Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

DETAILED DESIGN MANAGEMENT
UNIT CONSTRUCTION MANAGEMENT

Maria Teresa Palomba – A-Key S.r.l.
t.palomba@a-key.it

a-key

Riferimenti:

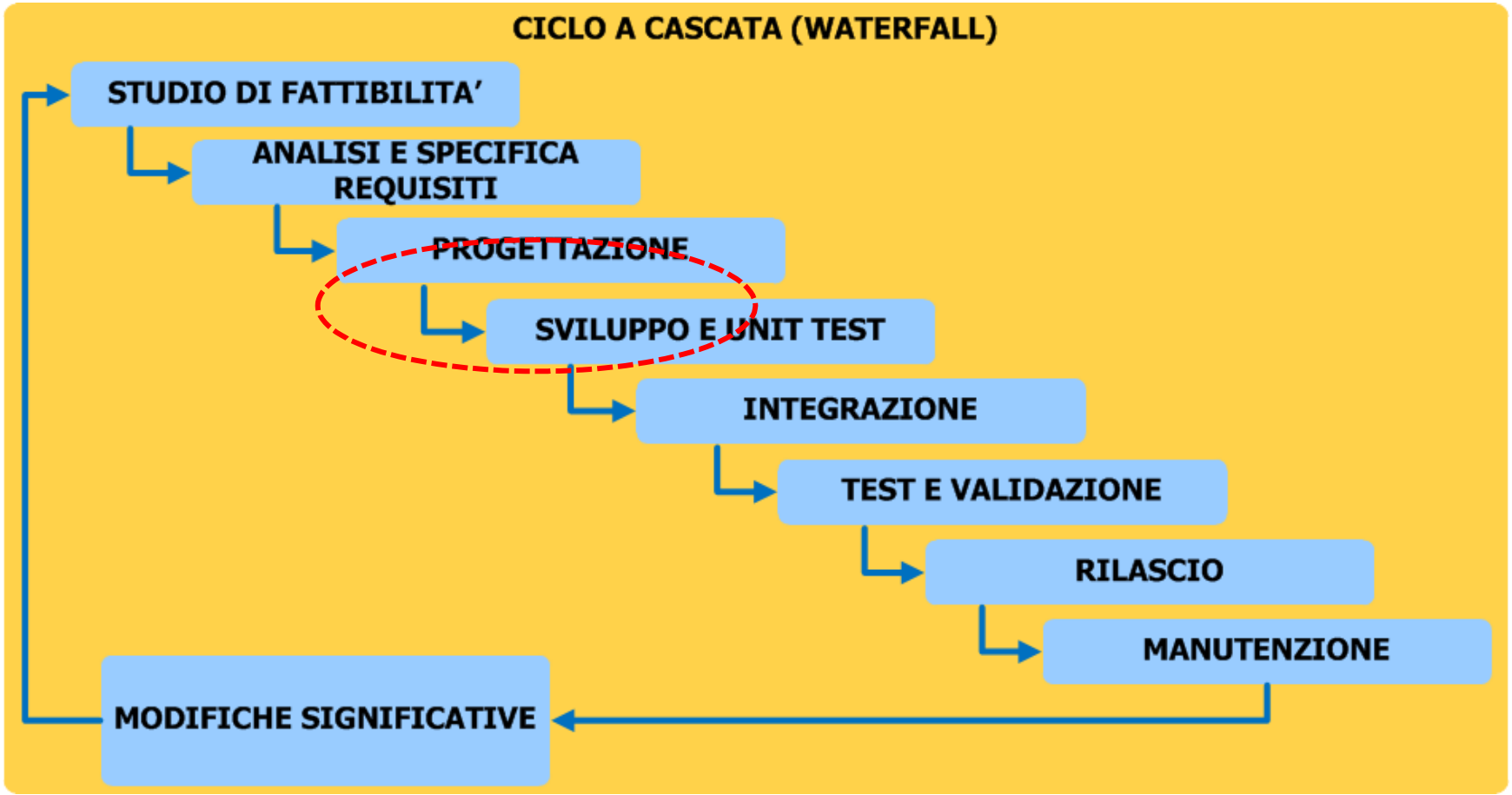
A. http://www.computer.org/web/swebok - Software Engineering Body of Knowledge (SWEBOK), version 3.0

B. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm modello CMMI-DEV v. 1.3

C. http://www.automotivespice.com/fileadmin/software-download/Automotive_SPICE_PAM_30.pdf - Automotive SPICE PAM PRM v.3.0

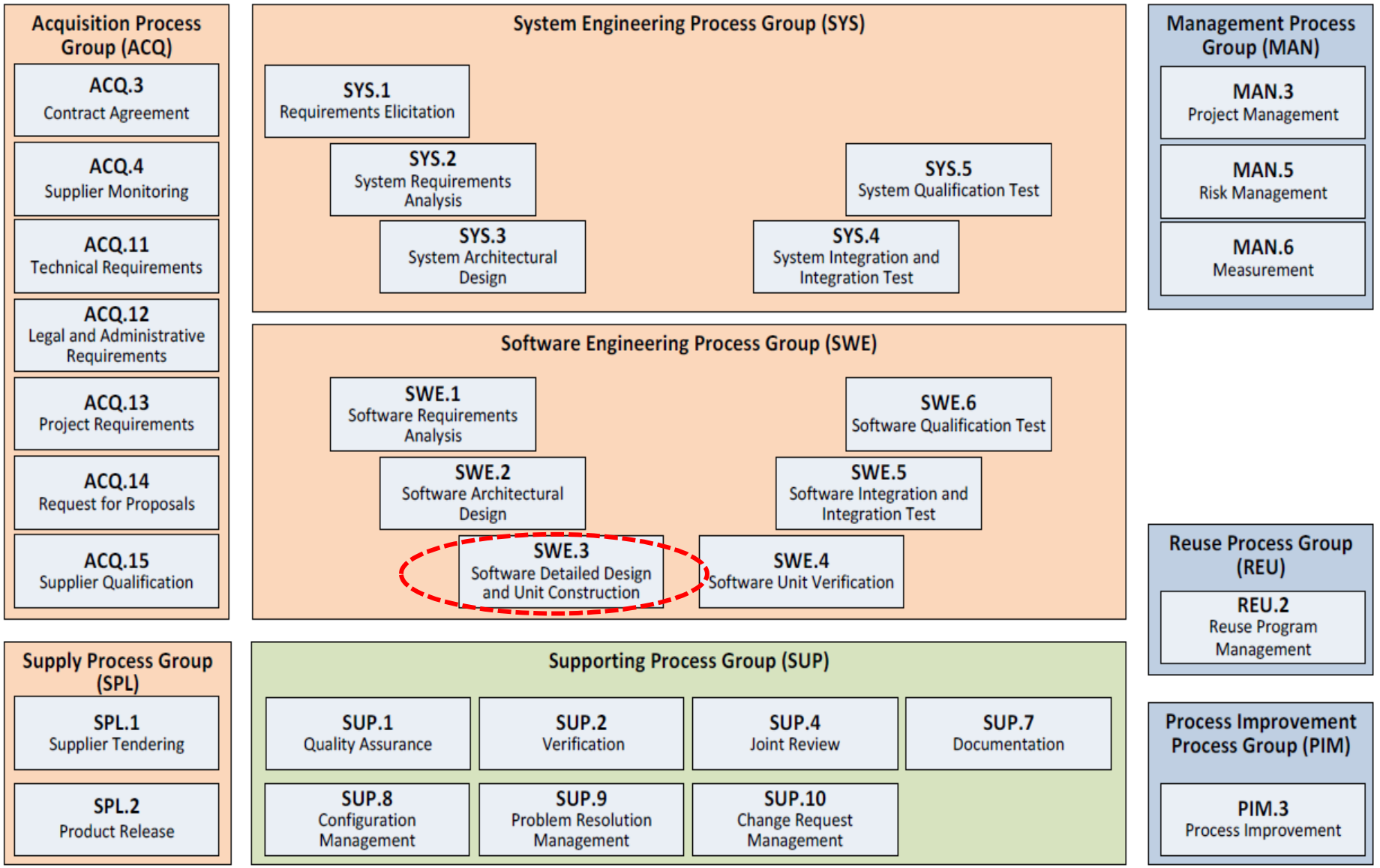
D. http://www.sei.cmu.edu/reports/00sr004.pdf - SEI CMU - Software Architecture Documentation in Practice: Documenting Architectural Layers

