Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

ARCHITECTURAL DESIGN 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

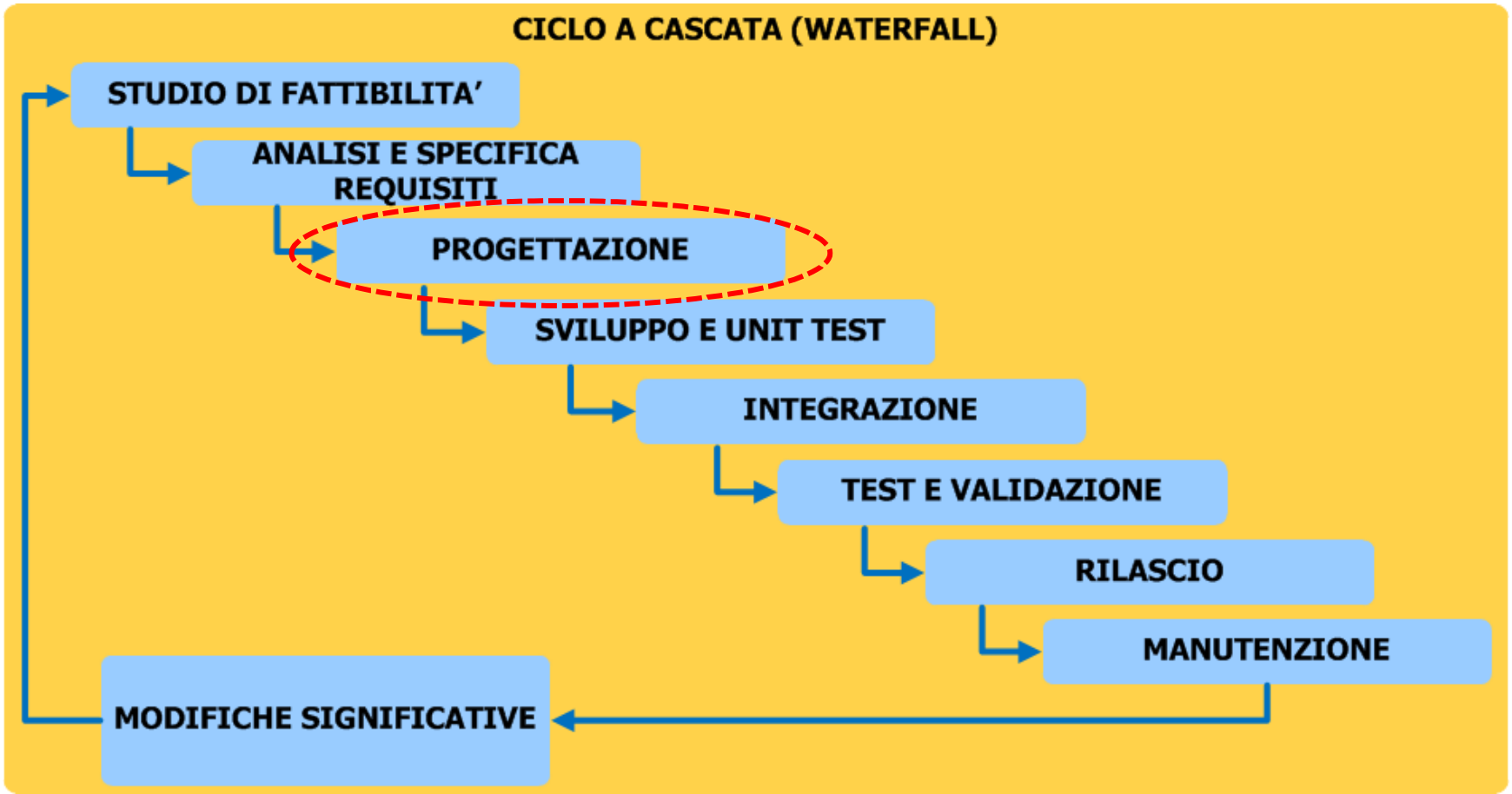
B. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm modello CMMI-DEV v. 1.3

