CEFRIEL
FORGING INNOVATION POLITECNICO DI MILANO

Software: modelli di sviluppo nelle applicazioni per gli apparecchi elettrodomestici

Prof. William Fornaciari

Politecnico di Milano - Dip. di Elettronica e Informazione

william.fornaciari@polimi.it

Quando l'elettronica entra nell' elettrodomestico

IMQ-Milano, 18 Maggio, 2007

Sistemi Embedded: il caso elettrodomestico

- Sistemi embedded
  - Dedicati a specifiche applicazioni, programmazione o riprogrammazione impossibile o poco frequente, non da utente finale in genere
  - Ottimizzazione di molte esigenze, es. costo, prestazioni, dimensioni, potenza, time to market, limitatezza delle risorse, affidabilità, sicurezza, manuteniblità, …
  - Pervasive, ubiquitous, invisible…sono ormai consumer, inclusi nei white goods
- Progettazione complessa
  - Risorse limitate, sw e hw interagente: come integrare?
  - Serve un sistema operativo? Come lo seleziono?
  - Time-to-market: primo progetto spesso “disordinato” e a basso costo, metodologia poco strutturata con difficoltà di riuso
  - Uso di outsourcing: problema di controllo del know-how e della qualità del prodotto. Si deve gestire evoluzione di famiglie di prodotti
  - Off the shelf vs custom: si accettano rischi? Quali tecnologie?
  - Gli attuali studenti conoscono tecnologie sw troppo di fascia alta, ma il mercato è differente…es. la lavatrice non si programma in Java (per ora!)

Sw embedded: caso elettrodomestico

- Obiettivi
  - Piccole memorie e potenze di calcolo
  - Riparabilita/aggiornabilità limitata
  - Hard e soft real time
  - Interfacce utente semplificate
  - Eredità storica
  - Costi
  - ancora costi!
- Vincoli
  - Affidabilità
  - Sicurezza
  - Evoluzione nel tempo
  - Specializzazione su famiglie di prodotti simili
  - Disaccoppiamento fra interfaccia utente e controllo
  - Non basare il successo solo su talento di un progettista

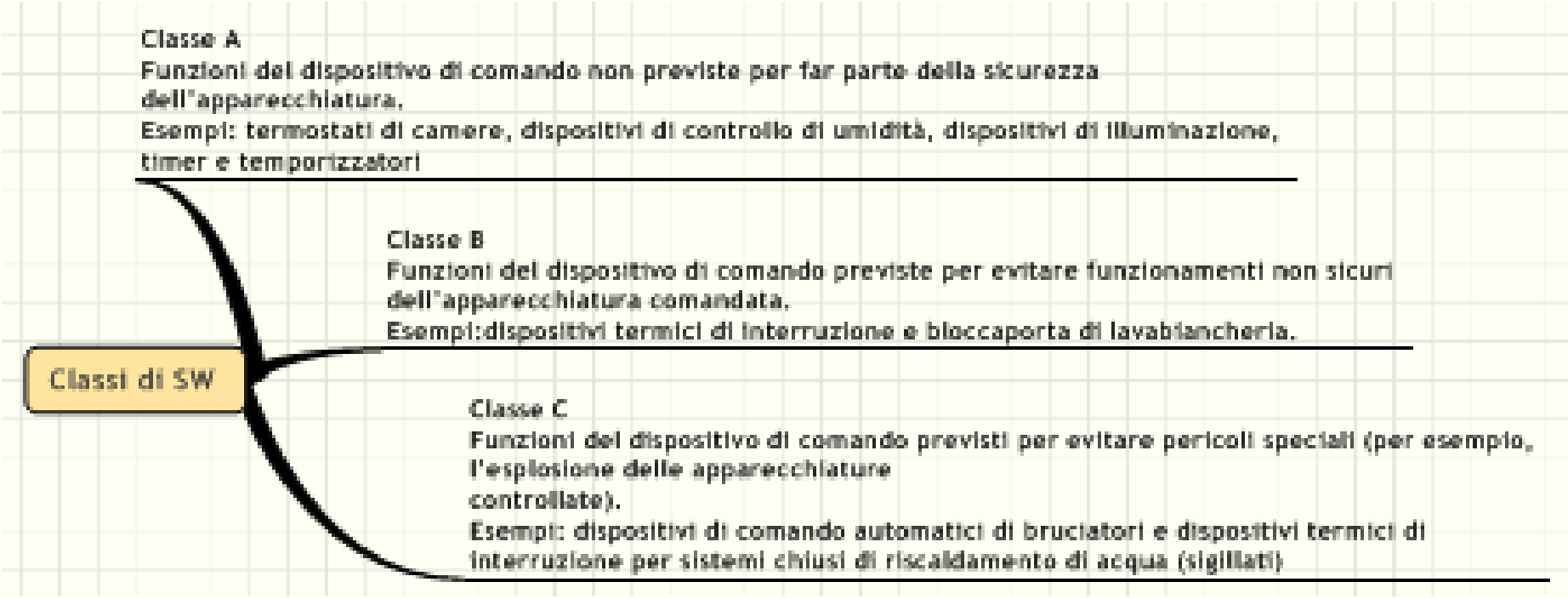
Spinge il mercato o la normativa?

Esempio: Dispositivi elettrici automatici di comando per uso domestico e similare

- **Prescrizioni generali**
  - Il costruttore deve fornire adeguate informazioni per confermare:
  - che venga scelto un dispositivo di comando adatto;
  - che il dispositivo di comando può essere montato ed usato in modo tale che esso possa soddisfare le prescrizioni della presente Norma;
  - e che per assicurare la conformità con la presente Norma, possano essere effettuatele prove appropriate.

- **Metodi per fornire le informazioni**
  - Le informazioni devono essere fornite usando uno o più metodi che seguono:
  - Con Marcatura (C): fornite tramite marcatura sullo stesso dispositivo di comando, eccetto nel caso di un dispositivo di comando integrato, in cui la marcatura può essere su una parte adiacente dell'apparecchiatura, purché sia chiaro che essa si riferisce al dispositivo di comando
  - **Con Documentazione (D): fornite all'utilizzatore o all'installatore del dispositivo di comando, e devono consistere in istruzioni leggibili.**
  - **Con Dichiarazione (X): fornite al responsabile delle prove ai fini delle prove e nel modo concordato tra questo ed il costruttore.**

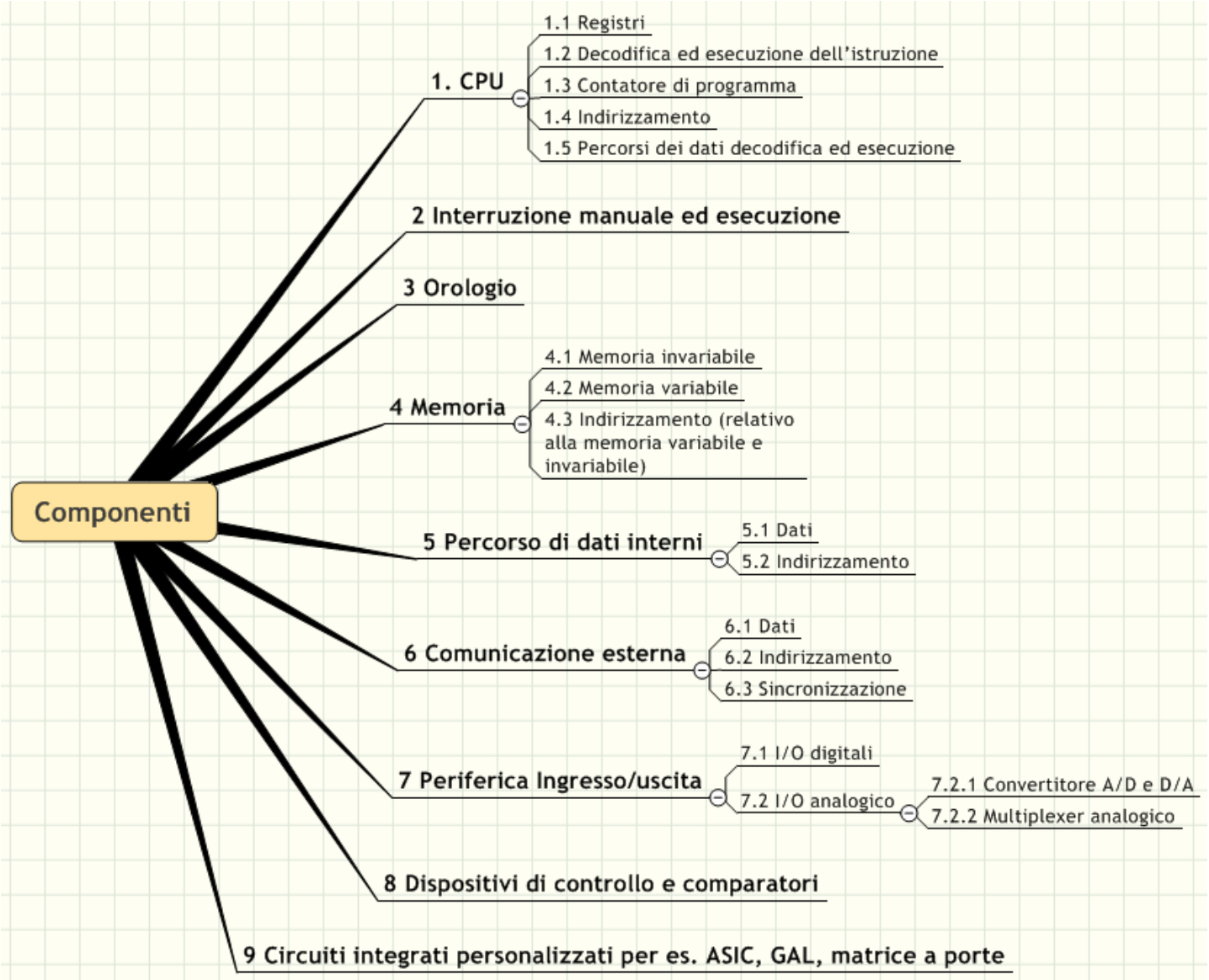
Classi di software



Doc del software: esempio di requisiti …..c'è molto da fare!