Ciclo di vita del Software



a-key

Automotive SPICE process reference model - Overview



SWEBOK

Software design is generally considered a two-step process:

- **Architectural design** (also referred to as high level design and top-level design) describes **how software is organized into components**.
- **Detailed design** describes the **desired behavior** of these **components.**

Automotive SPICE V3.0

An **architecture** consists of **architectural "elements"** that can be further decomposed into more fine-grained **architectural sub-"elements"** across appropriate hierarchical levels.

The term "**software component**" is used for the **lowest level elements** of the software architecture **for which finally the detailed design is defined**.

A software "**component**" consists of **one or more software "units"**

A **software unit** is a software component that is **not further subdivided.**

SOFTWARE DETAILED DESIGN AND UNIT CONSTRUCTION

The purpose of the Software Detailed Design and Unit Construction Process is to:

- provide an **evaluated detailed design for the software units**
- **produce the software units.**

Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

SOFTWARE DETAILED DESIGN

a-key

SEI Rif. D

"The process of engineering design has the following generic steps:

a) define the problem
b) gather pertinent information
c) generate multiple solutions
d) analyze and select a solution
e) implement the solution

All of the engineering design steps are **iterative**, and knowledge gained at any step in the process may be used to inform earlier tasks and trigger an iteration in the process. "

Obiettivo:

- Definire le specifiche **a più basso livello** dei componenti SW:
  - Interfacce interne
  - Algoritmi
  - Obiettivi di consumo delle risorse
  - Specifiche dei dati
  - Comportamento dinamico

Fa riferimento **all'Architectural Design** per le specifiche delle **interfacce esterne** e delle **interazioni esterne**