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

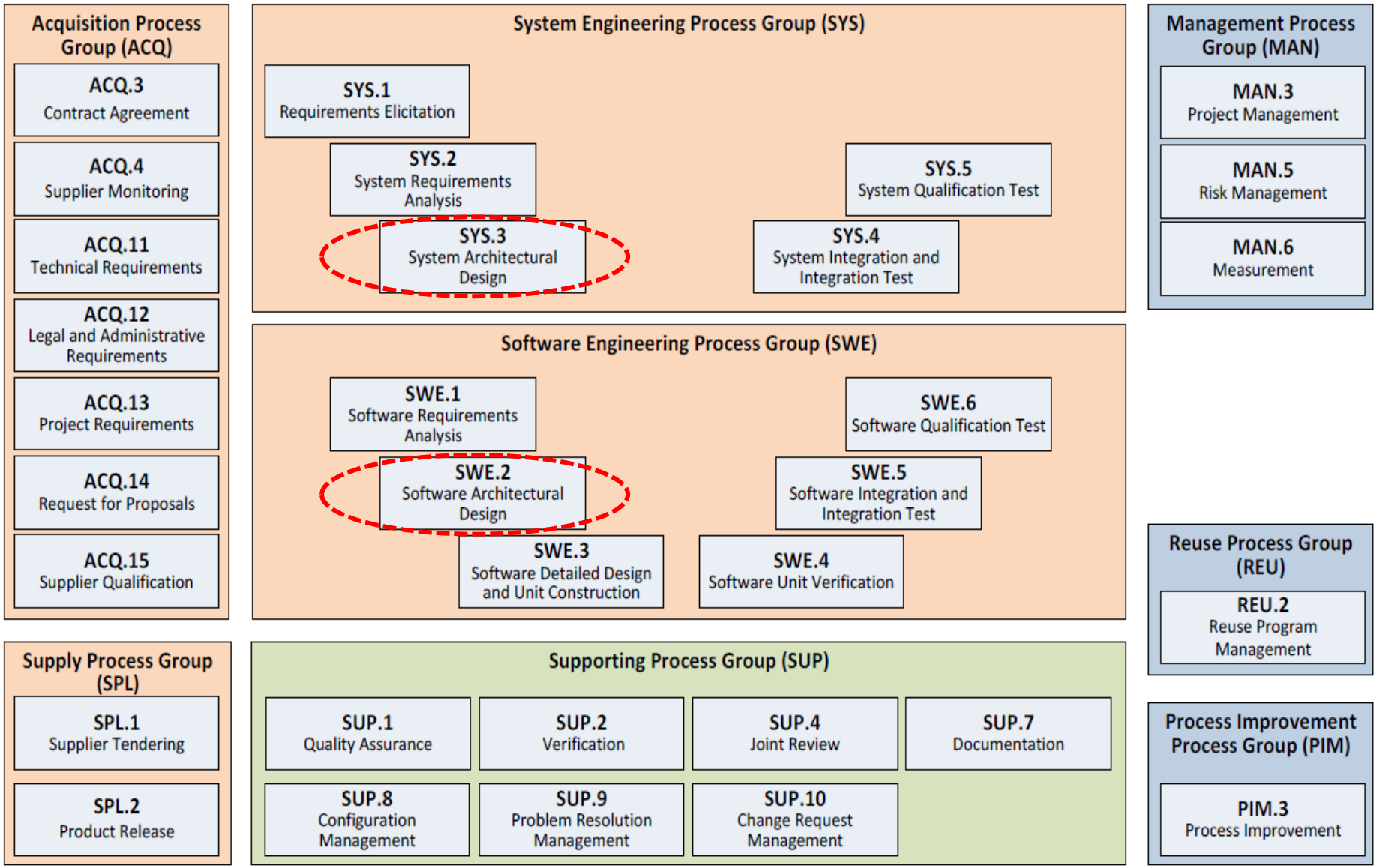
E. http://www.autosar.org/ - AUTOSAR development partnership