- **Documentazione della sequenza del SW**
- **Documentazione del programma**
- Per i dispositivi di comando con funzione dichiarata come software di Classe B o C, le informazioni devono essere fornite solo per i segmenti di software relativi alla sicurezza. Le informazioni sui segmenti che non sono relativi alla sicurezza devono essere sufficienti per stabilire che essi non influenzano i segmenti relativi alla sicurezza.
- La sequenza del software deve essere documentata e, insieme alla sequenza di funzionamento della prescrizione 46 che deve comprendere la descrizione della filosofia del sistema di comando, della portata del dispositivo, del flusso dei dati e delle sincronizzazioni.
- La documentazione della programmazione deve essere fornita nel linguaggio di programmazione dichiarato dal costruttore.
- I dati ed i segmenti di sicurezza della sequenza del software, le cui disfunzioni potrebbero causare una non conformità alle prescrizioni 17, 25, 26 e 27, devono essere identificati. Questa identificazione deve comprendere la sequenza di funzionamento e può, per esempio, prendere la forma di un'analisi d'albero di guasto che deve comprendere i guasti/errori della Tab. H.11.12.7 derivabili da questa non conformità. L'analisi del guasto del software deve essere riferita all'analisi del guasto dell'hardware H.27.
- Esempi di altre informazioni che si possono inserire nella documentazione per sistemi software:
  - Specifiche funzionali comprendenti una procedura di ripristino a seguito della mancanza di alimentazione.
  - Studio del modulo di programma comprendente una descrizione delle interfacce dell'apparecchiatura e delle interfacce verso gli utilizzatori.
  - Studio dettagliato, compresa la descrizione dell'utilizzo della memoria.
  - Elenco dei codici comprendente l'identificazione del linguaggio di programmazione, i commenti e l'elenco delle subroutine.
  - Specifiche di prova.
  - Manuale di installazione, di utilizzazione e/o di manutenzione

Esempio di analisi dei guasti



Il software (indipendentemente dalla motivazione)

- Codice sicuro
  - il primo passo è la qualità e la comprensibilità
- Sviluppo della documentazione
  - il software per applicazioni embedded non è meno complesso rispetto agli altri…anzi
- Qualità del codice
  - Stile e testing
  - Utilizzo di tool e flussi di sviluppo standardizzati

Il codice sorgente (1)

- Il codice serve a **comunicare**
  - con la macchina
  - con gli altri sviluppatori
  - con i collaudatori
  - con i certificatori (?)
- Il codice scritto deve poter essere letto da altri
  - È letto da persone più spesso che da macchine
- Il codice può avere anche una vita lunga
  - Più il codice è utile e più sarà letto per più tempo e da più persone
- Il significato e lo scopo del codice scritto può non essere facilmente comprensibile
  - lo sviluppatore è responsabile anche della comprensibilità del codice

Il codice sorgente (2)

- Il codice che scriviamo deve essere manutenibile

- Deve essere relativamente facile modificare il codice per cambiarne il comportamento
  - I problemi per cui scriviamo codice, o l'ambiente in cui opera, evolvono nel tempo, quindi il codice va adattato
  - L'estensibilità di una applicazione è anche una proprietà interna
  - I difetti vanno corretti

- I programmi di maggior successo sono usati in più ambiti diversi e sono quindi anche quelli che richiedono più manutenzione

Il codice sorgente (3)

- Il codice deve essere affidabile
  - l'errore è inevitabile
  - fiducia nel codice scritto da altri
  - fiducia che codice ben scritto sia indice di codice ben pensato

- Prestazioni
  - Codice veloce ma incomprensibile può essere migliorato con molta fatica
  - Codice comprensibile può essere ottimizzato

- Non sprecare risorse, compreso il tempo dei programmatori
  - Se gli sviluppatori hanno a che fare con codice ben scritto, risparmiano tempo

Qualità del software (1)

- Cos'è la qualità del software
- La certificazione del software è più semplice se la sua qualità è maggiore (??)
- Gli aspetti di qualità del codice da considerare sono:
  - La robustezza o affidabilità
  - La leggibilità
  - La correttezza
- Robustezza -> processo di sviluppo
- Leggibilità -> stile di programmazione
- Correttezza -> testing

La robustezza software (1)

- L'affidabilità è uno degli aspetti del software che deve essere tenuto in considerazione quando se ne determina la qualità

- Nonostante il termine "qualità" faccia riferimento ad una valutazione di tipo soggettivo, l'affidabilità del software può essere "misurata" attraverso criteri oggettivi, detti metriche

- A causa della crescente pervasività che il software embedded ha raggiunto nei dispositivi odierni, i difetti presenti nel software causano inconvenienti sempre più rilevanti

La robustezza software (2)

- La necessità di misurare oggettivamente la qualità del software deriva dalla necessità di applicare le tecniche dei settori ingegneristici tradizionali, come il riuso, allo sviluppo del software
- L'esigenza nasce sia da un punto di vista tecnico sia da un punto di vista legislativo
  - Il software non deve esibire un comportamento indesiderabile e non previsto
  - I difetti software possono provocare danni ai dati memorizzati, al sistema sul quale il software è eseguito o anche alle persone, nel caso di sistemi embedded presenti nei mezzi di trasporto e negli apparati medicali
- A parte la criticità della singola applicazione software, la pervasività del software sta aumentando e, se il tasso di crescita sarà confermato, si arriverà ad un punto in cui il software sarà l'elemento cruciale della società di domani
  - Se la società di domani si poggerà sul software, la sua qualità dovrà essere all'altezza delle aspettative

L'affidabilità nel processo di sviluppo (1)

- Requisiti
  - Il programmatore per sviluppare correttamente un'applicazione deve conoscerne il comportamento atteso, almeno in parallelo alla fase di sviluppo e con un livello di dettaglio adeguato
    - Raggiungere il livello di dettaglio adeguato può non essere tecnicamente o economicamente praticabile
    - Raffinare troppo i requisiti comporta uno spreco di tempo e risorse
    - Non raffinare i requisiti ad un livello adeguato porta ad avere un'applicazione che non sa come reagire alle situazioni non previste
- Design
  - Il design specifica come un programma debba essere fatto, almeno ad alto livello
  - Il design di alto livello permette di separare i problemi che si incontrano nella definizione dell'architettura, come la struttura di un programma e i concetti, dai problemi di codifica, che sono mirati all'elaborazione delle informazioni

L'affidabilità nel processo di sviluppo (2)

- Linguaggi di programmazione
  - L'evoluzione dei linguaggi di programmazione cerca di fornire strumenti che consentano di delegare sempre più lavoro ai calcolatori, riducendo la possibilità di introdurre difetti da parte dello sviluppatore
    - L'introduzione di macchine virtuali consente di rilevare difetti software a run-time
  - In altre parole, l'evoluzione dei linguaggi di programmazione deve consentire di gestire programmi sempre più complessi e di dimensioni maggiori
- Testing
  - Dato un programma o una sua parte, è possibile eseguire dei test, manualmente o in automatico, per determinare se il software si comporta come dettato dai requisiti
    - L'esecuzione automatica di suite di test permette di verificare facilmente se le modifiche introdotte in un programma ne hanno modificato la correttezza

Qualità del codice

- Quando possiamo dire che il codice è di qualità?
- Com'è possibile "misurare" la qualità?
- Esistono metriche:
  - verificabili automaticamente
  - verificabili attraverso ispezione
- Oppure:
  - Metriche di processo, che si calcolano introducendo "sensori" nel processo di sviluppo
  - Metriche di codice, calcolate direttamente sul codice sorgente
- In generale, il codice è di qualità quando presenta delle caratteristiche e una struttura che
  - non promuovano
  - non nascondano
  - aiutino ad evitare

  l'introduzione di difetti nel codice, sia quando il codice viene rilasciato, sia quando deve essere modificato/in manutenzione

Metriche di processo

- Le metriche di processo sono le più oggettive, in quanto misurano la qualità partendo dagli effetti che essa genera
- Misura della correttezza
  - Numero di difetti riscontrati dopo il rilascio del software
- Misura della manutenibilità
  - Tempo medio impiegato per correggere un difetto rilevato o per introdurre una nuova funzionalità
- Queste grandezze sono difficili da misurare automaticamente
  - Richiedono una modifica al processo di sviluppo al fine di tracciare le grandezze di interesse
  - Richiedono che gli sviluppatori collaborino alla misura delle grandezze

Metriche di codice

- Quando le metriche di processo non si possono utilizzare, ci si affida alle metriche di codice
  - Queste metriche sono meno oggettive di quelle di processo, in quanto è possibile influenzare il loro valore senza
- Il codice è documentato?
  - Una metrica può essere: numero di linee di commento / numero di linee totali
  - Questa metrica può non essere affidabile
    - lo sviluppatore può influenzare la misura

```
/* La funzione calcola il minimo comun denominatore */
void mcd(int * a);
```

```
/* La funzione calcola
il minimo comun denominatore */
void mcd(int * a);
```