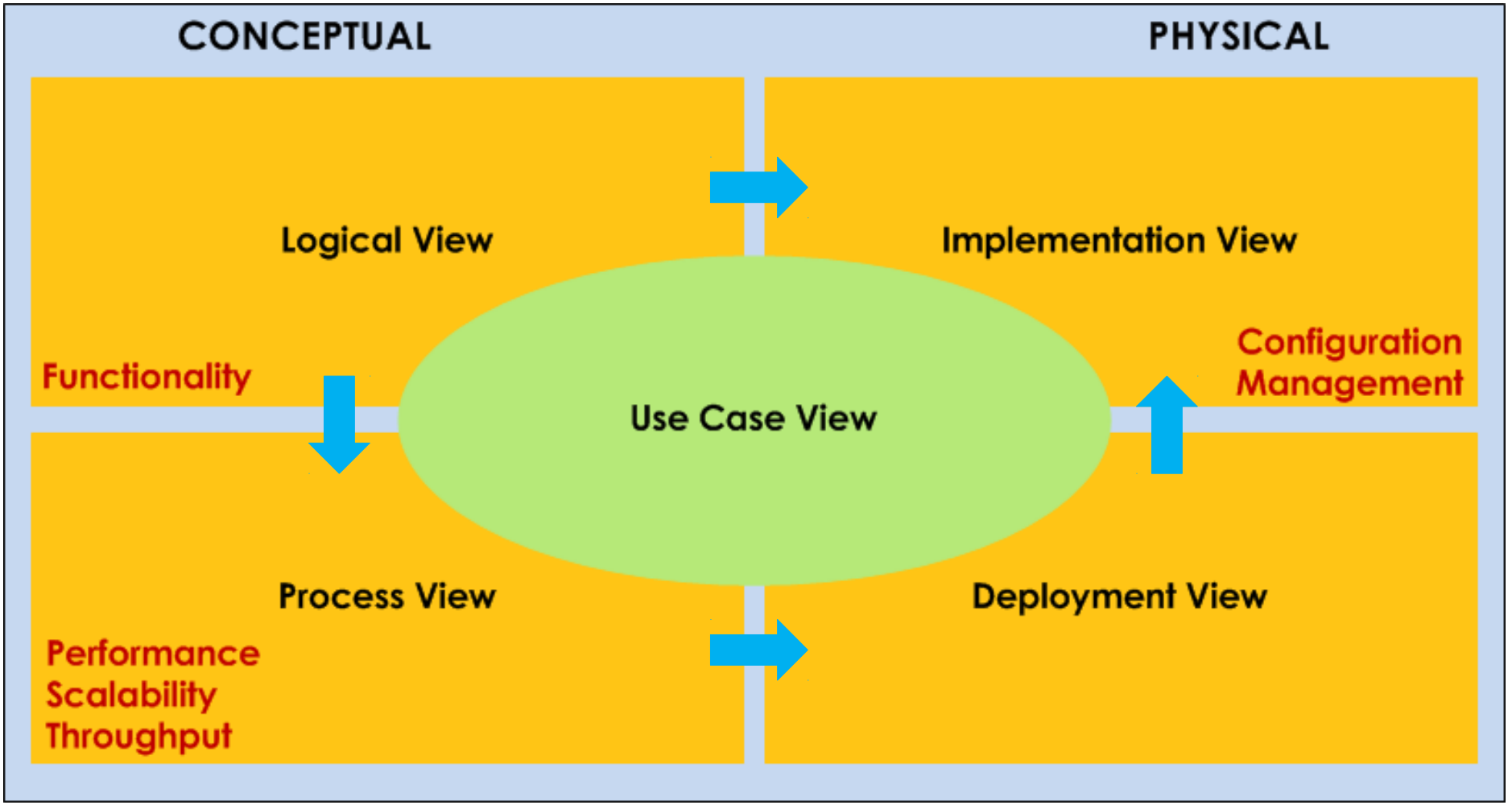
Modi di rappresentazione e creazione:

- Linguaggio naturale
- Linguaggi simbolici
- Linguaggi di modellazione (es. UML)

Spesso prodotto tramite **tool automatici** che possono **generare il codice sorgente** e **simularne il comportamento**

Modello di viste architetturali (Kruchten – 4+1)

- Insieme di viste che rappresentano l'architettura dalle diverse prospettive



Criteri di ingresso:

- Software Architectural Design approvato
- Software requirements specification approvato
- Informazioni di tracciabilità tra SW requirements e SW Architecture

Criteri di uscita:

- Scomposizione del sistema contenente la top-level structure e tutti i SW components che implementano i SW requirements
- Per ogni SW component, descrizione delle SW units e loro interfacce (implementabili e testabili)
- Allocazione di tutti i SW requirements ai SW components
- Tracciabilità tra SW Architecture e SW Detailed Design

Master Sistemi Embedded per Internet of Things

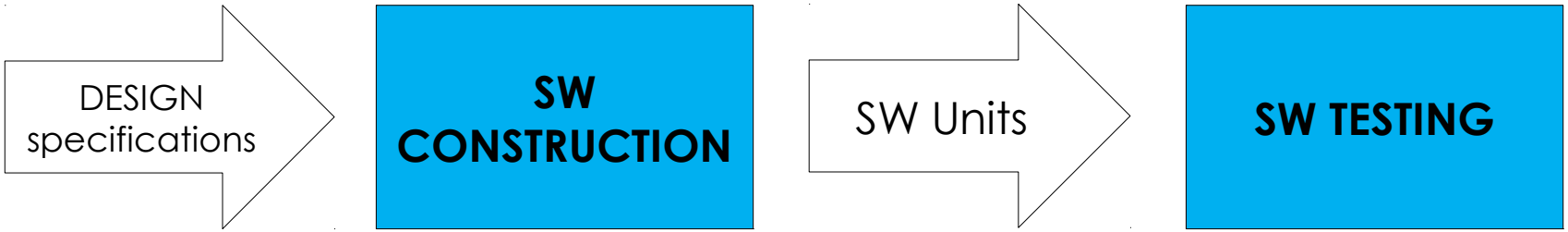
CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