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.**

Obiettivo:

- Analizzare e considerare **tutti i requisiti** del software
- Definire le **High Level Software Structures** che permettono di soddisfare i requisiti
- **Descrivere** il sistema da **diverse prospettive**

Risultato → documentazione di:

- Come il sistema è **scomposto ed organizzato in componenti**
- Quali sono le **interfacce** tra i componenti
- Qual è il **comportamento dinamico** del sistema

Modi di rappresentazione:

- Diagrammi a blocchi che rappresentano il comportamento e le relazioni dei componenti software
- Descrizione tecnica in linguaggio naturale
- Linguaggi di modellazione (es. UML)

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. "

SYSTEM ARCHITECTURAL DESIGN

The purpose of the System Architectural Design Process is to:

- **establish** a **system architectural design**
- **identify which system requirements are to be allocated to which elements of the system**
- **evaluate** the system architectural design against defined criteria.

SOFTWARE ARCHITECTURAL DESIGN

The purpose of the Software Architectural Design Process is to:

- **establish** an **architectural design**
- **identify which software requirements are to be allocated to which elements** of the software
- **evaluate** the software architectural design against defined criteria.

SWEBOK

«**Software Design Principles** are key notions that provide the basis for many different software design approaches and concepts:

- **abstraction**
- **coupling and cohesion**
- **decomposition and modularization**
- **encapsulation/ information hiding**
- **separation of interface and implementation**
- **sufficiency, completeness, and primitiveness**
- **separation of concerns**"

Astrazione

"a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information"

Meccanismi di astrazione:

- **parametrizzazione** : astrarre la rappresentazione dei dati dai dettagli dei dati, utilizzando i parametri
- **specifica** :
  - Astrazione procedurale
  - Astrazione dei dati
  - Astrazione del controllo

Scomposizione e modularizzazione

- **Elementi grandi di software sono suddivisi** in più **componenti di dimensioni inferiori**
- **Le interfacce** che descrivono le interazioni tra componenti **sono ben definite**
- **I moduli possono essere creati indipendentemente ed utilizzati in sistemi diversi**

Obiettivo:

- allocare **funzionalità e responsabilità diverse** su **diversi componenti**
- **Indipendenza dei componenti**

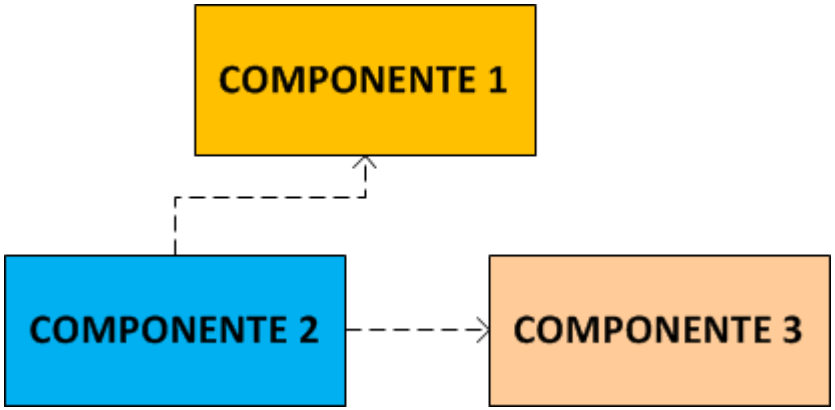
Scomposizione e modularizzazione

Gerarchizzazione dei moduli

- Riduzione delle dipendenze
- Definizione della topologia delle relazioni tra i moduli
- Scomposizione del dominio del problema
- Base per la parallelizzazione dello sviluppo
- Base per la segregazione delle modifiche e del versionamento
- Base per la prototipazione rapida