Migliorare la robustezza

- Le metriche possono fornire un'indicazione quantitativa della robustezza, ma non la migliorano
- La robustezza può essere migliorate attraverso
  - Analisi statica del codice
  - Ispezione del codice
- L'analisi statica del codice permette di rilevare possibili "situazioni pericolose" che a run-time potrebbero diventare difetti software
  - Ad esempio, i warning generati dal compilatore rappresentano la forma più rudimentale di analisi statica
- Esistono insiemi di regole di scrittura del codice che mirano a produrre codice più robusto
  - Ad esempio, esistono le regole MISRA che favoriscono robustezza e portabilità
  - Esistono strumenti commerciali che effettuano la verifica automatica delle regole MISRA
  - Le regole MISRA sono numerose: se ne consiglia un'adozione incrementale

Esempio di regole MISRA

La manutenzione del software può introdurre errori

```
int foo( int a )
{
      if( a > 10 )
            return 0;
      else
            return a++;
}
```

```
int foo( int a )
{
      if( a > 10 )
            return 0;
      else
            a = a + 2;
            return a;
}
```

Esempio di regole MISRA

Questa regola obbliga a introdurre le parentesi graffe anche nei blocchi di codice composti da una sola istruzione

Questa regola evita l'introduzione di possibili errori durante la manutenzione

```
int foo( int a )
{
        if( a > 10 )
        {
                return 0;
        }
        else
        {
                return a++;
        }
}
```

```
int foo( int a )
{
        if( a > 10 )
        {
                return 0;
        }
        else
        {
                a = a + 2;
                return a;
        }
}
```

L'ispezione

- L'ispezione del codice consiste in incontri periodici tra sviluppatori che "guardano" alcuni sezioni software
  - È prevista una check-list di controlli che gli sviluppatori devono effettuare sul codice da ispezionare
- L'ispezione non si può applicare a tutto il software, in quanto eccessivamente time-consuming
- Si suggerisce di prevedere sedute di ispezione del codice sui moduli più complessi o quelli più riusati o quelli più recenti
  - Permette di rilevare difetti, errori o inconsistenze
  - Consente di uniformare lo stile di formattazione tra gli sviluppatori
  - Aiuta il team di sviluppo ad abituarsi alle regole MISRA che i responsabili di progetto hanno di deciso di rispettare

Miglioramento della leggibilità (1)

- Un team di sviluppo, o anche tutta l'organizzazione, deve avere regole precise circa lo stile di scrittura del codice
- Le regole di stile nella scrittura del codice sono detti "pattern stilistici"
  - Sono pratiche riguardanti la scrittura pura del codice, non la progettazione
  - Spesso, sono supportati dall'IDE
  - Spesso, sono verificabili
- Tutto il codice è scritto seguendo le stesse regole, quindi, per chi legge, è più semplice orientarsi
- I pattern stilistici devono essere condivisi e adottati da tutti i componenti del team di sviluppo

Miglioramento della leggibilità (2)

- Inizialmente, potrebbe essere utile affiancare ai pattern stilistici, uno strumento di formattazione automatica
  - Tali strumenti consentono “forzare” uno stile di programmazione e possono sopperire a eventuali “sviste” o vecchie abitudini dei programmatori
  - Possibili strumenti liberamente distribuiti
    - AStyle, semplice da usare, ma poco flessibile
    - indent, molto flessibile e configurabile, ma un po’ più complesso

```
int foo(int a){
        if(a>10)
                return 0;
        else
                return a++;
}
```

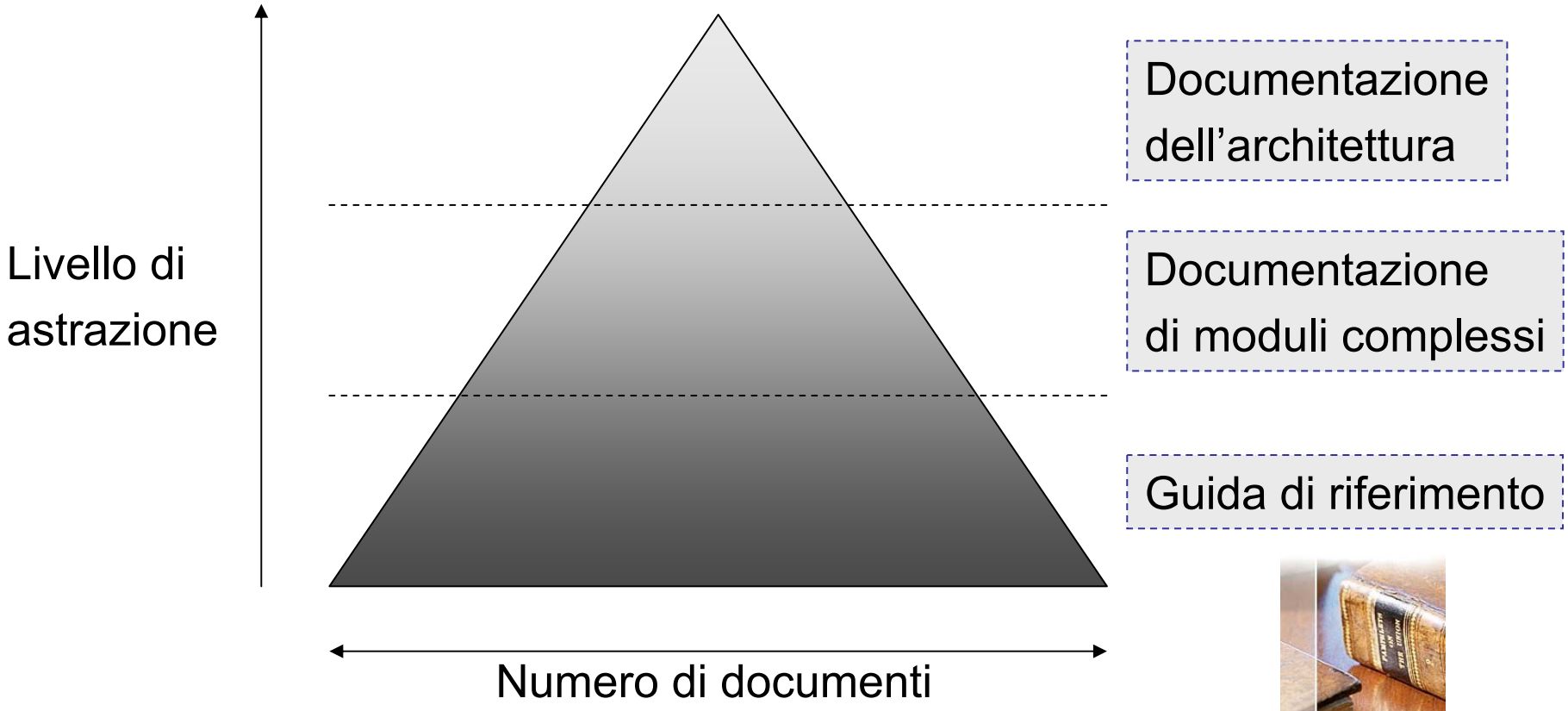
```
int foo( int a )
{
        if( a > 10 )
                return 0;
        else
                return a++;
}
```

Documentazione del software

- Il processo di certificazione del software si appoggia sulla documentazione
- La documentazione si può classificare in:
  - Documentazione di progetto
  - Guida di riferimento
  - Documentazione dei casi di test

Migliorare la documentazione

- La documentazione dovrebbe coprire principalmente tre aspetti:



Documentare l'architettura di sistema

- Descrive il progetto nel suo insieme
- Può include anche la descrizione esaustiva dei moduli p semplici
- Eventuali informazioni di dettaglio trasversali ai moduli possono essere "confinate" in appendici
- La frequenza di aggiornamento di questo documento deve essere bassa
  - Se, per esigenze di coerenza, il documento subisce continui e frequenti aggiornamenti, significa che:
    - Include informazioni troppo dettagliate
    - L'architettura è ancora in forte evoluzione

Miglioramento della documentazione
Documenti di maggiore dettaglio

- Al fine di non appesantire la trattazione dell'architettura, i moduli più complessi, o gruppi di moduli correlati o interi sotto-sistemi possono essere illustrati in documenti a parte
  - Ad esempio, il protocollo di comunicazione potrebbe essere documentato in un documento a parte
- Si considera opportuno non prevedere un documento per ogni modulo, al fine di evitare il proliferare di documenti con basso contenuto informativo
- La frequenza di aggiornamento di questi documenti può essere più elevata di quella della documentazione dell'architettura, anche se non deve corrispondere a quella di modifica del software

Miglioramento della documentazione
La guida di riferimento

- La guida di riferimento è un documento che descrive dettagliatamente tutti i tipi di dato e le funzioni presenti in tutta la piattaforma
- Solitamente, questo tipo di documentazione ha una velocità di obsolescenza molto rapida, soprattutto durante le fasi di sviluppo del codice
- Nasce, quindi, l'esigenza di utilizzare strumenti opportuni che permettano di generare in modo automatico la guida di riferimento
- Utilizzo di strumenti di generazione automatica di documentazione partendo dai commenti
  - Doxygen richiede semplicemente una formattazione dei commenti
  - Molti moduli dispongono già di commenti significativi
    - È sufficiente aggiungere i tag per Doxygen e la guida di riferimento è già pronta
  - Doxygen può generare una documentazione in html navigabile da browser
  - Doxygen può generare documenti in rtf o in latex, per produrre manualistica
  - È possibile far generare anche i grafi delle inclusioni, i grafi delle chiamate di funzione e il class diagram