SOFTWARE CONSTRUCTION

a-key

SOFTWARE CONSTRUCTION

"**creazione di software funzionante** tramite una combinazione di **codifica, verifica, unit testing, integration testing, and debugging.** "



I confini tra design, construction e testing dipendono dal ciclo di vita selezionato

- Molta parte del lavoro di Design è effettuata durante la construction
- Parte del lavoro di Test è effettuata durante la construction

SWEBOK

«**Software construction fundamentals** include:

- **minimizing complexity**
- **anticipating change**
- **constructing for verification**
- **reuse**
- **standards in construction**

The first four concepts apply to design as well as to construction."

Minimizzare la complessità

Privilegiare la **creazione di codice che sia semplice e leggibile** piuttosto che "brillante e intelligente"

Tecniche:

- Utilizzare gli **standard di codifica**
- Applicare il **design modulare**
- Utilizzare **tecniche di codifica**
- Utilizzare **tecniche di qualità**

Anticipare il cambiamento

Il software e il suo ambiente operativo cambiano nel tempo

Anticipare il cambiamento permette di **costruire software estendibile**

→ il software può essere migliorato senza doverne rimaneggiare pesantemente la struttura

Tecniche:

- Utilizzare gli **standard di codifica**
- Applicare il **design modulare**
- Utilizzare **tecniche di codifica**
- Utilizzare **tecniche di qualità**

Costruire per la Verifica

Costruire il software in modo tale che le **anomalie possano essere facilmente individuate**:

- dagli sviluppatori
- dai tester
- dagli utenti

Tecniche:

- Seguire gli **standard di codifica**
- Eseguire le **code review**
- Eseguire lo **unit testing**
- **Strutturare il codice in modo che permetta di effettuare test automatici**
- **Limitare l'uso di strutture sintattiche complesse o difficili da capire**

Standard di codifica

Applicare standard di sviluppo del codice durante la costruzione.

Vantaggi → miglior controllo di

- **Produttività**
- **Qualità**
- **Costi**
- **Sicurezza**

Tipi di standard che indirizzano specifiche problematiche della codifica:

- Metodiche di comunicazione (formati e contenuti delle comunicazione)
- Linguaggi di programmazione
- Standard di codifica (naming conventions, layout, indentazione, annidamento)
- Piattaforme
- Standard di modellazione

Riutilizzo del codice

Utilizzare **elementi esistenti** per risolvere **altri problemi**

Es.: librerie, moduli, componenti, codice sorgente, commercial off-the-shelf (COTS)

Best practices per il riutilizzo:

- Gestire il riutilizzo in modo **sistematico**
- Applicare un **processo** di riutilizzo ben **definito e ripetibile**

Vantaggi → miglioramento di

- **Produttività**
- **Qualità**
- **Costi**

Riutilizzo del codice

- **Costruzione per il riutilizzo** → Creare oggetti software riutilizzabili

- **Costruzione con il riutilizzo** → Riutilizzare oggetti software già esistenti per la costruzione di altri progetti

Costruzione per il riutilizzo

- Creare software **potenzialmente riutilizzabile**
- Prospettiva ampia, valutazioni multisistema, disegno per il riutilizzo

Tecniche:

- **Incapsulare il codice riutilizzabile** in librerie o componenti ben strutturati
- **Implementare la variabilità** con meccanismi come parametrizzazione, compilazione condizionale, design patterns
- **Incapsulare la variabilità** per rendere il software facile da configurare e adattare
- **Testare la variabilità**
- **Descrivere e documentare** il software riutilizzabile