Dipendenze

Le dipendenze limitano la riusabilità di un componente



- Componente 2 dipende dai componenti 1 e 3 :
- Il funzionamento di C2 dipende dal funzionamento di C1 e C3
- Una anomalia in C1 o C3 comporta anomalia nel funzionamento di C2
- Una modifica in C1 o C3 comporta in genere la modifica anche di C2
- Il riutilizzo di C2 richiede anche il riutilizzo di C1 e C3

Accoppiamento e coesione

- **Accoppiamento** = misura della **interdependenza tra moduli** in un programma SW

- **Coesione** = misura della **coerenza funzionale degli elementi all'interno di un modulo**

Principio di buona progettazione dei moduli:

- **Alta coesione**
- **Basso accoppiamento**

Incapsulamento e information hiding

- **Raggruppare e impacchettare** i **dettagli interni** di una astrazione
- Rendere i dettagli interni **inaccessibili ad entità esterne**

Separazione delle interfacce e della loro implementazione

Separare le interfacce e la implementazioni richiede:

- **definire un componenti tramite la specifica della sua interfaccia pubblica**
- **Separare la specifica dell'interfaccia dai dettagli di come il componente è implementato** (incapsulamento ed information hiding)

Sufficienza, completezza, primitività.

Sufficienza e completezza = i componenti SW catturano tutte e sole le caratteristiche rilevanti di una astrazione

Primitività = il design deve essere basato su **pattern facili da implementare**

Separazione delle problematiche (concerns).

"Un concern è un'area di interesse rispetto al SW design"

Un **design concern** è un **ambito di design rilevante** per uno o più degli stakeholder

Ogni **vista architetturale inquadra** uno o più **design concerns**.

Separare i concerns tramite le viste architetturali permette agli stakeholder interessati di focalizzarsi su un insieme ristretto di problematiche alla volta ed è uno strumento di gestione della complessità

SWEBOK

«**Key Issues in Software Design**
… issues that "tend not to be units of software's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways"

- **Control and Handling of Events**
- **Data Persistence**
- **Distribution of Components**
- **Error and Exception Handling and Fault Tolerance**
- **Interaction and Presentation**
- **Security**

Concorrenza

Riguarda:

- **Scomposizione** del software in **processi, tasks, e threads**
- Gestione dei problemi correlati:
  - **Efficienza**
  - **Atomicità**
  - **Sincronizzazione**
  - **Scheduling**

Controllo e gestione degli eventi

Come:

- Organizzare il **flusso dei dati e dei controlli**
- Gestire **eventi reattivi e temporali**

Persistenza dei Dati

Come gestire i **dati persistenti**.

Distribuzione dei componenti

Come:

- **distribuire il SW sui componenti Hardware** (computer hardware, network hardware, etc.)
- Gestire la **comunicazione tra i componenti**
- Utilizzare il **middleware** per gestire **software eterogeneo**

Gestione degli errori e delle eccezioni e tolleranza agli errori

Come:

- **Prevenire, tollerare e gestire gli errori**
- Gestire le **condizioni eccezionali**

Interazione e Presentazione

- Come strutturare e organizzare:
  - **Interazione con gli utenti**
  - **Presentazione delle informazioni** (es. Separazione tra il livello di presentazione e la business logic – pattern Model-View-Controller)

Security

Design per la security:

- **Minimizzare il rischio di:**
  - **Divulgazione, creazione, modifica, cancellazione non autorizzata**
  - **Negazione dell'accesso all'informazione o altre risorse**
- **Gestire la tolleranza ad attacchi o violazioni:**
  - **Limitazione del danno**
  - **Continuazione del servizio**
  - **Rapidità di riparazione e recupero**
  - **Sicurezza nel recupero**

Safety

Sistemi Safety critical → sistemi in cui un **errore di sistema** può **nuocere** alla **vita umana**, ad **altri esseri viventi**, a **strutture fisiche all'ambiente**
Es. Sistemi di trasporto, dispositivi medicali, impianti chimici, systemi avionici, etc.