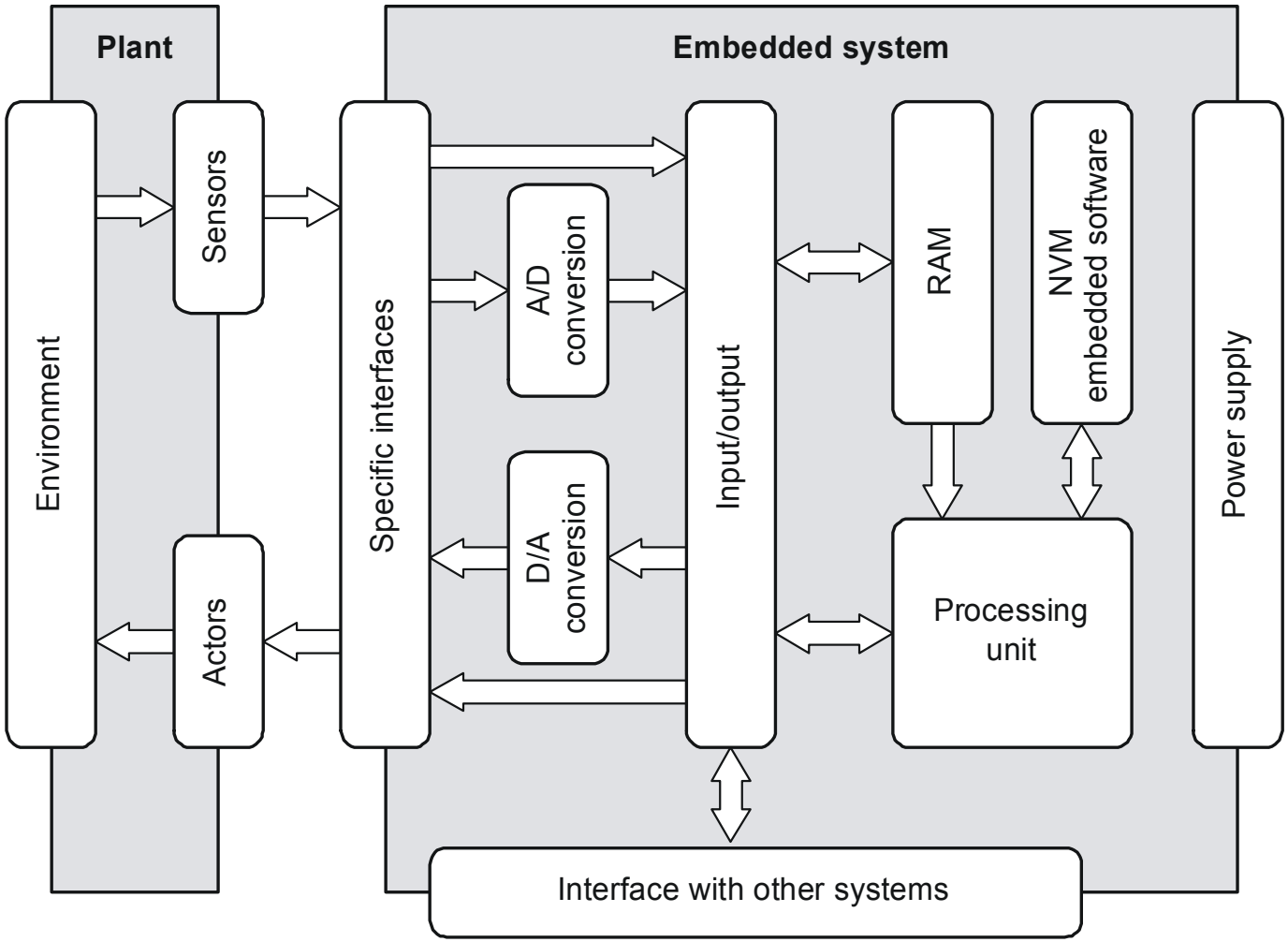
CEFRIEL
FORGING INNOVATION POLITECNICO DI MILANO

Ben fatto!…..Ma funziona?

Cosa significa fare testing

Come fare il testing?

- Attenzione, non si testa solo un programma
  - spesso è l'intero sistema embedded che viene verificato

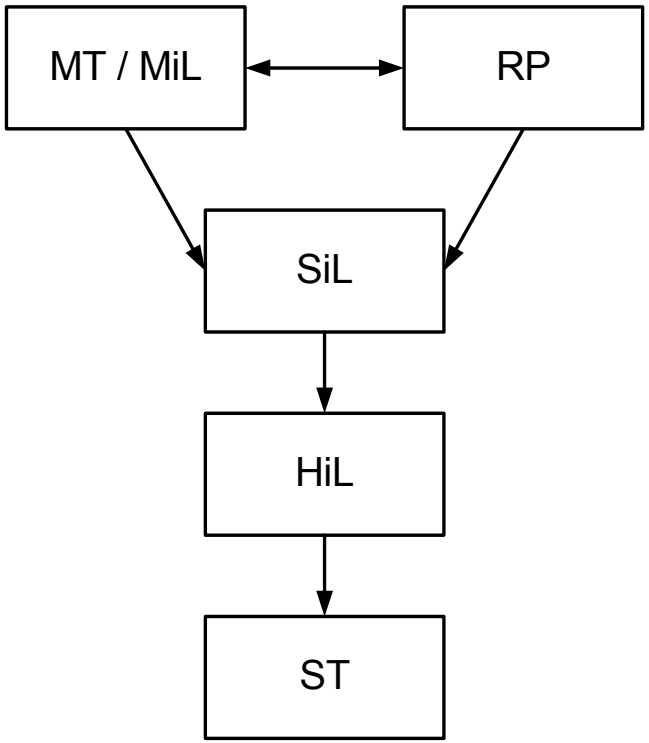


Fasi nel testing di sistemi Embedded

- Simulazione
  - tutti gli elementi del sistema sono simulati
- Prototipazione
  - alcuni elementi del sistema sono simulati
  - alcuni elementi del sistema sono prototipali (reali, ma non pensati per la produzione, solo per la prototipazione)
  - alcuni elementi del sistema sono reali
- Pre produzione
  - tutti gli elementi del sistema sono reali
- Post produzione
  - tutti gli elementi del sistema sono reali

Test: transizione dal sistema simulato al sistema reale

- MT: Model Test
  - (simulazione)
- MiL: Model in the Loop
  - (simulazione)
- RP: Rapid Prototyping
  - modello “irreale” (con molte risorse), ambiente reale
  - (simulazione)
- SiL: Software in the Loop
  - sw reale testato su hw simulato
  - (prototipazione)
- HiL: hardware in the Loop
  - hw reale (uno o più componenti) testato in un ambiente simulato
  - (prototipazione)
- ST: System Test
  - hw reale e sw reale
  - (pre produzione)



Come procedere

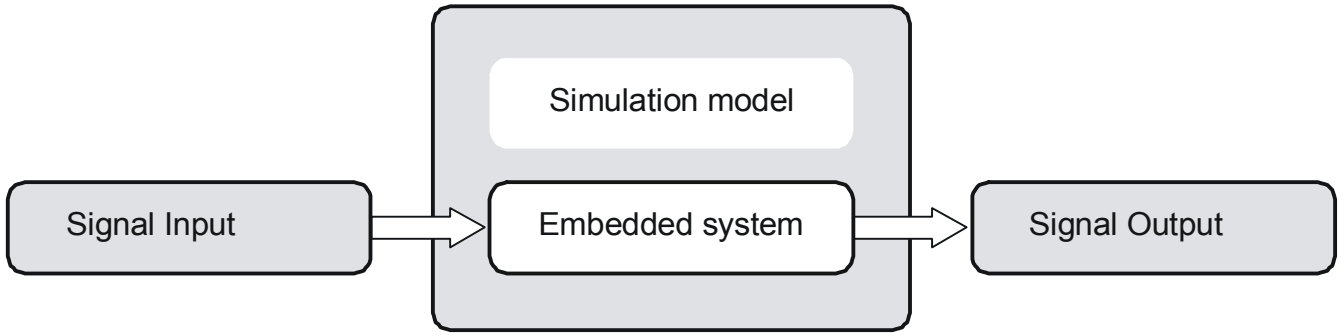
- Simulazione
  - One-way
  - Feedback
  - Prototipazione rapida
- Prototipazione
  - Componenti simulabili
  - Tipologia di test
- Pre produzione, Post produzione

Simulazione

- aka: Model Testing, Model in the Loop
- Obiettivi
  - Progettazione
  - Verifica concettuale
- Non è una fase “obbligatoria”
- Tipologie
  - Simulazione one-way (S1)
  - Simulazione con feedback (SF)
  - Prototipazione rapida (RP)
- Best practice:
  - **Prima** costruire il modello dell’Ambiente
  - **Poi** il modello del Sistema Embedded

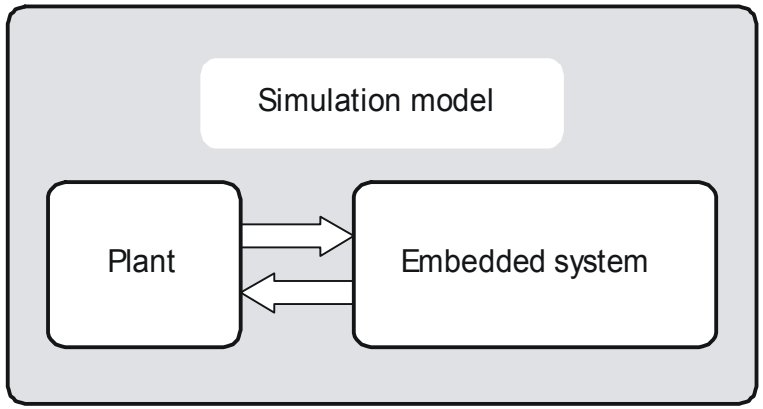
Simulazione: one-way

- Assunzioni
  - Il comportamento dinamico dell'ambiente viene ignorato.
  - Vengono generati segnali di Input, raccolti e analizzati segnali di Output.
- Implementazione
  - Simulazione del sistema embedded attraverso pc.
  - Input e Output
    - canali di comunicazione (simulati) del sistema simulato.
    - controllo e accesso diretto registri e variabili del sistema simulato.
- Automazione
  - Strumento di confronto fra l'Output atteso e l'Output reale.



