ELETTRIC80

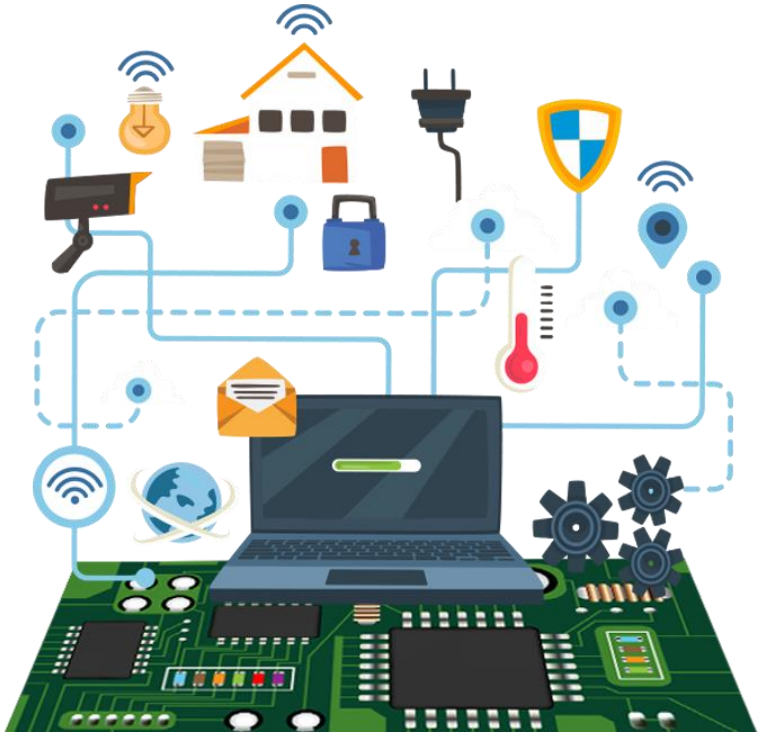
30.11.2019 – Qualità del codice nei sistemi embedded

Sistemi embedded

- Un ***Sistema Embedded*** è un Sistema informatico dedicato, pensato cioè per svolgere un compito preciso e determinate: svolgere poche o specifiche operazioni.
- Sono spesso parte integrante di sistemi più grandi di cui sono alla base del funzionamento con compiti di controllo, elaborazione, memorizzazione…

Principali requisiti

- Principali requisiti di progetto imposti a un sistema embedded :
  - *Costi.*
  - *Dimensioni, prestationi, consumi.*
  - *Tempi di sviluppo (time-to-market).*
  - *Flessibilità.*
  - *Robustezza.*
  - *Aggiornabilità.*



Complessità vs. qualità del codice

- ***Processi antagonisti :***
  - Aumento della potenza computazione e conseguente aumento delle funzionalità inserite all'interno dei dispositivi embedded (più linee di codice).
  - *Time-to-market* e pianificazione dei vari cicli di progettazione.
  - Co-design hardware e software: forte iterazione e collaborazione tra i vari reparti di sviluppo e i diversi attori della supply chain.