Costruzione con il riutilizzo

Creare nuovo software riutilizzando software esistente

Es:

- Utilizzo delle librerie fornite dal linguaggio, dalla piattaforma dal tool
- Utilizzo di librerie open-source
- Utilizzo di software COTS
- Riutilizzo di software sviluppato per altri progetti

Metodica:

A. **Selezionare le units riutilizzabili**
B. **Valutare la riutilizzabiltà** delle units individuate
C. **Integrare le units riutilizzabili nel software attuale**
D. **Documentare e riportare le informazioni sul riutilizzo sul nuovo codice, test e dati di test**

Problematiche di codifica

A. Tecniche per produrre codice comprensibile:

a) Naming conventions
b) Layout del codice
c) Indentazione
d) Annidamento
e) Commenti

B. Utilizzo delle strutture di controllo

C. Gestione degli errori e delle eccezioni

D. Gestione dell'utilizzo delle risorse:

a) Meccanismi di esclusione e disciplina dell'accesso alle risorse

E. Organizzazione del codice

F. Documentazione del codice

Metodologie per la Qualità del software

- Test-first development
- Uso delle asserzioni
- Uso del defensive programming
- Debugging
- Unit testing ed integration testing
- Ispezioni del codice (code inspections)
- Riesami tecnici
- Analisi statica

Metodologie per la Qualità del software

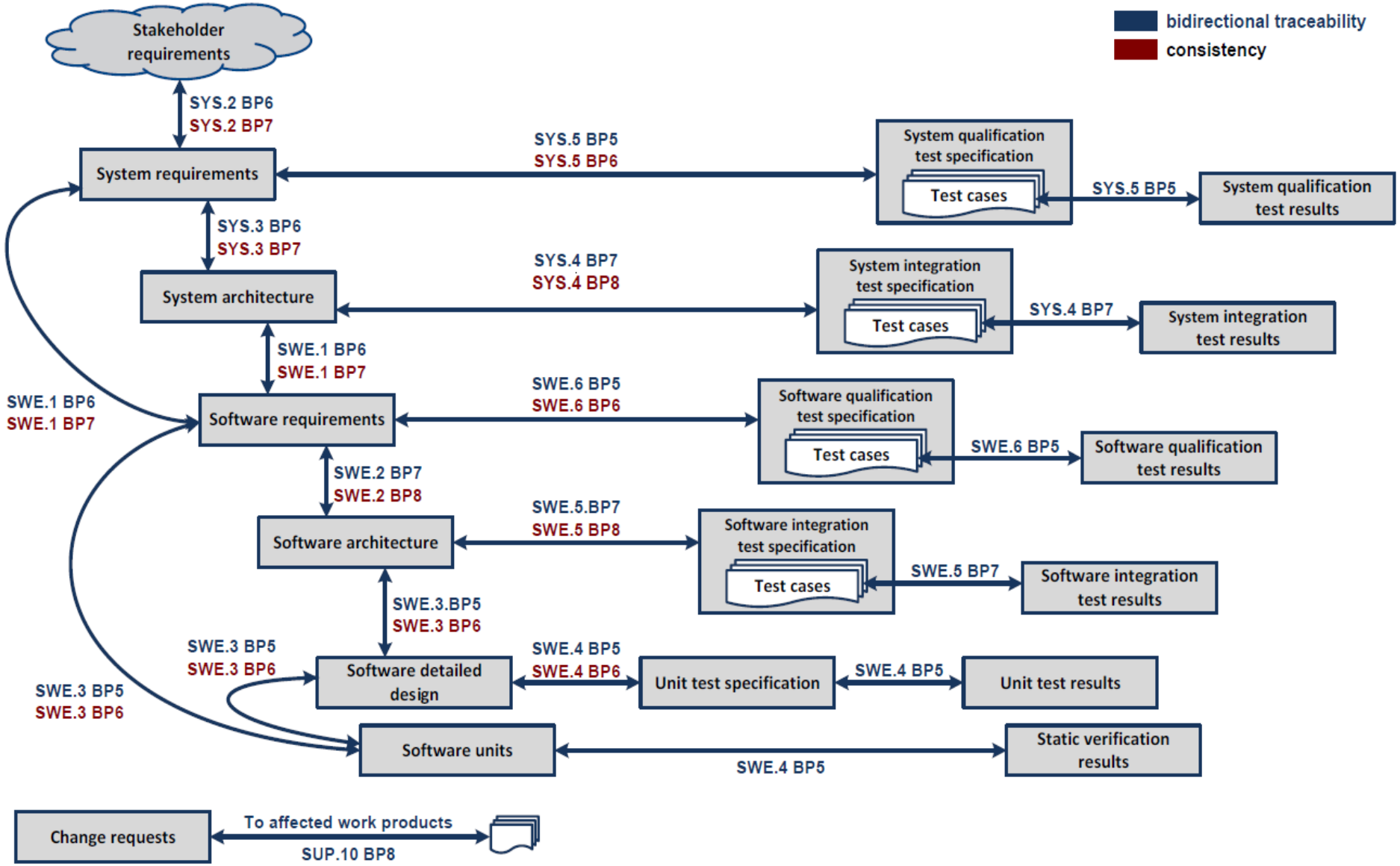
Defensive programming (Secure programming)

