Ingegneria del Software mod. A
Produzione di software critico

Docente: Tullio Vardanega
tullio.vardanega@math.unipd.it

Sicurezza di sistema

- La nozione di sicurezza (safety) di un sistema [*software*] emana da e si arricchisce con l'analisi degli incidenti [*software*]

Joint Software System Safety Committee
SOFTWARE SYSTEM SAFETY HANDBOOK
A Technical and Managerial Team Approach

dicembre 1999

Sicurezza: definizioni - 1

- Lo standard MIL-STD 882B (1984) definisce la sicurezza in termini di attributo di sistema

> L'assenza di (intesa come libertà da) condizioni capaci di causare danni distruttivi o severi alle persone, alle cose od alle proprietà

Sicurezza: definizioni - 2

- Lo stesso standard definisce la sicurezza di sistema anche in termini di processo

> L'applicazione di principi, criteri e tecniche ingegneristici e gestionali per ottimizzare le caratteristiche di sicurezza del sistema nel rispetto dei vincoli di efficienza d'uso, di produzione e di costo, durante tutte le fasi del ciclo di vita del sistema

Sicurezza: definizioni - 3

- Due termini con significato distinto
  - Sicurezza (*safety*): i requisiti di sicurezza tendono a rendere il sistema incapace di produrre danni catastrofici
  - Affidabilità (*reliability*): riguarda la prevenzione di ogni tipo di errore che possa condurre ad un guasto di sistema
- Ai fini della sicurezza non è importante prevenire ogni guasto, ma assicurare che quelli che avvengano abbiano conseguenze tollerabili
  - Gli obiettivi di sicurezza e di affidabilità possono confliggere

Livelli di criticità - 1

- Molti standard di settore assegnano ai sistemi un livello di criticità che dipende da:
  - La severità del danno conseguente ad un guasto di sistema
  - La probabilità di occorrenza del guasto
- Al *software* in esecuzione sul sistema viene attribuito il livello di severità dei danni che possono risultare dal suo malfunzionamento

Livelli di criticità - 2

La **FAA** (*Federal Aviation Authority*) riconosce 5 categorie di guasto ed altrettanti livelli di criticità *software*

Effetto del guasto	Livello
Catastrofico	A
Rischio elevato	B
Rischio significativo	C
Rischio trascurabile	D
Senza effetto	E

Standard di settore - 1

- Aeronautica
  - **RTCA**
    - Originariamente: *Radio Technical Commission for Aeronautics*
    - Oggi: *Requirements and Technical Concepts for Aviation*
    - A composizione mista industria-governo
    - Emette linee guida e requisiti standard

Standard di settore - 2

- Aeronautica
  - **RTCA** (segue)
    - Una delle commissioni di lavoro affiliate all'RTCA (la SC-167) è responsabile per la preparazione e la revisione di standard per la certificazione del *software* aeronautico
    - Nel dicembre 1992, la commissione SC-167 ha prodotto un documento denominato DO-178B che regola la fornitura di materiale atto ad ottenere l'approvazione di FAA per *software* di volo

Standard di settore - 2

- Difesa
  - Nell'agosto 1997, il dipartimento della difesa inglese (MoD) ha prodotto uno standard denominato Def-Stan 00-55 che regola l'acquisizione di *software* con caratteristiche di sicurezza
  - Def-Stan 00-55/56 hanno assunto forma di norma Europea sotto forma di IEC 61508

Standard di settore - 3

- Automobilistico
  - **MISRA** (*Motor Industry Software Reliability Association*)
    - È a composizione mista tra industrie automobilistiche, fornitori di componenti ed enti universitari
    - Emana linee guida per la produzione di software con caratteristiche di sicurezza, che applicano al suo intero ciclo di vita
    - Una vasta parte delle linee guida concerne l'uso di linguaggi di implementazione e dei compilatori

Standard di settore - 4

- Nucleare
  - Nel 1986, la **IEC** (*International Electrotechnical Commission*) ha pubblicato uno standard denominato IEC 880 (*Software for Computers in the Safety Systems of Nuclear Power Stations*)
  - Lo standard fornisce linee guida per la selezione del linguaggio di implementazione

Ambiti di criticità

- **Economica** (*business-critical*)
  - Un guasto comporta perdite economiche
- **Applicativa** (*mission-critical*)
  - Un guasto comporta perdita o degrado della missione
- **Di sicurezza** (*safety-critical*)
  - Un guasto comporta danni distruttivi a persone o cose

