) parte 1

rappresentano il **comportamento di un sistema** attraverso:

S: nodi (stati)

I: archi: (passaggi di stato)

δ : possibili combinazioni (transizioni)

FMS con output:

MEALY per ciascuna **transizione** S, I, **O**, δ, **λ** *con λ funzione di output*

MOORE per ciascuno stato S, I, O, δ, λ, **F** *con F insieme di stati finali*

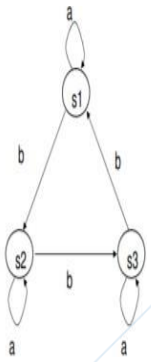
RAPPRESENTAZIONE GRAFICA: **FMS**

Esempio:

S = {s1, s2, s3}

I = {a, b}

δ= { (s1,a,s1), (s1,b,s2),
(s2,a,s2), (s2,b,s3),
(s3,a,s3), (s3,b,s1)}



FMS mealy

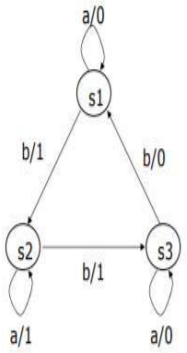
S = {s1, s2, s3}

I = {a, b}

O = {0, 1}

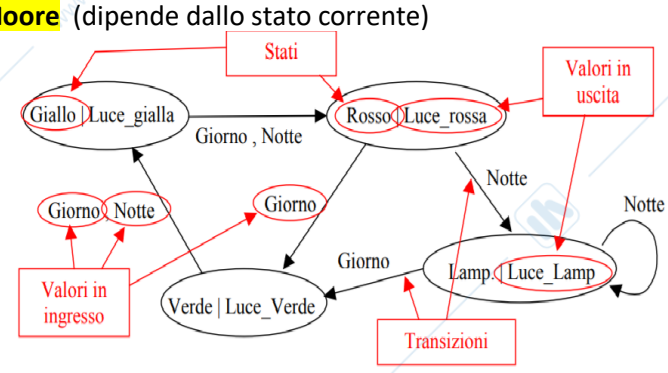
δ = { (s1,a,s1), (s1,b,s2),
(s2,a,s2), (s2,b,s3),
(s3,a,s3), (s3,b,s1)}

λ = { (s1,a,0), (s1,b,1),
(s2,a,1), (s2,b,1),
(s3,a,0), (s3,b,0)}

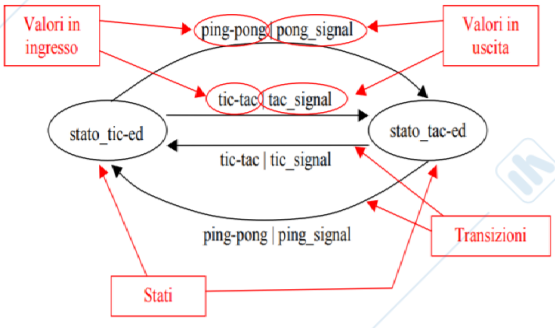


Rappresentazione con **STG** State Transition Grapher

Moore (dipende dallo stato corrente)

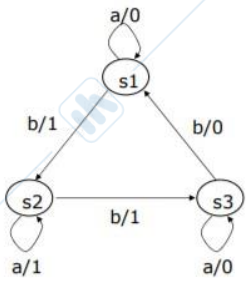


Mealy (“**ciclico**”) “meno stati”



FMS intendo d'ora in poi FMS con output di MEALY
con parametri [S, I, **O , T**] con **T** possibili transazioni e **O** possibili valori in output

S = {s1, s2, s3}
I = {a, b}
O = {0, 1}
T = {(s1,a,0,s1),
(s1,b,1,s2),
(s2,a,1,s2),
(s2,b,1,s3),
(s3,a,0,s3),
(s3,b,0,s1)}



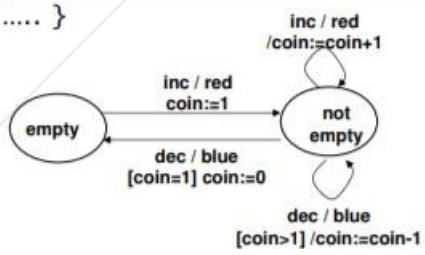
LIMITI: Possibile rappresentare solo un numero finito di stati.

