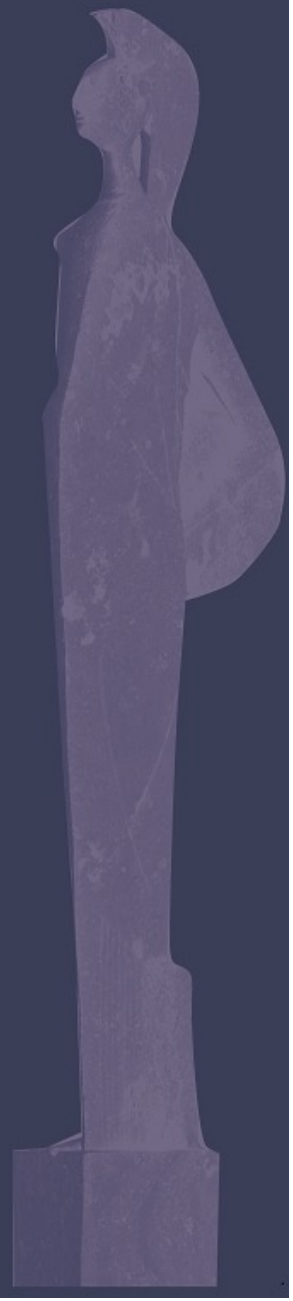
Università degli Studi di Trieste

Corso di Laurea Magistrale in
INGEGNERIA CLINICA

MODELLAZIONE DI PROCESSI E SOFTWARE

Corso di Informatica Medica

Docente Sara Renata Francesca MARCEGLIA



Dipartimento di Ingegneria e Architettura

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

IL SOFTWARE: DEFINIZIONI

1. L'insieme o una parte dei programmi, delle procedure, delle regole e della relativa documentazione di un **sistema di elaborazione dell'informazione.**

2. Programmi, procedure, ed eventuale documentazione e dati relativi all'operazione di un **sistema di calcolo**.

3. Programma o insieme di programmi usati per **l'operazione di un calcolatore**.

Systems and software engineering – Vocabulary

INGEGNERIA DEL SOFTWARE: DEFINIZIONI

- L'applicazione di conoscenze scientifiche e tecnologiche, metodi ed esperienza al progetto, l'implementazione, il collaudo e la documentazione del software.

Systems and software engineering – Vocabulary

- L'applicazione di un approccio sistematico, disciplinato, quantificabile, allo sviluppo, all'operazione ed alla manutenzione del software

IEEE Standard 610-12-1990

- Disciplina tecnologica e manageriale che riguarda la produzione sistematica e la manutenzione dei prodotti software,. . . sviluppati e modificati entro i tempi e i costi preventivati"

R. Fairley. Software Engineering Concepts. McGraw-Hill, 1985.

LA STORIA

- Nascita dei primi computer (anni ‘50)→
  - Problema della scrittura del codice
  - L’utente è sempre uno sviluppatore
- Fine anni ’50 – Inizio anni ’60 →
  - Inizia lo sviluppo dei linguaggi di alto livello
  - Essere programmatore diventa una professione
  - Pochissimi progetti software complessi
- Seconda metà anni ’60 →
  - Primi tentativi di grandi progetti software (sistemi operativi, e.g. IBM 360)
  - Si evidenziano le problematiche di gestione di questi grossi progetti → “crisi del software”→ troppo budget, ritardi di consegna
  - Coniato il termine “ingegneria del software” → ci si accorge che il metodo di implementazione di un prodotto software complesso doveva essere analogo a quello degli altri prodotti dell’ingegneria

LE MOTIVAZIONI

- La creazione di software è un'attività ancora giovane che si è sviluppata velocemente → necessita di una sistematizzazione efficace

- Progettisti e committenti non hanno le stesse competenze →
  - Difficoltà di comprensione
  - Difficoltà a stabilire i requisiti
  - Difficoltà a fornire i feedback

- È necessario creare linguaggi comuni tra analisti, utenti e sviluppatori

- È importante riuscire a definire i rischi economici e umani connessi all'uso del software → l'ingegneria del software studia anche gli aspetti sociali ed organizzativi sia dell'ambiente in cui viene sviluppato il software, sia di quello in cui il software viene applicato

