Produzione di *software* critico

IS

Anno accademico 2007/8
Ingegneria del Software mod. A

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 come attributo di sistema
  - L'assenza di (intesa come libertà da) condizioni potenzialmente capaci di causare danni distruttivi o severi
    - Alle persone
    - All'ambiente
    - 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
  - Per massimizzare le caratteristiche di sicurezza del sistema
  - Nel rispetto dei vincoli d'efficienza di produzione, d'uso e di manutenzione
  - Durante tutte le fasi del ciclo di vita del sistema

Sicurezza: definizioni – 3

- Due termini con significato distinto
  - Sicurezza intesa come *safety* (altro da *security*)
    - I requisiti di sicurezza tendono a rendere il sistema dimostrabilmente **incapace** di produrre **danni catastrofici**
  - Affidabilità (*reliability*)
    - Riguarda la prevenzione di **ogni tipo di errore** che possa condurre a un **guasto** di sistema inteso come deviazione dal comportamento atteso
- Ai fini di *safety* non è importante prevenire ogni guasto ma assicurare che quelli che avvengano abbiano solo 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 a un guasto di sistema
  - La probabilità di occorrenza del guasto
- Al *software* di sistema viene attribuito il livello di criticità dei danni che possono risultare dal suo malfunzionamento

Università degli Studi di Padova

Produzione di *software* critico

Livelli di criticità – 2

- **La FAA (*Federal Aviation Authority*) riconosce 5 categorie di severità di guasto ed altrettante di criticità del sistema *software***

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

Università degli Studi di Padova

Produzione di *software* critico

Standard di settore – 1

- **Sistemi aeronautici**
  - **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
    - 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 soggetto all'approvazione di FAA per l'utilizzo in sistemi di bordo

Università degli Studi di Padova

Produzione di *software* critico

Standard di settore – 2

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

Università degli Studi di Padova

Produzione di *software* critico

IEC 61508 – 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** Application area: flight-critical aerospace, life-critical medical systems, transport control systems, hazardous process control systems, automotive breaking systems
3	**Serious injury or financial loss** Application area: automotive engine management
2	**Inconvenience or disappointment to the public** Application area: small consumer goods, "point of sale" equipment
1	**No inconvenience** Application area: student project, non-hazardous research

Università degli Studi di Padova

Produzione di *software* critico

Standard di settore – 3

- **Sistemi automobilistici**
  - **MISRA (*Motor Industry Software Reliability Association*)**
    - 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 programmazione e dei loro compilatori
      - Per esempio "*safe* C"

Università degli Studi di Padova

Produzione di *software* critico

Livelli di integrità secondo MISRA

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

Università degli Studi di Padova

Produzione di *software* critico

Standard di settore – 4

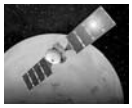
- **Sistemi nucleari**
  - **Nel 1986, la IEC ha pubblicato uno standard denominato IEC 880**
    - *Software for Computers in the Safety Systems of Nuclear Power Stations*
  - **Lo standard fornisce linee guida per la selezione del linguaggio di programmazione per lo sviluppo del *software* di controllo delle centrali**

Università degli Studi di Padova

Produzione di *software* critico

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**



Università degli Studi di Padova

Produzione di *software* critico

Standard di sicurezza *software*

- **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**

Università degli Studi di Padova

Produzione di *software* critico

Livelli di accertamento a fini di valutazione

- **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

Università degli Studi di Padova

Produzione di *software* critico

Software dependability

- **Poter confidare nel funzionamento sicuro e affidabile di un sistema *software***
- **Confidenza che si ottiene tramite**
  - ***Fault avoidance***
    - I processi produttivi adottati assicurano l'assenza di guasti logici o fisici
  - ***Fault removal***
    - I processi di qualifica (V&V) sono capaci di rilevare eventuali guasti logici o fisici nel sistema e quindi condurre alla loro rimozione
  - ***Fault tolerance***
    - Il progetto del sistema assume la permanenza di alcuni guasti logici e fisici (***fault model***) e dispone di mezzi operativi per rilevarli e trattarli a tempo d'esecuzione prima che possano avere effetto sul sistema

Università degli Studi di Padova

Produzione di *software* critico

Definizioni

- **Guasto** ⇒ ***Fault***
  - **Atto od omissione, possibile causa (anche umana) di comportamento fuori norma** — **programmatore distratto**
  - **Può non avere effetto**
- **Errore o Difetto** ⇒ ***Error***
  - **Stato erroneo di sistema introdotto da un guasto** — **dato erroneo in una procedura**
  - **Può restare latente, va eliminato**
- **Malfunzionamento** ⇒ ***Failure***
  - **Effetto di un errore, risulta in deviazione da specifica sbagliato** — **emissione di un valore**
  - **Può causare danni rilevanti**