Extended FMS [S,I,O,V,T]

con t [s , i , o , **g** , **a** , **s**] **g** guardia, **a** azione, **s** target

Salvadanaio elettronico: formalmente

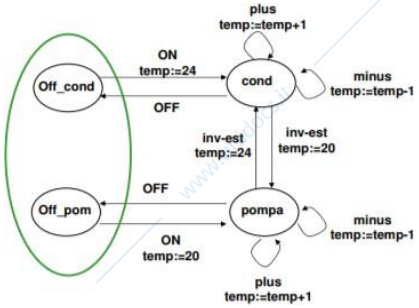
- S = {empty, not-empty}
- I = {inc, dec}
- O ={red, blue}
- V ={coin}
- T ={(empty, inc, red, coin:=1, not-empty),
(not-empty, dec, blue, coin=1, coin:=0, empty),
..... }



STATO GLOBALE: (S, V = σ) (stato, variabile = valore) 7 NODI
(empty, coin=0) **(STATO, VARIABILEGLOBALE = VALORE)**

TRANSIZIONE GLOBALE: (S,σ), I, O, (S', σ') ARCHI
(empty, coin=0), inc, red, (notempty, coin=1) STATO VA IN UN ALTRO ATTRAVERSO OPERAZIONE INC

Modellare con una EFSM il comportamento di un dispositivo che funziona come pompa di calore e condizionatore.
Il dispositivo è inizialmente spento e viene settato come pompa di calore o come condizionatore dal tasto inverno/estate.
Se spento, il dispositivo memorizza l'ultima modalità.
Se il dispositivo funziona come pompa di calore, la temperatura viene settata a 20 gradi e può essere aumentata o diminuita mediante telecomando.
Se il dispositivo funziona come condizionatore, la temperatura viene settata a 24 gradi e può essere aumentata o diminuita mediante telecomando.



CFSM Communicating

modellano un insieme di FSM e comunicano tra loro Utilizzato in insiemi EMBEDDED (incorporati) e protocolli di comunicazione

Tupla (**C,P**) *con C canali e P numero FMS (con S,I,O,T)*

CFSM **temporale**: Le transazioni possono contenere l'info. Dell'iintervallo di tempo [t1,t2] entro cui l'azione può avvenire (dopo essere entrati nello stato)

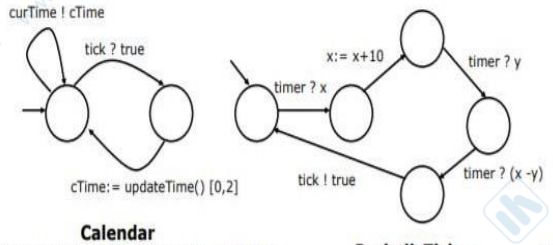
Modellare un calendario elettronico che indica l'ora e la data corrente, con una CFSM estesa e temporale.

Tramite il canale di IO *curTime*, un utente (da non modellare) può accedere a data ed ora corrente che sono memorizzate in *cTime*.

La macchina *Calendar* riceve un tick da un processo *Periodic Ticker* ed ad ogni tick aggiorna l'ora e la data cor un comando interno nel tempo di 2 istanti di clock.

Periodic Ticker è un processo che invia tick ogni 10 secondi. Fa uso di un clock interno (da non modellare) che fornisce il valore del tempo corrente attraverso il canale *timer* (timer?y assegna il tempo globale corrente alla variabile y). *Periodic Ticker* usa inoltre l'espressione (x - y) per ritardare fino allo start del successivo periodo.

Processi: **Calendar**, **PeriodicTicker**
Canali: **curTime** (tra Calendar e utente)
tick (tra Calendar e PeriodicTicker)
timer (tra PeriodicTicker e clock interno)
Variabili: **cTime** (di Calendar) memorizza la data e l'ora corrente
x e **y** (di PeriodicTicker) memorizzano il tempo corrente



COPERTURA parte 1

MALFUNZIONAMENTO?

DIFETTO?

FAILURE: Il programma non funziona come previsto

FAULT: parte del sorgente che non corrisponde alle aspettative

ERRORE: il motivo del difetto

Statement coverage: ogni istruzione viene eseguita almeno una volta
MCC Multiple Condition Coverage
MCDC *Modified* Condition ***Decision*** Coverage

DESIGN BY CONTRACT D b C parte 2

PROGETTAZIONE PER CONTRATTO : Metodologia utilizzata da clienti/fornitori attraverso un contratto che definisce specifiche

TESTING parte 3

5.1 Descrivere cosa si intende per program-based testing e specification-based testing. Definire i criteri per la copertura del program-based testing.

5.2 Definire cosa sono i test di accettazione, conformità e integrazione.

5.3 Descrivere le possibili classificazioni di testing a seconda del livello a cui si effettua, del tipo di accesso e degli aspetti da testare.

5.4 Classificare l'attività di testing rispetto ai requisiti, all'architettura ed al codice.

5.5 Definire la differenza tra testing funzionale e testing di sicurezza in sistemi software sicuri.

Dare la definizione di criterio affidabile, valido e ideale. Enunciare il teorema di Goodenough e Gerhart.

Dare la definizione di criterio valido, affidabile ed ideale. Determinare un criterio ideale per il seguente programma che dovrebbe restituire il valore della funzione f(x,y)=x*(y*10) per valori x,y interi, crescenti e maggiori di 1, 0 altrimenti, ma che contiene un errore:

```
int foo (int x, int y) {
    if (x>1 & y>x) return x*(y+10); else return 0;
}
```