- **Necessario garantire la qualità del software.**

Qualità del software

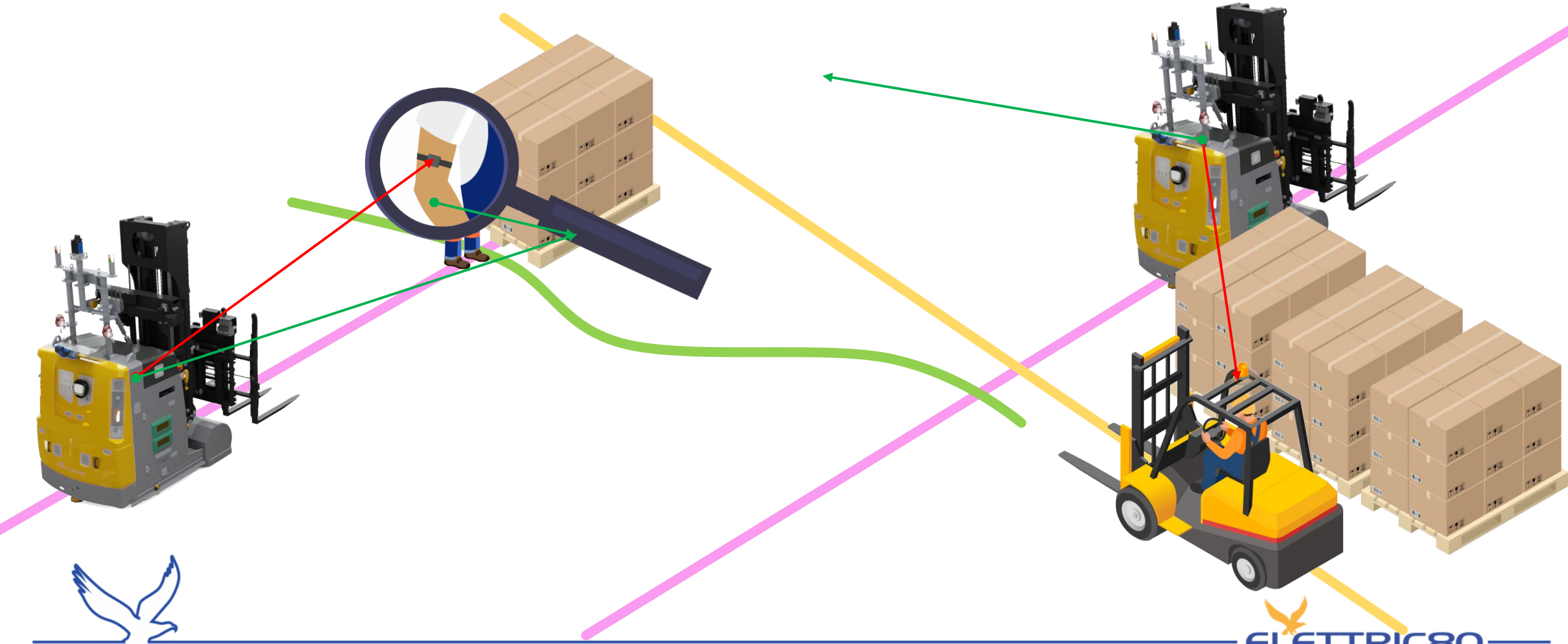
- Richiesta elevata affidabilità di funzionamento e la capacità di fornire performance di tipo deterministico.
  - *Sistemi in campo medicale.*
  - *Sistemi in ambito automobilistico.*
  - *Sistemi per uso industriale.*
  - *Funzionalità di sicurezza.*

- La qualità del software non può essere un oggetto di compromesso.

- ***Utilizzo di varie pratiche di sviluppo e produzione nei vari domini dell'embedded mirate a mantenere una elevata la qualità del software.***

ELETTRIC80

Qualità del software: un esempio



Qualità del software :

