Lesson1

Basics

Sistema:
in questo caso pensiamo ad un sistema generico. In questo corso noi considereremo sistemi cyber-physical ma per il momento rimaniamo sul generale.
Il sistema è nel mezzo fra l'hardware e l'applicazione. A seconda del sistema operativo i driver potrebbero essere dentro il sitema operativo oppure no. In questo caso non andiamo così nel dettaglio, consideriamo un sistema generico.

Ci sono applicazioni che possono fallire senza problemi (*Es:* un gioco sul telefono) e applicazioni che non possono fallire in nessun modo (*Es:* Cruise control su un aeroplano).
Nel secondo caso bisogna cercare di fare in modo che il software non abbia problemi e anche se li ha, che le conseguenze non siano catastrofiche.

Ci sono requisiti che riguardano effettivamente la capacità di non fallire e requisiti di altro tipo (confidenzialità, sicurezza ecc.).
Questo secondo insieme si definisce come software dependabilities.

Validazione: il processo di determinare se quello che abbiamo realizzato corrisponde alle aspettative. (Il sistema rispetta le specifiche?)
Verifica: il processo di determinar se il sistema corrisponde alle specifiche richieste. (Il sistema è ben realizzato?)
Sono basate entrambe su:

- **Specifica:** descrizione dicosa il sistema dovrebbe poter fare
- **Ralizzazione:** descrizione di cosa il sistema è e fa in effettivo

Cose da validare:

- **Proprietà funzionali**
- **Dependability:** capacità di evitare che il servizio fallisca un numero di volte inaccettabile e di garantire che le parti che falliscono più spesso abbiano poche conseguenze sul sitema. (*Esempio:* Cerco di garantire che un sistema dove i treni non si rompono spesso, ma anche se uno si rompe in mezzo ai binari non si hanno conseguenze catastrofiche)

Come dovrebbe quindi funzionare:
Servizio corretto --Fallisce--> Fornisce un servizio erratto
Se il sistema è ben fatto è presente una freccia inversa dato che il sistema è capace di riprendersi (restoration). *Esempio:* in un server si hanno due dischi in RAID, se il primo fallisce si rimuove, si usa il secondo e se ne inserisce uno nuovo che servirà da backup.

Nella realtà ci sono diverse gradi di funzionamento di un servizio. *Esempio:* un treno funziona, cosa deve fare per funzionare correttamento? Quando è che il suo servizio è corretto? Quando si muove in orario e si ferma quando deve? Oppure semplicemente se nessuno muore a bordo? Dipende da

cosa decido di analizzare e da come definisco davvero un sistema che funziona bene.
Altro esempio: Ho un protocollo di rete, posso controllare quanto gestite bene i casi di fallimento, posso controllare he non risponda mai oltre un certo tempo di risposta, posso controllare che non ci siano troppi settori del disco che sono rotti, posso controllare che il software funzioni sempre nel modo corretto. Alcune di queste cose richiedono gradi alti di successo (99.9999%) altre no.

Proprietà di un sistema dependable:

- **Availability:** servizio corretto disponibile sempre
- **Reliability:** il servizio funziona in continuo sempre (senza neanche piccole pause *Es.* Aereo non può avere neanche un secondo down)
- **Cofidenziality:** essere sicuri che nessuno possa leggere le informazioni che non lo riguardano
- **Integrity:** essere sicuri che il sistema rimanga integro (inalterato da persone non autorizzate)
- **Maintanability:** facilità di modificare e riparare il Sistema
- **Coverage:** probabilità che il sistema possa tollerare un problema e comunque fornire correttamente il servizio (Derivata quindi da alcune precedenti proprietà)
- **Safety:** (approfondita dopo)

Come si ottiene la dependability:
ci sono alcuni modi per fare sì che il sistema sia dependable:

- **Prevenzione dei faults:** se so che alcuni input causano problematiche cerco di prevedere gli errori prima di rilasciare il mio software (*Es.* non acquisto software non testato)
- **Rimozione dei faults:** se vedo che degli errori ci sono, li testo e li risolvo
- **Tolleranza ai faults:** (*Es.* ho un incontro di lavoro importante, non uso solo il telefono come sveglia, metto anche un'altra sveglia classica per fare sì che due device debbano fallire affinchè io non mi svegli)

Threats: Ci sono 3 tipi di minacce: Faults, errors e failures.
Un fault causa uno o più erorri che a loro volta portano ad uno o più failures. *Es.* la corrente non va, io provo ad accenderla ma ho un errore dato che la corrente non si accende, decido di andarmene del quadro ma cado perchè non vedo (failure)

Safety: ci sono failures che non creano grandi problemi e altri che sono catastrofici. Io voglio un sistema dove non succedono i secondi. Di solito questo vuol dire che nessun uomo, infrastruttura e ambiente che viene danneggiato dal nostro sistema. *Es.* il mio sistema proetegge gli uomini ma causa 10 ettari di distruzione di un bosco, questo è comunque catastrofico.
Un sistema safety critycal è un sistema in cui io devo garantire di non avere failure catastrofici.

Safety e Availability:
È facile avere un sistema che rispetta la safety (*Es.* un treno che rimane fermo in stazione senza persone dentro) ma il sistema deve essere anche available altrimenti il sistema risulta sostanzialmente inutile. Vorrei il 100% sia in Availability che in safety ma nella realtà questo è impossibile.
Es. metto dei controlli su un treno sia all'interno che all'esterno per fare sì che in caso di problemi il treno si fermi in sicurezza sulla linea. Quando il treno si ferma l'availability diminuisce quindi **NON è al 100%** però cerco di raggiungere invece il 100% di safety.

Riassunto:
un sistema che vuoi che non abbia failure catastrofici è un safety critical system. In questo caso parliamo di un sistema generico, di solito composto anche da più moduli hardware e software.

Nota: nelle slides c'è la parte sulla validation ma la prossima lezione andrà sulla metrology. Metterà le slides sempre prima della lezione così da seguirle direttamente e prenderci appunti sopra.