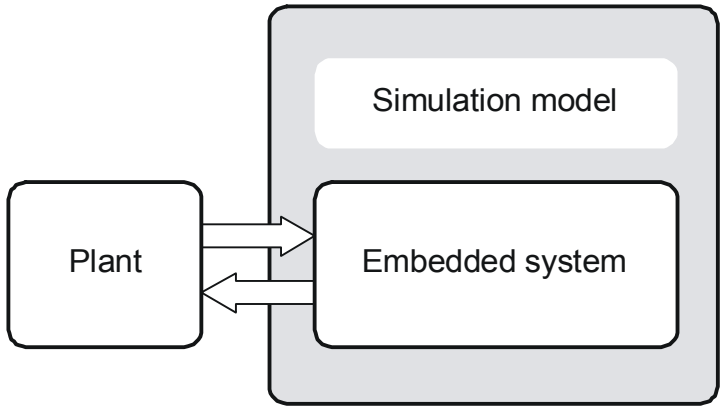
Simulazione: feedback

- Assunzioni
  - Il comportamento dinamico dell'ambiente viene simulato
  - Il modello di simulazione comprende sia l'ambiente che il sistema embedded
  - La simulazione di tipo feedback è applicabile solo nei casi in cui sistema e ambiente possono essere semplificati senza comprometterne l'accuratezza



Simulazione: prototipazione rapida

- Assunzioni
  - Il comportamento dinamico dell'ambiente non viene simulato, ma ottenuto dall'ambiente reale (o una sua copia sufficientemente equivalente).
  - Il modello di simulazione non comprende l'ambiente, ma il solo sistema embedded



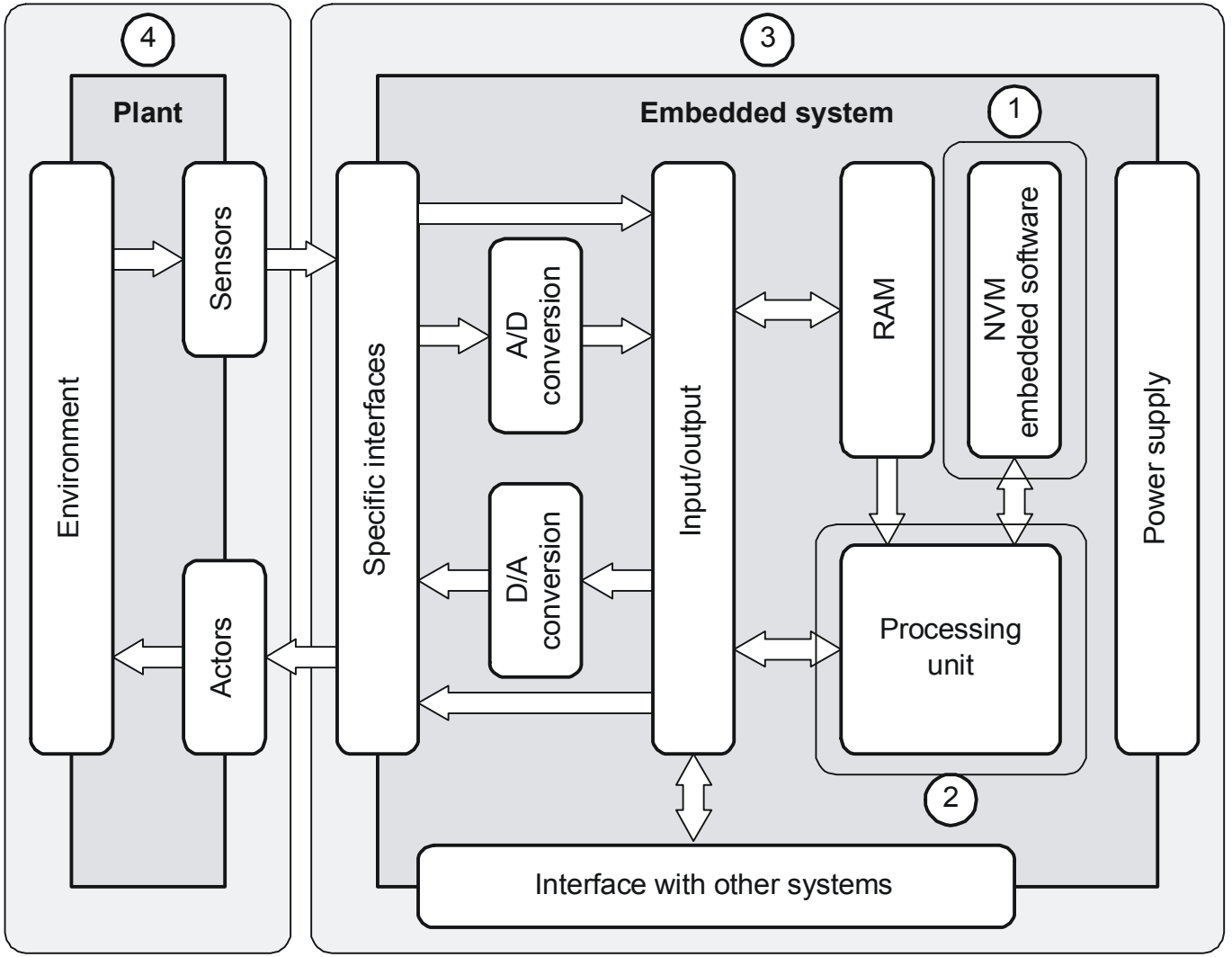
Simulazione

		SW	*Processor*	*Embedded system*	*Ambiente - Plant*
S1	MT	simulato	-	-	-
SF	MiL	simulato	-	-	simulato
RP	RP	sperimentale	sperimentale	sperimentale	reale

Prototipazione

- aka: Model Testing, Model in the Loop
- Obiettivi
  - Verifica del modello di simulazione (se presente)
  - Verifica che il sistema soddisfi i requisiti
  - Rilascio dell'unità di pre produzione
- Processo
  - Nella fase di prototipazione hw e sw reali gradualmente sostituiscono i componenti simulati
  - Molte iterazioni
- Osservazioni
  - Necessità di interfacce fra hw e sw (simulati e non)
  - Alcuni segnali disponibili nei modelli simulato possono non essere disponibili nei modelli reali.
    - Strumentazione: raccoglitori di segnali, trasduttori, strumentazione registrazione di segnali...
    - Calibrazione strumentazione: automatica, ripetibile, documentata

Componenti simulabili: schema



Componenti simulabili: descrizione

1. Embedded Software

   Il sw può essere simulato:
   - su una piattaforma host, compilato per la piattaforma host
   - su un emulatore della piattaforma target eseguito dal computer host, compilato per la piattaforma target
2. Processore

   Il processore può essere sostituito da un processore di potenza maggiore.

   Il processore di potenza maggiore può emulare il processore target.
3. Restante sistema embedded

   Il restante sistema embedded può essere simulato attraverso:
   - emulatore sulla piattaforma host
   - costruzione di una configurazione hw sperimentale
   - costruzione di una piastra prototipale corredata di ogni suo componente
4. Ambiente
   - Simulazione statica: generazione segnali
   - Simulazione dinamica: interfacciamento con una piattaforma di simulazione (computer)

Prototipazione: tipologie di test

- Test di unità software (SW/U)
  - Test dei singoli componenti sw
- Test di integrazione software (SW/I)
  - Test delle interazioni fra componenti sw
- Test di integrazione hardware/software (HW/SW/I)
  - Test delle interazioni fra componenti sw e hw
- Test di integrazione di sistema (SI)
  - Verifica che il sistema si comporti come specificato
- Test ambientale (E)
  - Test del sistema in specifiche condizioni ambientali

Test di unità software (SW/U)
Test di integrazione software (SW/I)

- Sistema di test
  - i moduli software sono l'oggetto sotto test
  - il resto della piattaforma è simulato
- Tipologie di test
  1. moduli sw compilati per la piattaforma host
     esecuzione sulla macchina host
     nessun vincolo sulle risorse
  2. moduli sw compilati per la piattaforma target
     esecuzione sulla macchina host tramite emulatore della piattaforma target
- Ambiente di test
  - stimola i moduli sw sotto test
  - registra/monitora/controlla i segnali in uscita

Test di integrazione hardware/software (HW/SW/I)

- Sistema di test
  - parte della piattaforma hw è l'oggetto sotto test (sistema embedded sperimentale)
  - moduli sw compilati per la piattaforma target
  - processore reale
  - ambiente simulato
- Ambiente di test
  - l'ambiente di test deve interfacciarsi con l'hw non simulato
    - segnali di ingresso attraverso generatori di segnale
    - segnali di uscita monitorati con oscilloscopi
    - elementi aggiuntivi su piastra per monitorare il comportamento del sistema non visibile attraverso i segnali di uscita
    - simulazione dell'ambiente pilotata (segnali di ingresso generati dinamicamente e segnali di uscita verificati)

Test di integrazione di sistema (SI)