IEC61508 (Def-Stan 00-55/56) Safety Complexity Integrity Levels

SCIL Level	Consequences of Software Failing
4	**Death of one or more persons, significant financial loss** *(Areas: flight-critical aerospace, life-critical medical systems, transport control systems, hazardous process control systems, automotive breaking systems)*
3	**Serious injury or financial loss** *(Areas: automotive engine management)*
2	**Inconvenience or disappointment to the public** *(Areas: small consumer goods, point of sale equipment)*
1	**No inconvenience** *(Areas: student project, research)*

MISRA Integrity Levels

Integrity Level	Controllability by vehicle occupants	Acceptable Failure Rate	*Examples of Software Failure*
4	Uncontrollable	Extremely improbable	*Loss of power assisted steering*
3	Difficult to control	Very remote	*Braking system failure*
2	Debilitating	Remote	*Windshield wiping system failure*
1	Distracting	Unlikely	*Electrical window system failure*
0	Nuisance Only	Possible	*Radio/CD system failure*

Software Security Standards

- TCSEC (Orange Book)
  - *Trusted Computer Security Evaluation Criteria*
- *Common Criteria For Information Technology Security Evaluation* (ISO/IEC 15408-1)
  - Criteri di valutazione di sicurezza *software*
  - 7 livelli di (criticità di) sicurezza

Evaluation Assurance Levels

EAL	Constraints on the Software Development
7	Formally Verified Design & Tested
6	Semi-formally Verified Design & Tested
5	Semi-formally Designed & Tested
4	Methodically Designed, Tested & Reviewed
3	Methodically Tested and Checked
2	Structurally Tested
1	Functionally Tested

Caratteristiche del linguaggio di programmazione - 1

- Un linguaggio adatto per la realizzazione di sistemi con caratteristiche di sicurezza possiede, almeno, le seguenti caratteristiche:
  - È definito mediante uno standard internazionale (p.es. ISO)
    - Consente di accertare l'adeguatezza (*conformance*) del compilatore
  - Ha struttura modulare e supporta progettazione e codifica strutturate
    - Facilita l'astrazione ed il riuso di componenti verificate ed affidabili
  - Accelera la rilevazione di errori
    - Particolarmente a livello di compilazione

Caratteristiche del linguaggio di programmazione - 2

- (segue)
  - Consente accesso logico e strutturato agli elementi *hardware* del sistema
    - Ha costrutti che rappresentano le caratteristiche salienti dell'*hardware* sottostante, p.es. l'associazione di componenti di strutture logiche a registri
  - Fornisce controllo rigoroso sulla visibilità di tipi, operazioni e dati ai moduli del programmi
    - Permette di determinare regole di accesso

Certificazione - 1

- Certificazione della sicurezza di una applicazione
  - L'applicazione comprende il supporto a tempo di esecuzione (sistema operativo) del quale fa uso
  - Consiste nel verificare che sviluppo e verifica siano state condotte secondo standard ingegneristici (derivati p.es. da ISO 12207) e di sicurezza (p.es. DO-178B) rigorosi ed adeguati al dominio

Certificazione - 2

- I processi *software* utilizzati determinano la certificabilità dell'applicazione
- Uso preferenziale di un modello di sviluppo sequenziale
- I processi utilizzati sono fortemente orientati/e alla produzione di documentazione di supporto (evidenza di certificazione)

Certificazione - 3

- Pianificazione della documentazione di supporto alla certificazione
  - Piano per gli aspetti *software* concernenti la certificazione
  - Piano di sviluppo del *software*
  - Piano di qualifica del *software*
  - Piano di gestione della configurazione
  - Piano di accertamento della qualità

Certificazione - 4

- Uso di standard di processo consolidati in ogni fase di sviluppo
  - Standard per la definizione dei requisiti *software*
  - Standard per la progettazione (*design*) del *software*
  - Standard di codifica
  - Non esistono standard riconosciuti per la qualifica, ma solo linee guida ed obiettivi!

Certificazione - 5

- Una vasta quantità di informazione deve essere collezionata come evidenza (analizzabile) di esecuzione dei processi adottati
  - Specifiche ed analisi dei requisiti *software*
  - Descrizione e definizione del prodotto *software*
  - Codice sorgente ed eseguibile
  - Definizione, realizzazione e risultato delle prove
  - Catalogo del sistema di configurazione
  - Lista dei problemi
  - Resoconto delle azioni di gestione e controllo di qualità