Università degli Studi di Padova

Produzione di *software* critico

Software privo di difetti

- I processi di sviluppo attuali e la loro tecnologia di supporto permettono di realizzare sistemi *software* privi di difetti
  - Certamente almeno per sistemi di modeste dimensioni
- Per il *software* essere privo di difetti significa essere completamente conforme alle sue specifiche
  - Assumendo che esse siano corrette
  - Quindi è importante sforzarsi di produrre specifiche corrette!
- Lo sviluppo di *software* privo di difetti può essere più costoso del pagare le conseguenze dei guasti prodotti da *software* difettoso
  - Questo è però completamente inaccettabile in domini soggetti a vincoli di criticità

Università degli Studi di Padova

Produzione di *software* critico

Strumenti di facilitazione

- Processi ingegneristici affidabili
- Gestione della qualità
- Specifiche rigorose (magari formali)
- Verifiche statiche
- Linguaggi di programmazione con tipi forti
- Stili di programmazione sicuri e difensivi
- Incapsulazione e protezione dei dati critici

Università degli Studi di Padova

Produzione di *software* critico

Stili di programmazione

- Usare costrutti e tecniche che contribuiscano attivamente a evitare, rilevare e trattare i difetti
  - Progettare cercando la semplicità architetturale e algoritmica
  - Assicurare che i dati sensibili sia protetti dagli accessi incontrollati
  - Evitare o ridurre al minimo l'uso di costrutti rischiosi

Università degli Studi di Padova

Produzione di *software* critico

Protezione dell'informazione

- Incapsulazione
  - L'informazione dovrebbe essere esposta solo alle parti del programma che hanno bisogno di accedervi
  - Questa precauzione richiede la creazione e la gestione di entità dati (oggetti o tipi di dato astratto) che incapsulino e mantengano l'informazione statica e forniscano operazioni per manipolarla in modo controllato
- Tre tipi di benefici
  - Si riduce il rischio che parte dell'informazione sensibile venga acceduta e corrotta accidentalmente
  - L'informazione sensibile e le operazioni utili a manipolarla sono incapsulate e controllate e dunque è più basso il rischio che esse possano corrompere il resto del sistema
  - L'informazione sensibile è localizzata e dunque è più agevole esaminarla in sede di verifica per accertarne la correttezza

Università degli Studi di Padova

Produzione di *software* critico

Costrutti a rischio di errore – 1

- Numeri in virgola mobile
  - Sono imprecisi per definizione
  - L'approssimazione può cumularsi e condurre a errori importanti e a confronti svianti
- Puntatori
  - L'accesso erroneo e non controllato a zone di memoria può corrompere i dati in modo subdolo e irreparabile
  - L'uso di *aliasing* rende i programmi difficili da comprendere e modificare
- Allocazione dinamica della memoria
  - Può portare a esaurimento della disponibilità e alla sovrapposizione di aree sensibili
- Ricorsione
  - Può portare a esaurimento della memoria oppure alla non terminazione
- Concorrenza
  - Se mal progettata può condurre a situazioni di malfunzionamento sottili e difficili da rimediare

Università degli Studi di Padova

Produzione di *software* critico

Costrutti a rischio di errore – 2

- Gestione delle interruzioni
  - Potendo esercitare prerilascio sulle attività normali e operando nell'ambiente del processo prerilasciato può causare effetti indesiderati
- Ereditarietà
  - Il codice funzionale non è più localizzato in un singolo modulo ma "sparso" in moduli diversi e distinti. Questo può facilitare l'insorgenza di errori a fronte di modifiche locali
- *Aliasing*
  - Usare più di 1 nome per accedere alla stessa informazione rende facile causare effetti laterali inconsapevoli
- Limite dei strutture lineari (*array*)
  - La scansione di *array* senza controllo preventivo sul loro limite può causare errori gravi

Università degli Studi di Padova

Produzione di *software* critico

Trattamento delle eccezioni

- Le eccezioni sono tipicamente sincrone
  - Avvengono in conseguenza di azioni erronee del programma rilevate dal sistema operative
- Le eccezioni asincrone sono causati da eventi anomali esterni al programma
- È largamente preferibile fare uso di costrutti per il trattamento delle eccezioni
  - Piuttosto che usare costrutti di controllo per accertarne l'occorrenza
  - Oltre a onere improprio d'esecuzione comportano l'aggiunta di logica funzionale al programma e quindi il rischio di difetti