- Sistema di test
  - la piattaforma hw (completa) è l'oggetto sotto test (sistema embedded prototipale)
  - moduli sw compilati per la piattaforma target
  - processore reale
  - ambiente simulato
- Ambiente di test
  - l'ambiente di test deve interfacciarsi con l'hw non simulato
    - segnali di ingresso attraverso generatori di segnale
    - segnali di uscita monitorati con oscilloscopi
    - elementi aggiuntivi su piastra per monitorare il comportamento del sistema non visibile attraverso i segnali di uscita
    - simulazione dell'ambiente pilotata (segnali di ingresso generati dinamicamente e segnali di uscita verificati)

Test ambientale (E)

- Sistema di test
  - la piattaforma hw (completa, sufficientemente matura) è l'oggetto sotto test (sistema embedded prototipale)
  - moduli sw compilati per la piattaforma target
  - processore reale
  - ambiente simulato
- Obiettivo
  - identificare problemi con l'ambiente prima di arrivare alla fase di pre produzione
    - quanto il sistema embedded influenzi l'ambiente (measuring)
    - quanto il sistema embedded sia influenzato dall'ambiente (generating)

Test prototipale

		SW	*Processor*	*Embedded system*	*Ambiente - Plant*
SW/U SW/I 1	SiL	sperimentale / reale (host)	host	simulato	simulato
SW/U SW/I 2	SiL	reale (target)	emulatore (target su host)	simulato	simulato
HW/SW/I	HiL	reale (target)	reale (target)	sperimentale	simulato
SI	HiL	reale (target)	reale (target)	prototipale	simulato
E	HiL / ST	reale (target)	reale (target)	prototipale, maturo	simulato

Pre produzione

- Obiettivi:
  - dimostrare che i requisiti sono soddisfatti
  - dimostrare la conformità a standard (imposti, qualità, legge...)
  - dimostrare la costruzione del sistema nell'ambiente di sviluppo con effort e tempi previsti
  - dimostrare l'affidabilità del sistema nell'ambiente di uso (mtr: mean time to repair, ...)
  - presentare il prodotto agli utenti finali
- Scenari di test predeterminati.
- Test: accettazione, sistema, sicurezza, qualifica rispetto astandard, ispezioni...

Post produzione

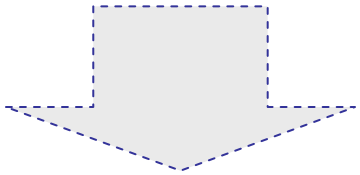
- Sviluppo e test della catena di produzione
  - ispezione della qualità del prodotto
  - verifica tempo e costi di produzione
- Ispezione del primo prodotto
  - ispezione e test del primo manufatto prodotto dalla catena
- Test di produzione e mantenimento
  - test di ispezione della qualità
    - sommari su ogni manufatto prodotto
    - approfonditi su campioni casuali scelti fra i manufatti prodotti

A che punto siamo in azienda…..?

Esempio di assessment
R&D e sviluppo nuovi prodotti (NPD)

Assessment R&D (1)

- Le prestazioni delle attività di R&D e NPD (new product development) possono determinare non solo il successo e il vantaggio competitivo di un'azienda, ma anche la sua stessa sopravvivenza
- Il mercato odierno richiede ai costruttori di elettrodomestici di competere con prodotti basati su tecnologie digitali sempre più innovative e di elevata qualità



- Una stima periodica delle performance e una valutazione delle attività di R&D e NPD dovrebbero essere punti cardine per ogni azienda industriale innovativa
  - identificazione e valutazione di parametri che evidenzino il livello qualitativo della ricerca, progettazione e produzione
- In particolare, per i produttori di elettrodomestici questo diventa sempre più vero per la componente elettronica (HW e SW) dei loro prodotti

Assessment R&D (2)

- Il progetto di assessment di progetti elettronici si basa sull'utilizzo di un insieme di *key indicators* che coprono tutte le fasi fondamentali della progettazione e dello sviluppo di un sistema elettronico
  - la quantità e il tipo dei *key indicators* dipende di volta in volta dallo specifico scenario R&D e NPD sotto analisi
- Per ogni *key indicator* viene data una valutazione:
  - Dettagliata e descrittiva, contenente anche, dove necessario, indicazioni di azioni migliorative/correttive
  - Sintetica
    - Ad esempio un giudizio tra 1 e 5
      - 1) below average; 2) average; 3) above average; 4) very good; 5) outstanding.
  - Sul trend, per tener conto dell'evoluzione percepita dell'indicatore nel tempo
    - Ad esempio: high worsening; worsening; stable; improving; high improving
- Per facilitare analisi, valutazioni e visualizzazioni al top management, gli indicatori puntuali possono essere raggruppati in "macro" indicatori
- Alla fine del progetto di assessment, sono evidenti i punti di forza e quelli di debolezza del processo di progettazione e sviluppo e dei corrispondenti prodotti finali

Esempio di key indicators per un progetto con componenti HW e SW

Indicatori progetto HW

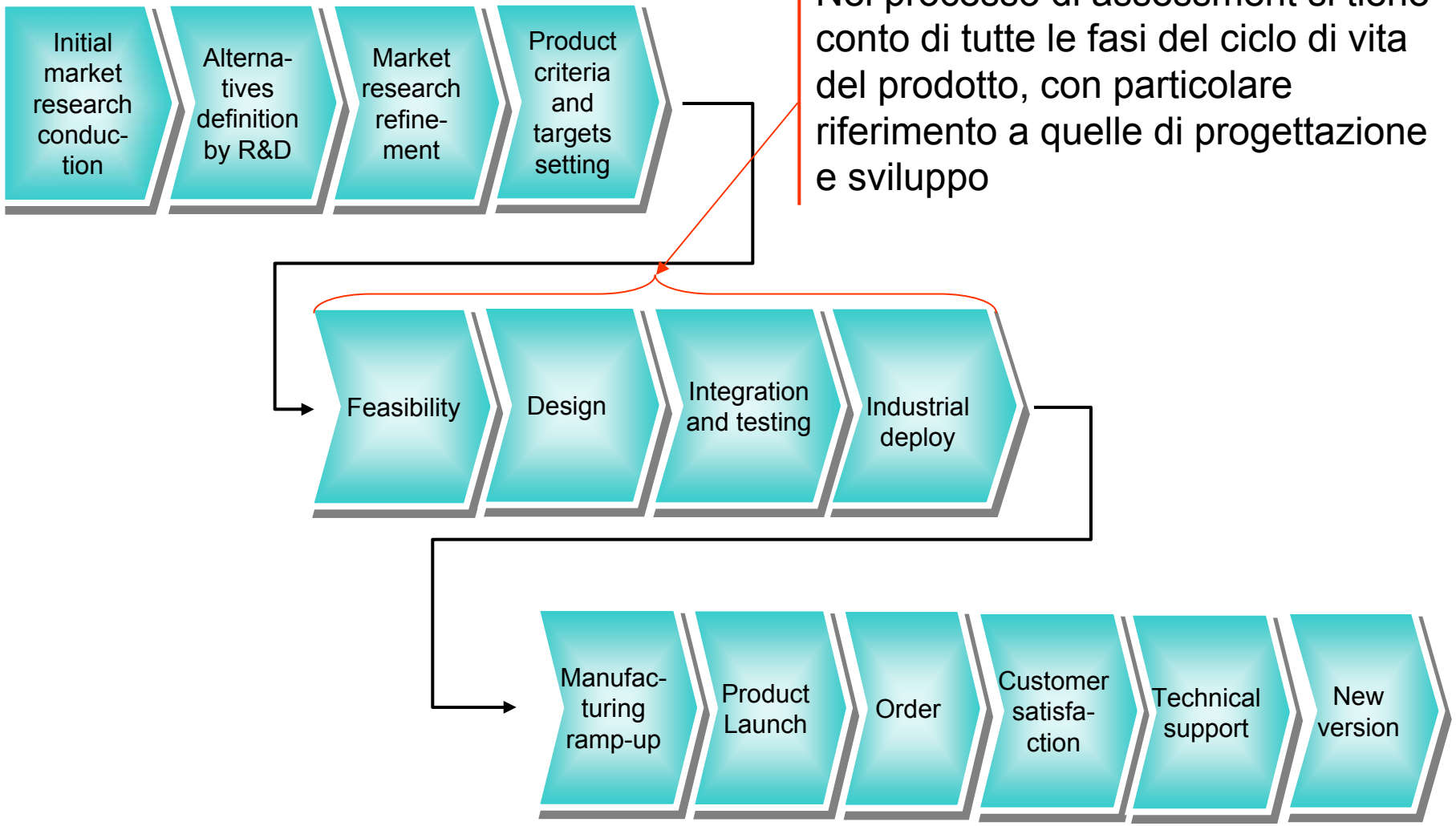
- Analisi dei requisiti
- Configuration Management
- Documentazione dei requisiti
- Architettura del sistema elettronico
- Soluzioni circuitali
- Scelta e dimensionamento componenti
- Layout delle schede
- Livello di sicurezza
- Pianificazione delle attività
- Riusabilità
- Flusso di validazione
- Gestione dei fornitori
- Documentazione di sistema
- Specifiche tecniche
- Documentazione tecnica di dettaglio
- Schemi elettrici ed elettronici
- Strumenti di sviluppo
- Strumentazione di misura
- Strumenti di configuration management
- Formazione del personale
- …