Utilizzato **quando si ipotizza un uso errato o scorretto** del software

Migliora il codice in termini di:

- **Qualità generale**: riduzione di defects e anomalie → i difetti possono essere sfruttati per attacchi malevoli
- **Comprensibilità del codice**
- **Predicibilità del comportamento** del software **anche in casi d'uso non previsti**

Automotive SPICE – Traceability and Consistency



Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

CMMI- DEV
AUTOMOTIVE SPICE

a-key

Process areas

TECHNICAL SOLUTION - TS

"The purpose of Technical Solution (TS) is to select, design, and implement solutions to requirements. Solutions, designs, and implementations encompass products, product components, and product related lifecycle processes either singly or in combination as appropriate."

TECHNICAL SOLUTION - TS

Specific Goal and Practice Summary

- **SG 1 Select Product Component Solutions**
  - SP 1.1 Develop Alternative Solutions and Selection Criteria
  - SP 1.2 Select Product Component Solutions
- **SG 2 Develop the Design**
  - SP 2.1 Design the Product or Product Component
  - SP 2.2 Establish a Technical Data Package
  - SP 2.3 Design Interfaces Using Criteria
  - SP 2.4 Perform Make, Buy, or Reuse Analyses
- **SG 3 Implement the Product Design**
  - SP 3.1 Implement the Design
  - SP 3.2 Develop Product Support Documentation

Process purpose

SWE.3 Software Detailed Design and Unit Construction

"The purpose of the Software Detailed Design and Unit Construction Process is to **provide an evaluated detailed design** for the software units and to **produce the software units**"

Process outcomes

1) **interfaces of each software unit** are defined
2) **dynamic behavior** of the software units is defined
3) **consistency and bidirectional traceability** are established between:
    1) **software requirements** and **software units**
    2) **software architectural design** and **software detailed design**
    3) **software detailed design** and **software units**
4) **software detailed design** and the **relationship to the software architectural design** is **agreed** and **communicated** to all affected parties
5) **software units** defined by the software detailed design **are produced**

Base Practices

- SWE.3.BP1: Develop **software detailed design**.
- SWE.3.BP2: Define **interfaces of software units**.
- SWE.3.BP3: Describe **dynamic behavior**.
- SWE.3.BP4: **Evaluate software detailed design**.
- SWE.3.BP5: Establish **bidirectional traceability**.
- SWE.3.BP6: Ensure **consistency**.
- SWE.3.BP7: **Communicate agreed software detailed design**.