In questi sistemi, **il SW** è esso stesso **safety critical**

Design per la safety **=** Disegnare per **minimizzare il rischio**

- **Pericoli per la safety** (probabilità, impatto, rischio)
- **Difetti SW**
- **Errori umani**
- Altri: pianificazione, costi , tempi, etc.

Safety

Riduzione del rischio di safety:

- **Riduzione della complessità del SW e delle interfacce**
- Progettazione **orientata alla safety**, anche se meno user-friendly
- Valutazione attenta del **fattore umano**
- Progettazione in **ottica di testabilità** in fase di sviluppo e integrazione
- **Aumento** della **disponibilità di risorse** agli aspetti a **più alto rischio**
- Definizione della **strategia** per ottenere il livello **di fault tolerance** richiesto nelle diverse componenti del systema

Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

SOFTWARE ARCHITECTURE METHODS

a-key

SWEBOK

Software Design Strategies and Methods

Esempi di strategie generali utili nel processo di progettazione:

- Function-Oriented (Structured) Design
- Object-Oriented Design
- Data Structure-Centered Design
- Component-Based Design (CBD)

SWEBOK

Function-Oriented (Structured) Design

La scomposizione è centrata sulla **identificazione delle funzioni SW principali** e sulla loro elaborazione e dettaglio in modo gerarchico.

Modello SW:

- sviluppato primariamente da un **punto di vista funzionale e comportamentale**
- **Parte** dalla **vista ad alto livello del SW** (funzioni, dati ed elementi di controllo)
- I **componenti** del modello sono **progressivamente scomposti** in modo gerarchico **top-down**
- Utilizzi principali:
  - generalmente **usato dopo una Analisi Strutturata** che abbia prodotto anche **diagrammi di flusso** dei dati e **descrizioni di processo**

SWEBOK

Object-Oriented Design

Un modello Object-Oriented è rappresentato da un **insieme di Oggetti** che **incapsulano dati e relazioni** e che **interagiscono** tra loro tramite **metodi**.

- Oggetti: rappresentano elementi reali o virtuali
- Modello SW: utilizza **diagrammi** che rappresentano **viste selezionate del SW**
- Raffinamenti progressivi → disegno di dettaglio → Implementation view (componenti e package)

SWEBOK

Data Structure-Centered Design

Parte dalla **struttura-dati del programma**:

1. **Descrizione delle strutture-dati in input e in output**
2. **Sviluppo della struttura di controllo del programma** basata sui diagrammi di struttura dei dati

Data Modeling Methods:

- Modello dei dati dal punto di vista dei dati utilizzati dal programma
- **Tabelle e relazioni**
- Utilizzi principali:
  - requisiti dei dati per la **progettazione di database o repositories**
  - Software per **applicazioni di business** → i dati sono un risorsa di business

SWEBOK

Component-Based Design (CBD)

Componente SW = unità indipendente, che ha interfacce e dipendenze ben definite che possono essere composte e distribuite indipendentemente.

- **Providing**
- **Developing**
- **Integrating**

} **Components** → **Improve reuse**

Gestione della affidabilità : componenti che hanno un certo livello di affidabilità **non devono dipendere** da componenti o servizi **meno affidabili**

Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

SOFTWARE ARCHITECTURE DOCUMENTATION

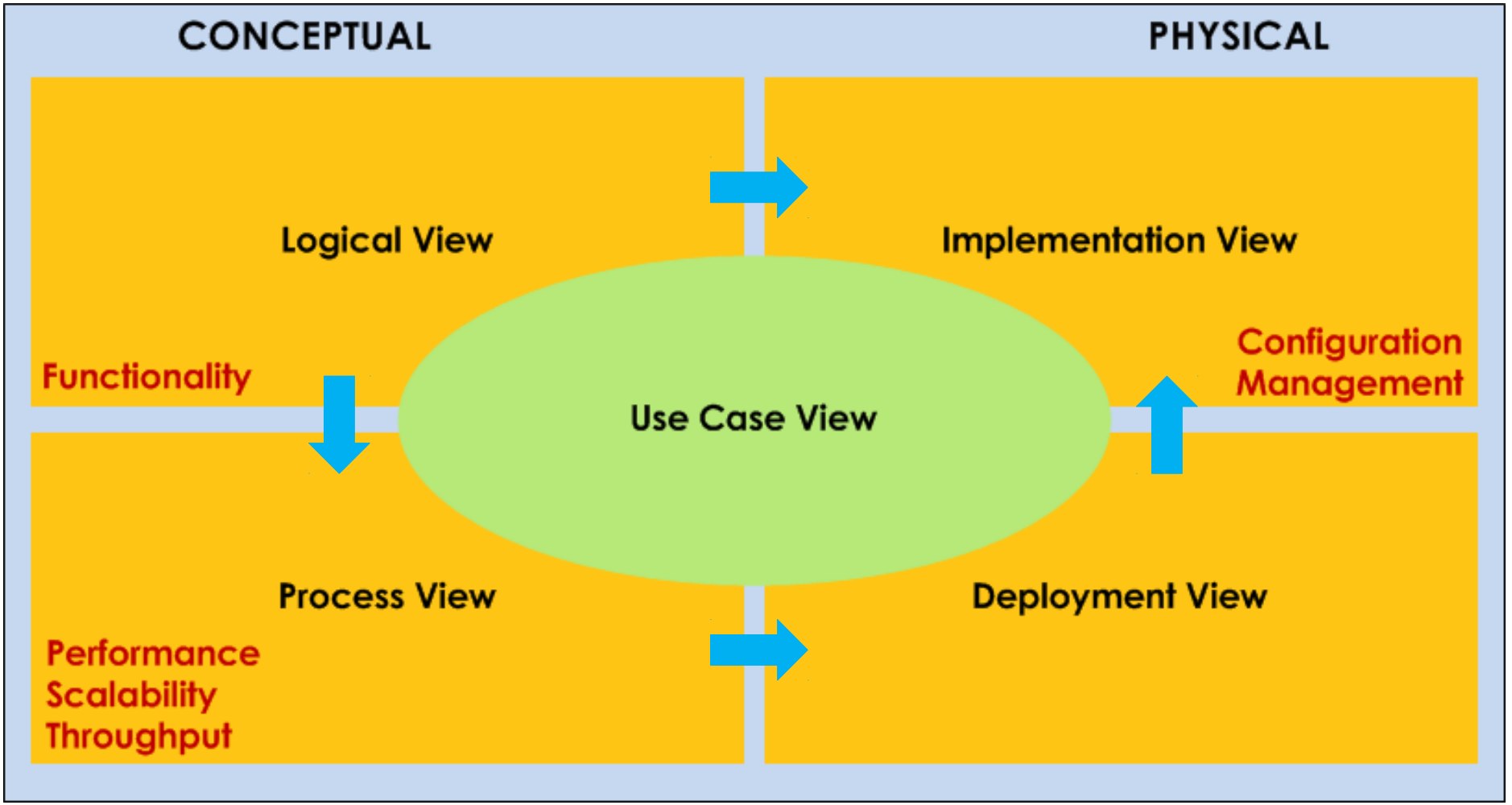
a-key

La progettazione Architetturale richiede una descrizione tramite **diverse prospettive** perchè:

- Il comportamento a runtime e la struttura logica sono diversi tra loro
- La progettazione architetturale deve fornire i constraints per la progettazione a più basso livello
- L'architettura deve essere considerata nella organizzazione della attività di sviluppo
- Diverse viste architetturali facilitano la comunicazione tra i teams

Modello di viste architetturali (Kruchten – 4+1)

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



View	Description	UML Diagrams
Logical view	is concerned with the **functionality** that the system provides **to end-users** Describes **what the system should provide** in terms of services to its users to satisfy software requirements. The focus is on **decomposition in elements** like **objects or objects classes** to represent the **logical architecture** of the software system	**Class diagram** **Communication diagram** **Sequence diagram**