Indicatori progetto SW

- Analisi dei requisiti
- Configuration Management
- Documentazione dei requisiti
- Architettura software
- Linguaggio di programmazione e librerie
- Correttezza del codice
- Riusabilità
- Flusso di validazione
- Indipendenza dal team di progetto
- Uso di metriche
- Gestione dei fornitori
- Definizione delle responsabilità
- Documentazione dell'architettura
- Specifiche tecniche
- Documentazione tecnica di dettaglio
- Manuali d'uso
- Strumenti di sviluppo
- Strumenti di test e debug
- Strumenti di Configuration Management
- Formazione del personale
- …

Nota: questo set di indicatori non è esaustivo e non si riferisce ad alcun progetto specifico, ma ha soltanto scopo esplicativo

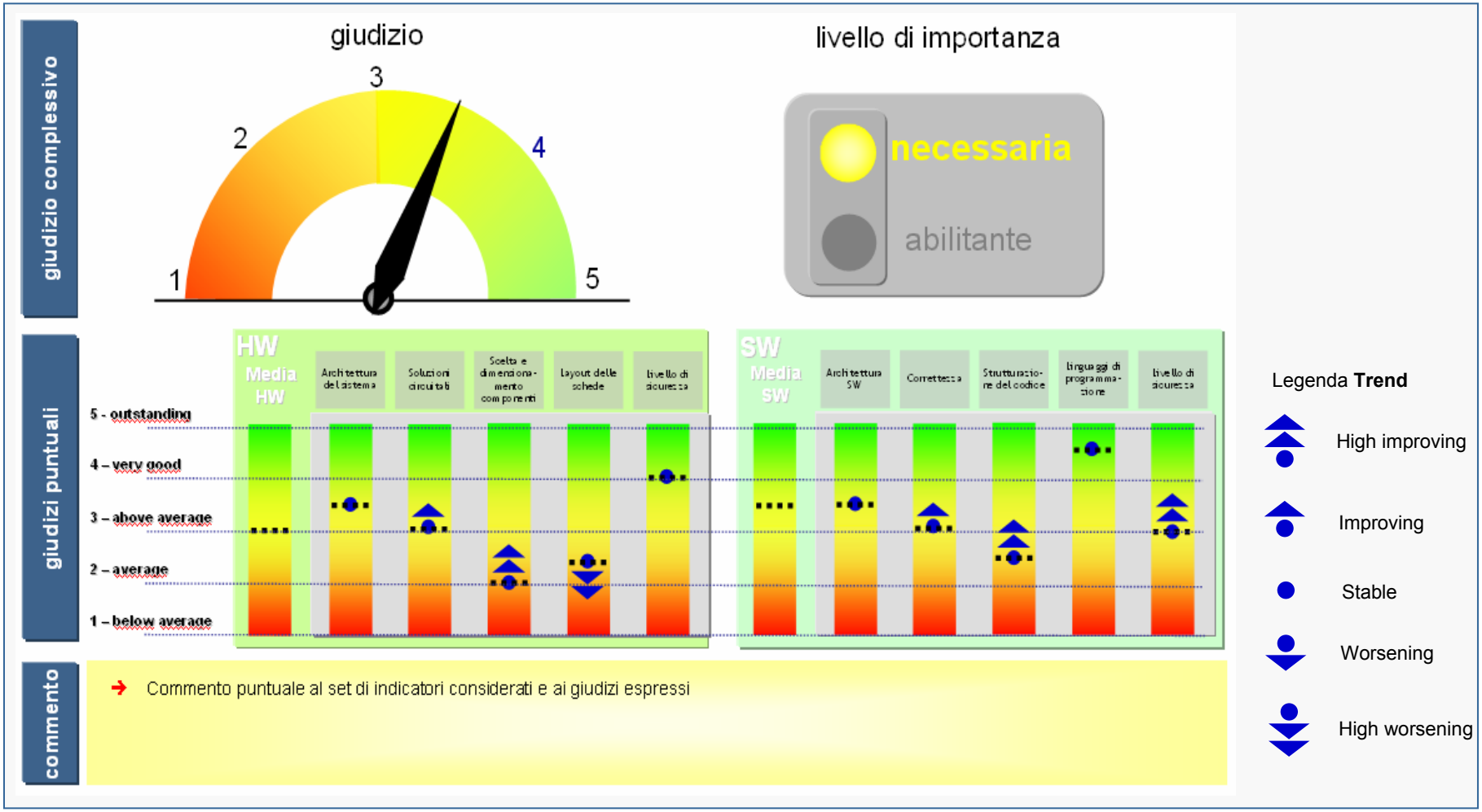
Product Life Cycle



Nel processo di assessment si tiene conto di tutte le fasi del ciclo di vita del prodotto, con particolare riferimento a quelle di progettazione e sviluppo

Assessment R&D
un esempio di evaluation dashboard

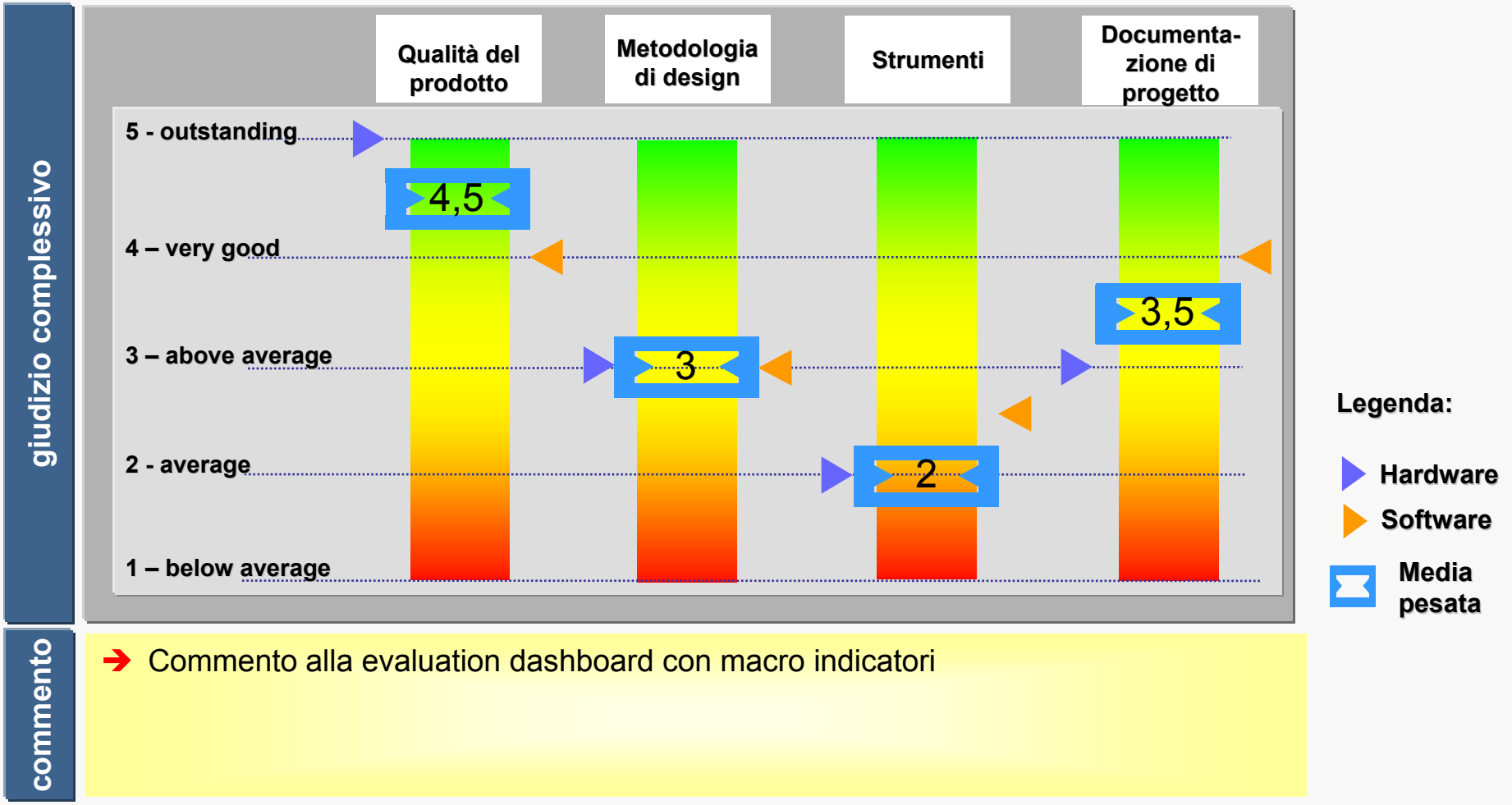
I risultati dell'assessment sono riassunti in una serie di "evaluation dashboard" grafiche per semplicità e immediatezza di utilizzo



Nota: questa evaluation dashboard non si riferisce ad alcun progetto specifico, ma ha soltanto scopo esplicativo

Assessment R&D
un esempio di evaluation dashboard sintetica

Evaluation dashboard sintetica (basata su macro indicatori)
adatta per presentazioni al top management



Nota: questa evaluation dashboard non si riferisce ad alcun progetto specifico, ma ha soltanto scopo esplicativo