PROGRAMMAZIONE vs INGEGNERIA DEL SOFTWARE

PROGRAMMAZIONE

- Soluzione stand alone
- Programma completo
- Monosviluppatore
- L'attività principale è lo sviluppo
- Competenze: linguaggi e programmazione

INGEGNERIA DEL SOFTWARE

- Sistemi complessi
- Componenti e/o moduli
- Multisviluppatore
- L'attività principale è la progettazione
- Competenze: approcci di progetto, gestione dei processi, definizione delle specifiche, comunicazione e interazione (con il committente/utente e con lo sviluppatore)

PARTICOLARITÀ DEL SOFTWARE

- Immateriale, non occupa spazio fisico (soft) →
  - Gli aspetti di "materiale di produzione" sono pressochè nulli
  - È difficilmente rappresentabile in assenza di notazioni condivise
  - È difficile definirne gli aspetti di qualità
- Human intensive → per essere costruito richiede molto lavoro umano
- Duttile → modificabile "con poco sforzo"
- Complesso → molte istruzioni che intergiscono tra di loro
- Non lineare → una piccola modifica in un punto può determinare un cambiamento radicale del comportamento dell'intero sistema

QUALITÀ DEL SOFTWARE: CLASSIFICAZIONI (1)

QUALITÀ INTERNE

- Relative alla stesura del codice
- Percepibili dagli esperti di programmazione (sviluppatori)

QUALITÀ ESTERNE

- Relative al prodotto finale (interfaccia esterna)
- Percepibili dagli utenti

Strettamente connesse tra di loro e interdipendenti

QUALITÀ DEL SOFTWARE: CLASSIFICAZIONI (2)

QUALITÀ DI PROCESSO

- Relative al processo di sviluppo (documentazione del processo)
- Percepibili dagli esperti di ingegneria del software

QUALITÀ DI PRODOTTO

- Relative all'artefatto (anche prodotto intermedio)
- Percepibili dal committente

PROPRIETÀ DEL SOFTWARE

	INTERNO	ESTERNO	PRODOTTO	PROCESSO
CORRETTEZZA	X		X	X
AFFIDABILITÀ			X	
ROBUSTEZZA	X		X	
PRESTAZIONI			X	
SCALABILITÀ			X	
EFFICIENZA	X		X	X
USABILITY	X		X	
VERIFICABILITÀ	X		X	X
MANUTENIBILITÀ	X		X	
RIUSABILITÀ		X	X	
PORTABILITÀ		X	X	
COMPRENSIBILITÀ			X	X
INTEROPERABILITÀ			X	
PRODUTTIVITÀ				X
TEMPESTIVITÀ				X
TRASPARENZA		X	X	X

CORRETTEZZA

- Capacità del software di rispondere ai requisiti funzionali e non funzionali →

  a fronte di input corretti, il software produce il risultato atteso

- Dipende dal livello di definizione delle specifiche
- Le specifiche devono anche definire quando un input è corretto (*precondizioni*)
- Può essere valutata nella fase di test del prodotto → è uno degli obiettivi fondamentali della valutazione

AFFIDABILITÀ

- Probabilità che il software si comporti nel modo atteso in un certo intervallo di tempo
- Qualità "relativa" → se la conseguenza di un errore non è grave, anche un software non corretto può essere considerato affidabile
- Software corretti sono anche affidabili, non viceversa

- Molti software sono rilasciati con bug già noti

ROBUSTEZZA

- Capacità del software di funzionare anche in situazioni anomale non previste dai requisiti

- Caso classico→ errore dati in input
- Livelli di robustezza:
  - livello 0: anomalia non rilevata. Il software continua l'esecuzione e produce risultati errati/entra in cicli infiniti;
  - livello 1: anomalia rilevata. L'esecuzione del programma viene interrotta immediatamente;
  - livello 2: anomalia rilevata. Viene eseguito del codice opportuno per risolvere il problema/ presentare diagnostica.
- Il livello di robustezza dipende dalla capacità di prevedere e gestire le **eccezioni**.