View	Description	UML Diagrams
Implementation view	illustrates a system from a **programmer's perspective** and is concerned with **software management**. Describes the **components** used to **assemble and release** a physical system. This view **focuses on configuration management and actual software module organization** in the development environment.	**Component diagram** **Class diagram**

View	Description	UML Diagrams
Process view	deals with the **dynamic aspects** of the system, explains the **system processes** and how they **communicate**, and focuses on the **runtime behavior** of the system. The process view addresses **non functional aspects** of the software system: ○ **concurrency,** ○ **distribution,** ○ **integrators,** ○ **performance,** ○ **scalability** ○ **etc.**	**Activity diagram**

View	Description	UML Diagrams
Deployment view	depicts the system from a **system engineer's point of view.** Documents what parts of the system execute where, describing how the software executes on a network of processing nodes. These physical configurations can **differ** between **production, development and testing environments** It is concerned with the **topology of software components on the physical layer**, as well as the **physical connections** between these components.	**Deployment diagram**

View	Description	UML Diagrams
Use Case View	The Use Case View (the +1 view) **unifies the four primary views** describing **functional requirements** and the **behavior** of the system as seen by its end users and other stakeholders. The scenarios describe **sequences of interactions** between objects, and between processes. This is usually **the first view created** in the software development lifecycle. Scenarios are used to **identify architectural elements** and to **illustrate and validate the architecture design**. They also serve as a **starting point for tests** of an architecture prototype.	**Use case diagram**

Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

SOFTWARE ARCHITECTURE EXAMPLE
AUTOSAR

a-key

AUTOSAR

" **AUTOSAR** (**AUTomotive Open System ARchitecture**) is a **worldwide development partnership of vehicle manufacturers, suppliers and other companies** from the **electronics, semiconductor and software industry**. "

http://www.autosar.org/

Open Model for Development of architectures for automotive systems:

- to create a **development base for industry collaboration** on **basic functions**
- to provide a **platform** which continues to **encourage competition**

AUTOSAR Objectives

- **Standardization of basic software functionality** of **automotive ECUs**
- **Scalability** to different vehicle and platform variants
- **Transferability** of software
- Support of **different functional domains**
- Definition of an **open architecture**
- **Collaboration** between **various partners**
- Development of **highly dependable systems**
- Support of applicable **automotive international standards** and

AUTOSAR

The AUTOSAR scope includes **all vehicle domains**.

The AUTOSAR standard will serve as a **platform** upon which **future vehicle applications** will be implemented and will also serve to **minimize** the current **barriers between functional domains**.

It will, therefore, be possible to map functions and functional networks to different control nodes in the system, **almost independently from the associated hardware.**

Example of **Requirement Management** and documentation :

→ **AUTOSAR Main Requirements document** :

- **Overall project objectives** (PO) of AUTOSAR and their refinement
- AUTOSAR **Main Requirements** including its **link to** the AUTOSAR **objectives**
- AUTOSAR **Main Requirements** as fundamental **base to derive specific requirements**
- Requirements **writing rules**

(AUTOSAR_RS_Main.pdf)

Example of **Layered Software Architecture** and documentation:

→ **AUTOSAR Layered Software Architecture documentation** :

- **top-down approach**
- **hierarchical structure** of AUTOSAR software
- **Mapping** of the Basic Software Modules **to software layers** and showing of their **relationship**
- Architecture **description views**
- Application of concepts: **Modularity, abstraction, incapsulation, communication, dependancies**

(AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf)

Master Sistemi Embedded per Internet of Things

CORSO «ORGANIZZAZIONE, PROCESSI E CERTIFICAZIONI»

SOFTWARE ARCHITECTURE QUALITY AND EVALUATION

a-key

SWEBOK

"Quality Analysis and Evaluation Techniques

- **Software design reviews**: informal and formalized techniques to determine the quality of design artifacts (for example, architecture reviews, design reviews, and inspections; scenario-based techniques; requirements tracing)

- **Static analysis**: formal or semiformal static (**non-executable**) analysis that can be used to evaluate a design (for example, fault-tree analysis or automated cross-checking).
  - uses **mathematical models** that allow designers to **predicate the behavior** and **validate the performance** of the software **instead of having to rely entirely on testing**.
  - can be used to **detect residual specification and design errors**.