Manutenibilità

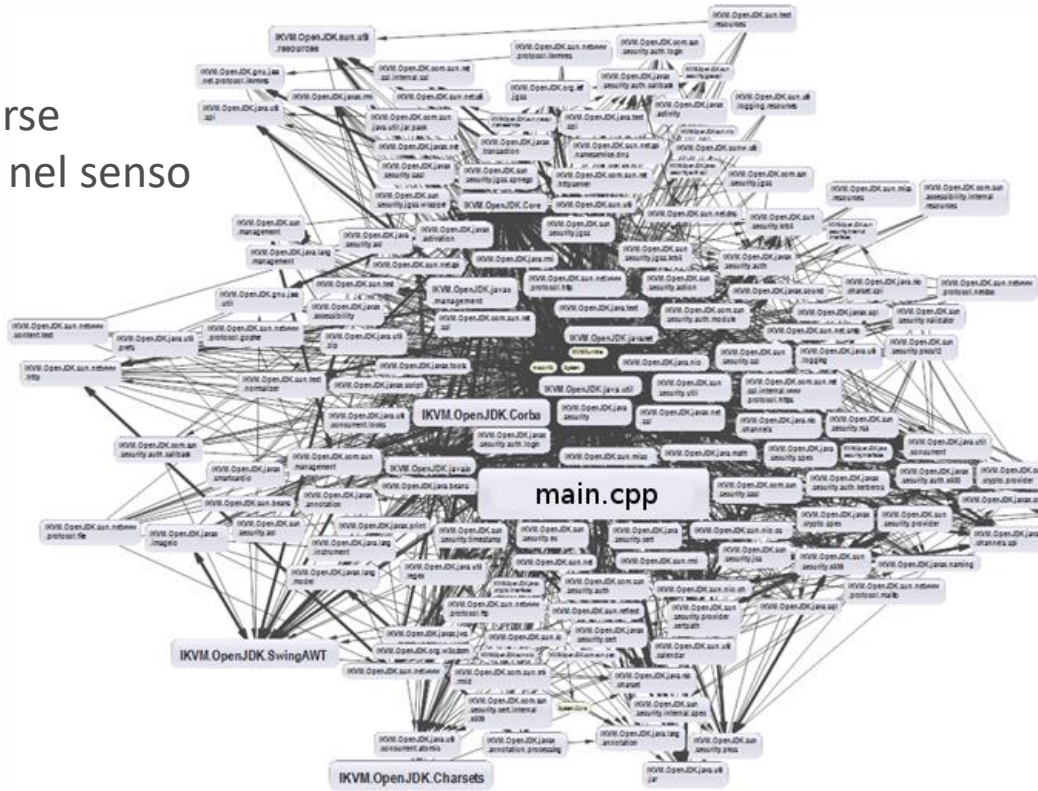
- ***Fase di progettazione e design:*** necessità di rispettare tutti i requisiti nonostante le pressioni dovute al time-to-market ed al co-design hardware e software.
  - *Necessario adottare tecniche per l'eliminazione di bug software già nelle prime fasi di sviluppo (tool di sviluppo e simulazione, development board, modelli matematici).*

- ***Crescita del progetto:*** Componenti e moduli del progetto sempre più articolati e indipendenti. Aumento della difficoltà nell'eseguire le operazioni di manutenzione, modifica e test.
  - *Ridurre la rigidità del codice.*

Dipendenze buone e cattive

- A livello architetturale, un'elevata interdipendenza dei sottosistemi, può provocare a cascata la necessità di apportare una serie di cambiamenti in tutti i vari pacchetti e di conseguenza una scarsa evoluzione e manutenibilità del codice.

- Un codice robusto, manutenibile e riutilizzabile si caratterizza per le scarse interdipendenze. E dove presenti, queste interdipendenze sono buone, nel senso che dipendono da classi stabili.



Ottimizzazione vs. portabilità

- Un software di qualità si contraddistingue per flessibilità, solidità e stabilità.
- Un codice per piattaforme embedded però è spesso un codice “su misura” che ottimizza caratteristiche specifiche del sistema:
  - Uso della memoria (accesso diretto ad aree «speciali»)
  - Velocità di esecuzione (utilizzo di busy-wait calibrate)
  - Dialogo con le periferiche (cicli di attesa)

- *Separare il codice applicativo dal codice specifico della piattaforma su cui si sta operando.*

C++ nei sistemi embedded industriali

- *Favorisce la portabilità.*
- *Fornisce un alto livello di astrazione e libertà espressiva.*
- *Impone una elevata conoscenza del linguaggio e delle librerie per non incorrere in errori dovuti all'elevato livello di astrazione.*

ELETTRIC80

THANK YOU FOR YOUR ATTENTION!