Università degli Studi di Padova

Produzione di *software* critico

Caratteristiche del linguaggio di programmazione – 1

- Un linguaggio adatto per la realizzazione di sistemi con caratteristiche di sicurezza possiede almeno i seguenti attributi
  - È definito mediante uno standard internazionale (p.es. ISO)
    - Ciò permette di accertare l'adeguatezza (***conformance***) del compilatore
  - Ha struttura modulare e supporta progettazione e codifica strutturate
    - Facilita l'astrazione e il riuso di componenti verificate e affidabili

Università degli Studi di Padova

Produzione di *software* critico

Caratteristiche del linguaggio di programmazione – 2

- …
  - Facilita la rilevazione di errori
    - Meglio se già a livello di compilazione
  - 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 a tipi, operazioni, risorse

Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 1

- Certificazione di sicurezza di applicazione
  - L'applicazione è considerata includere il supporto a tempo di esecuzione (il 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
- I processi *software* impiegati determinano la "certificabilità" dell'applicazione

Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 2

- Uso preferenziale di un modello di sviluppo sequenziale
- Processi fortemente orientati alla produzione di documentazione di supporto
  - Perché la documentazione fornisce evidenza di certificazione

Università degli Studi di Padova

Produzione di *software* critico

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

- **Impiego 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 programmazione e di codifica**
- **Non esistono standard riconosciuti per la qualifica ma solo linee guida e obiettivi!**

Certificazione – 5

- **Una vasta quantità di informazione deve essere collezionata come evidenza ispezionabile di esecuzione dei processi adottati**
  - **Specifiche e 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**

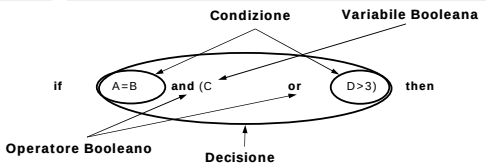
Certificazione – 6

- **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 verificare assicura che tutti i suoi elementi e **cammini di esecuzione** siano **necessari** e abbiano **comportamento verificato**
    - Il *test* deve assicurare che il programma esegua correttamente in ogni condizione e che tutte le condizioni logiche in esso contenute eseguano correttamente in ogni cammino

Esempio: MDCC – 1

- **Definizioni DO-178B**
  - **Condizione**
    - Espressione Booleana semplice che **non** contiene operazioni Booleane al suo interno
  - **Decisione**
    - Espressione composta contenente una o più **condizioni combinate** mediante operatori Booleani
- **Requisiti DO-178B**
  - **Che tutte le decisioni vadano soggette a *test* e tutti i loro possibili esiti siano effettivamente prodotti**
  - **Che ciascuna condizione all'interno di una decisione assuma entrambi gli esiti (vero/falso) almeno una volta**
    - *Modified Decision and Condition Coverage (MDCC)*

Esempio: MDCC – 2



- **Occorre verificare se e come ogni singola condizione o variabile Booleana possa da sola determinare gli esiti possibili della decisione**
- **Non è più sufficiente provare l'intera espressione come vera o falsa una sola volta!**

Esempio: MDCC – 3

`if A=B and (C or D>3) then …`

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

Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 7

- **Prove di copertura (*coverage*)**
  - **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

Università degli Studi di Padova

Produzione di *software* critico

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, *test*): applica a singole componenti del sistema o al sistema nella sua interezza

Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 9

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

Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 10

- **Analisi statica (II)**
  - **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
```


Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 11

- **Analisi statica (III)**
  - **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*)**
    - Accerta che i dati manipolati dal programma restino entro i limiti del loro tipo e dell'accuratezza richiesta
      - P.es.: *overflow*, arrotondamento, limiti di vettore

Università degli Studi di Padova

Produzione di *software* critico

Certificazione – 12

- **Analisi statica (IV)**
  - **Analisi d'uso di *stack***
    - Alcuni linguaggi di programmazione – ma non tutti – usano una zona di memoria (detta *stack*) per consentire a sottoprogrammi di localizzare dati locali e informazioni di contesto
    - L'analisi determina se l'ampiezza dello *stack* assegnato al programma sia sufficiente per l'uso che può farne per qualunque sua storia d'esecuzione
  - **Analisi temporale**
    - Concerne le proprietà richieste ed esibite dalle dipendenze temporali tra le uscite e gli ingressi del programma o di parti di esso
  - **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 diapositive 12 – 16 di questa lezione sono un adattamento del materiale didattico offerto da Franco Gasperoni di AdaCore e pubblicato all'indirizzo**
  - http://libre.act-europe.fr/Software_Matters
  - **Nei termini della *GNU Free Documentation Licence***
- **Le diapositive 17, 19 – 24 sono invece un adattamento del capitolo 20 del testo *Software Engineering* di Ian Sommerville (8a edizione)**