- **Simulation and prototyping**: dynamic techniques to evaluate a design (for example, performance simulation or feasibility prototypes).

Qualunque **sistema** sufficientemente **complesso** è **soggetto** ad

errore in **seguito** all'**errore** di uno o più **dei suoi sottosistemi**

Fault-tree analysis

- mappa le relazioni tra **errori, sottosistemi ed elementi di progettazione** creando un diagramma logico dell'intero sistema. Il risultato indesiderato è portato alla radice di un albero logico
- **Ragionamento deduttivo**, analizzare gli effetti di errori ed eventi sul sistema complesso

- **Steps**:
  - **Definizione** dell'evento indesiderato
  - **Comprensione e analisi del sistema:** studiare le cause con probabilità di occorrenza >0 e gli effetti sul sistema
  - **Costruzione dell'albero degli errori**: basato su porte AND e OR
  - **Valutazione dell'albero degli errori**: analisi del rischio e dei possibili miglioramenti
  - **Controllo** i pericoli e i rischi identificati

Event tree analysis

- Un albero degli eventi parte da un **iniziatore** indesiderato (**failure**) e segue i **possibili eventi successivi** sul sistema attraverso **una serie di conseguenze**.
- All'aggiunta di un **nuovo evento** viene aggiunto un **nuovo nodo** sull'albero con una **suddivisione di probabilità tra ciascun ramo**.
- Si può quindi valutare la **probabilità degli eventi finali** derivanti dall'evento iniziale

Failure mode and effects analysis (FMEA)

- **Revisiona** componenti, assemblati, sottosistemi per identificare le **potenziali modalità di fallimento** e le loro **cause** ed **effetti**.
- Per ogni componente si registrano modalità di fallimento e gli effetti risultanti sul resto del sistema in un **FMEA worksheet**.
- Può essere una analisi qualitativa o quantitativa.
- **Ragionamento induttivo**, basato sulla logica e sulla esperienza
- Essenziale nella analisi di **affidabilità, sicurezza e qualità**

Tipi di analisi FMEA:

- **Functional** FMEA
- **Design** FMEA
- **Process** FMEA (manufactoring & assembly)

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

Processes

SYS.3 System Architectural Design

"The purpose of the System Architectural Design Process is to establish a system architectural design and identify which system requirements are to be allocated to which elements of the system, and to evaluate the system architectural design against defined criteria. "

SWE.2 Software Architectural Design

"The purpose of the Software Architectural Design Process is to establish an architectural design and to identify which software requirements are to be allocated to which elements of the software, and to evaluate the software architectural design against defined criteria."

System Architectural Design Process
Process outcomes

1) a **system architectural design is defined** that identifies the **elements of the system**;

2) the **system requirements** are **allocated to the elements of the system**;

3) the **interfaces of each system element are defined**;

4) the **dynamic behavior objectives of the system elements** are defined;

5) **consistency and bidirectional traceability** are established between **system requirements and system architectural design**;

6) the **system architectural design is agreed and communicated to all affected parties**.

Software Architectural Design Process
Process outcomes

1) a **software architectural design is defined** that **identifies the elements of the software;**
2) the **software requirements** are **allocated to the elements** of the software;
3) the **interfaces of each software element are defined**;
4) the **dynamic behavior and resource consumption objectives** of the **software elements** are **defined**;
5) **consistency and bidirectional traceability** are established between **software requirements and software architectural design**;
6) **the software architectural design is agreed and communicated to all affected parties**.

Base Practices from standard

Automotive SPICE – Traceability and Consistency