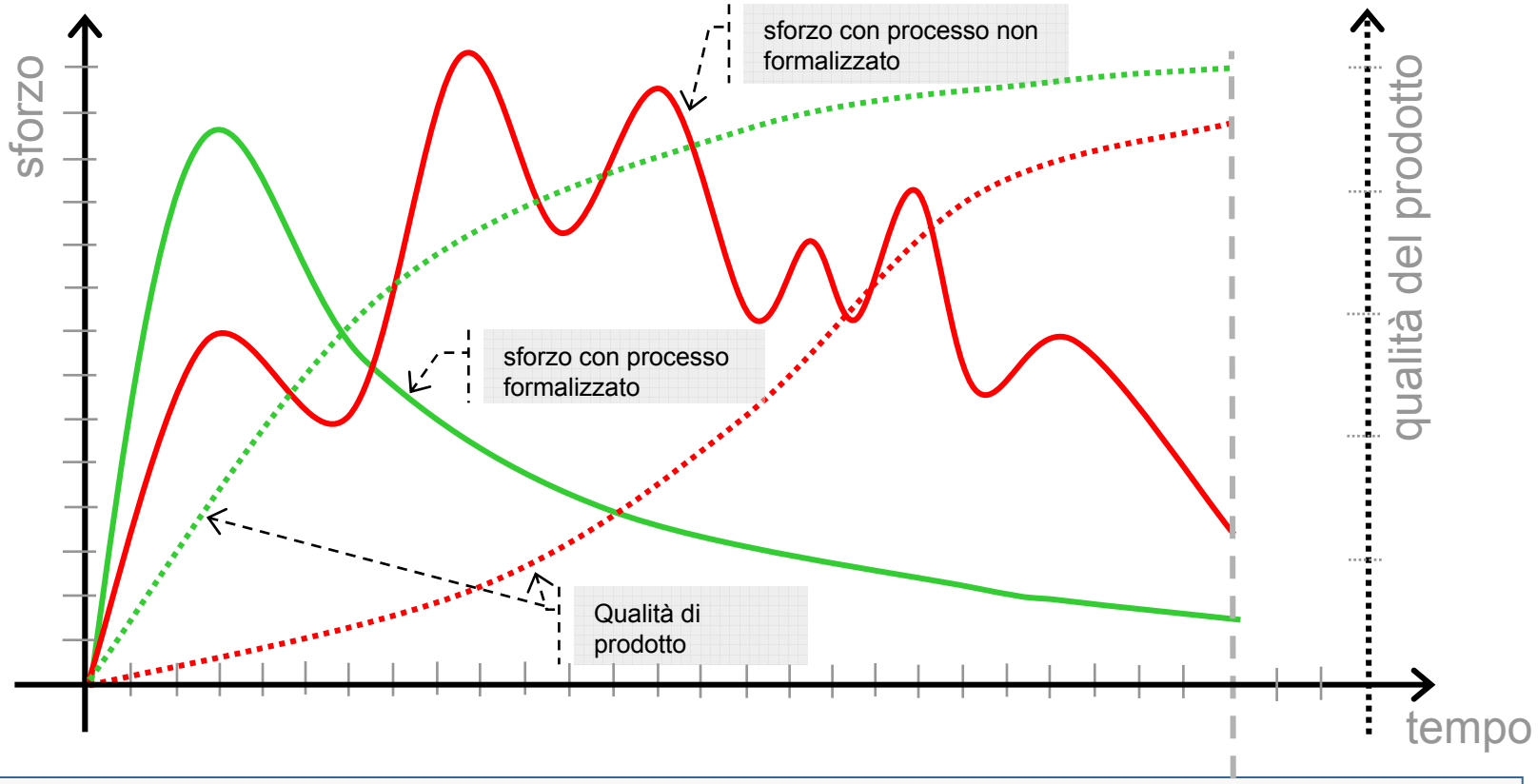
CEFRIEL expertise

CEFRIEL, come ente terzo “super partes”, fornisce alle aziende il servizio di assessment delle loro attività di R&D e NPD

- Determinazione delle metriche più adatte (*key indicators*) per l’assessment degli scenari R&D e NPD sotto analisi
- Progettazione di un sistema di misure per le metriche proposte
  - Realizzazione, se necessario, di laboratori di test e misure ad-hoc presso la struttura CEFRIEL per i prodotti sotto analisi
- Analisi e aggregazione dei risultati delle misure effettuate
- Integrazione dei risultati ottenuti nel sistema di knowledge management dell’azienda
  - Riorganizzazione della documentazione tecnica (scritta e orale), se necessario
  - Realizzazione di un documento di valutazione
- Realizzazione di dashboard grafici di valutazione, da aggiornare periodicamente, per il governo delle attività di R&D e NPD
- Mentoring e supporto per l’implementazione delle azioni migliorative/correttive indicate
- Valutazione ex post dell’impatto risultante dall’assessment sulla divisione R&D e sull’intera impresa

Processo di ricerca e sviluppo
Sforzo puntuale

andamento sforzo richiesto/qualità del prodotto nel tempo



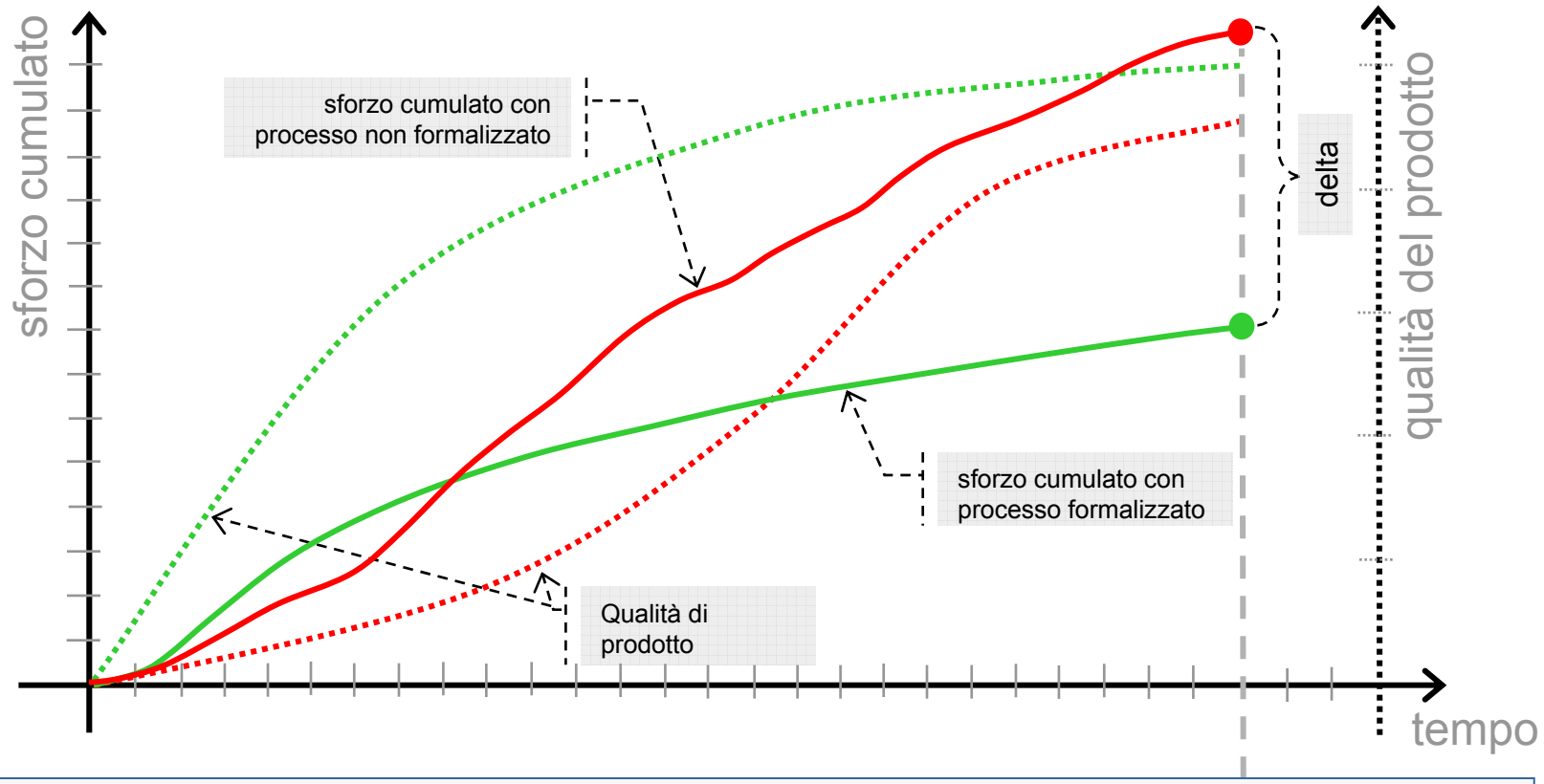
considerazioni

- Processi di sviluppo "caotici" (livello 1 CMMI)
  - sono basati sul buon senso
  - sono definiti di volta in volta
  - sono difficilmente predicibili e governabili
  - producono qualità adeguate in funzione delle capacità individuali dei professionisti
- L'adozione di un processo di sviluppo rigoroso
  - riduce lo sforzo e il tempo necessari al raggiungimento di una qualità di prodotto coerente con le esigenze di impresa
  - aumenta la probabilità di successo del prodotto

Nota: gli andamenti riportati in questa slide non si riferiscono ad alcun progetto specifico, ma hanno soltanto scopo esplicativo

Processo di ricerca e sviluppo
Sforzo cumulato

andamento sforzo richiesto/qualità del prodotto nel tempo



considerazioni

- Processi di sviluppo "caotici" (livello 1 CMMI)
  - sono basati sul buon senso
  - sono definiti di volta in volta
  - sono difficilmente predicibili e governabili
  - producono qualità adeguate in funzione delle capacità individuali dei professionisti
- L'adozione di un processo di sviluppo rigoroso
  - riduce lo sforzo e il tempo necessari al raggiungimento di una qualità di prodotto coerente con le esigenze di impresa
  - aumenta la probabilità di successo del prodotto

Nota: gli andamenti riportati in questa slide non si riferiscono ad alcun progetto specifico, ma hanno soltanto scopo esplicativo