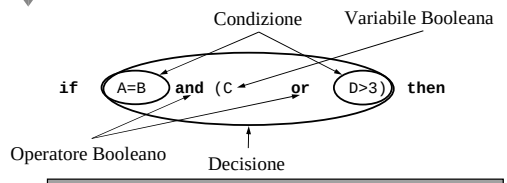
Certificazione - 6

- Strategie di verifica dinamica (prova, *test*)
  - Black box: verifica che ciascuna funzione produce il risultato atteso in tutte le possibili condizioni in cui essa possa eseguire
    - Il test include l'uso di valori tipici e di valori "ai margini" (condizioni estreme)
  - White box: sulla base della struttura della funzione da testare, assicura che tutti i suoi elementi e flussi di esecuzione siano necessari e di comportamento verificato
    - Il test deve assicurare che il programma esegue correttamente in ogni condizione e che tutte le condizioni logiche in esso contenute eseguano correttamente in ogni cammino

Esempio: prove di copertura di condizioni e decisioni - 1

- DO-178B definisce
  - Condizione
    un'espressione Booleana semplice, che non contiene operazioni Booleane
  - Decisione
    una espressione composta, contenente una o più condizioni combinate mediante operatori Booleani
- DO-178B richiede
  - Che tutte le decisioni vengano provate e tutti i rispettivi esiti siano prodotti
  - Che ciascuna condizione all'interno di una decisione assuma entrambi gli esiti (vero/falso) almeno una volta

Esempio: prove di copertura di condizioni e decisioni - 1



Occorre verificare se e come ogni singola condizione o variabile possa da sola determinare l'esito della decisione. Non basta provare l'intera espressione come vera o falsa una sola volta!

Esempio: prove di copertura di condizioni e decisioni - 3

`if A=B and (C or D>3) then …`

Caso	Condizione/Variabile			Esito
	A=B	C	D>3	
1	T	F	F	F
2	T	T	F	T
3	T	F	T	T
4	F	F	T	F

Certificazione - 7

- Prove di copertura
  - Si richiede che il *software* installato nel suo ambiente d'esecuzione soddisfi tutti i requisiti
  - Non si ammette la presenza di componenti *software* che non corrispondano a requisiti
    - Il rigore di questa prescrizione dipende dal livello di criticità assegnato al *software*
      - Livello A: corrispondenza tra requisiti e codice eseguibile
      - Livello B: corrispondenza tra requisiti e codice sorgente

Certificazione - 8

- Tecniche di verifica
  - Tracciamento
    - Per determinare completezza, rilevare omissioni, duplicazioni ed elementi superflui
  - Revisioni
    - Formali / informali (come da ISO/IEC 12207)
  - Analisi
    - Statica: applica a requisiti, disegno e codice
    - Dinamica (prova): applica a singole componenti del sistema od al sistema nella sua interezza

Certificazione - 9

- Analisi statica
  - Analisi di flusso di controllo
    assicura che il programma esegua nell'ordine atteso; che il codice sia ben strutturato; che non vi siano parti del codice irraggiungibili; localizza problemi di terminazione (p.es.: ricorsione, cicli infiniti)
  - Analisi di flusso dei dati
    assicura che nessun cammino di esecuzione acceda a variabili non inizializzate

Certificazione - 10

- Analisi statica (segue)
  - Analisi del flusso dell'informazione
    determina come l'esecuzione di una unità di codice crei dipendenze tra il suo ingresso e la sua uscita

```
X = A+B;          // X dipende da A e B
Y = D-C;          // Y dipende da C e D
if (X>0)
   Z = (Y+1);     // Z dipende da A, B, C e D
```


Certificazione - 11

- Analisi statica (segue)
  - Verifica formale del codice
    prova che il codice di un programma sia corretto rispetto alla specifica formale dei suoi requisiti (conservazione delle proprietà attese)
  - Verifica di limite (*range check*)
    verifica che i dati manipolati dal programma restino entro i limiti del loro tipo e dell'accuratezza richiesta
    - P.es.: *overflow*, arrotondamento, limiti di *array*

Certificazione - 12

- Analisi statica (segue)
  - Analisi d'uso di *stack*
    [Alcuni linguaggi usano una zona di memoria (detta *stack*) per consentire a sottoprogrammi di condividere e preservare dati ed informazioni di contesto]
    l'analisi determina se l'ampiezza dello *stack* disponibile sia sufficiente per l'uso che ne può fare l'esecuzione del programma

Certificazione - 13

- Analisi statica (segue)
  - Analisi temporale
    concerne le proprietà temporali richieste ed esibite dalle dipendenze delle uscite dagli ingressi del programma
  - Analisi di codice oggetto
    dimostra che il codice oggetto da eseguire sia una traduzione corretta del codice sorgente corrispondente e che nessun errore (od omissione) sia stato introdotto dal compilatore

Ringraziamenti

Le pagine 12-16 di questa lezione sono un adattamento di materiale didattico offerto da Franco Gasperoni di ACT Europe e pubblicato al sito http://libre.act-europe.fr/Software_Matters secondo i termini della licenza *GNU Free Documentation*