ESEMPIO

Calcolare la data successiva ad una data fornita in input

Data in input

Se il giorno non è l'ultimo del mese → aggiungo un giorno e mantengo inalterato mese e anno

Se il giorno è l'ultimo del mese e il mese non è dicembre → output è il primo giorno del mese successivo

Se il mese è dicembre → il giorno successivo è il 1 gennaio dell'anno successivo

ESEMPIO

- **Il software così pensato è corretto?**

- **Il software così pensato è robusto?**

ESEMPIO

Data in input

Se la data non è valida → blocco l'esecuzione
(LIVELLO 1)

Se il giorno non è l'ultimo del mese → aggiungo un giorno e mantengo inalterato mese e anno

Se il giorno è l'ultimo del mese e il mese non è dicembre → output è il primo giorno del mese successivo

Se il mese è dicembre → il giorno successivo è il 1 gennaio dell'anno successivo

Data in input

Se la data non è valida → invio un messaggio all'utente "Attenzione, data non valida, reinserire la data" e faccio ripartire l'esecuzione
(LIVELLO 2)

Se il giorno non è l'ultimo del mese → aggiungo un giorno e mantengo inalterato mese e anno

Se il giorno è l'ultimo del mese e il mese non è dicembre → output è il primo giorno del mese successivo

Se il mese è dicembre → il giorno successivo è il 1 gennaio dell'anno successivo

SCALABILITÀ

- Capacità del software di crescere in relazione al livello di utilizzo

- È collegato alle prestazioni attese
- Esempio: un software nasce per gestire un traffico di 10 utenti/ora ma si presenta la necessità di gestire 50 utenti/ora

USABILITY

- Facilità di utilizzo e di apprendimento da parte dell'utente

- Basata sulla valutazione di
  - Interfaccia utente
  - Facilità di configurazione
  - Capacità di adattamento al running environment
- È una misura soggettiva
- Deve essere specificato il tipo di utente

VERIFICABILITÀ

- Presenza di strumenti che consentono un monitoraggio del software

- La verifica/monitoraggio può essere effettuata mediante:
  - Analisi formale
  - Analisi informale
  - Procedure di test

MANUTENIBILITÀ (1)

- Il software può essere corretto/migliorato dopo il rilascio → capacità di evoluzione e di riparazione

- In generale la manutenzione del software è di tipo migliorativo:
  - Includere caratteristiche nuove non previste inizialmente (manutenzione perfettiva)
  - Correggere errori di specifica (manutenzione correttiva)
  - Rendere il software compatibile con cambiamenti/evoluzioni dell'ambiente, ad es. nuove versioni di sistema operativo (manutenzione adattiva)

MANUTENIBILITÀ (2)

EVOLVIBILITÀ

- Il software deve essere duttile
- Il software deve poter fornire nuove funzioni/modificare vecchie funzioni → le modifiche vanno progettate e documentate
- All'aumentare del tempo dal rilascio la capacità di evoluzione richiede interventi via via più complessi
- Facilitata dalla modularizzazione

RIPARABILITÀ

- I difetti del software devono essere corretti con una quantità di lavoro ragionevole
- Le parti "soggette a usura" devono essere accessibili (RAS: repairability, availability, serviceability)
- È utile l'utilizzo di parti standard

RIUSABILITÀ

- Capacità del software di essere riutilizzato, anche in parte, per applicazioni diverse

- Ha diversi livelli di granularità
- Favorita dall'approccio modulare
- Applicabile anche a livello di processo (ad es. si possono riusare i requisiti e applicare nuove specifiche)

PORTABILITÀ

- Capacità del software di essere eseguito in ambienti diversi (hardware o software)

- Concetto di "*porting*" e software "*platform independent*"
- Facilitato dall'uso di componenti standard (ad es caratteri ASCII)

COMPRENSIBILITÀ

- Il software deve essere leggibile → devono essere esplicite le scelte progettuali fatte
- Favorisce le modifiche future → manutenibilità e evoluzione
- Facilitato da
  - Approccio modulare
  - Astrazione di funzioni
  - Utilizzo di commenti, nomi di variabili autoesplicative, utilizzo di convenzioni

INTEROPERABILITÀ

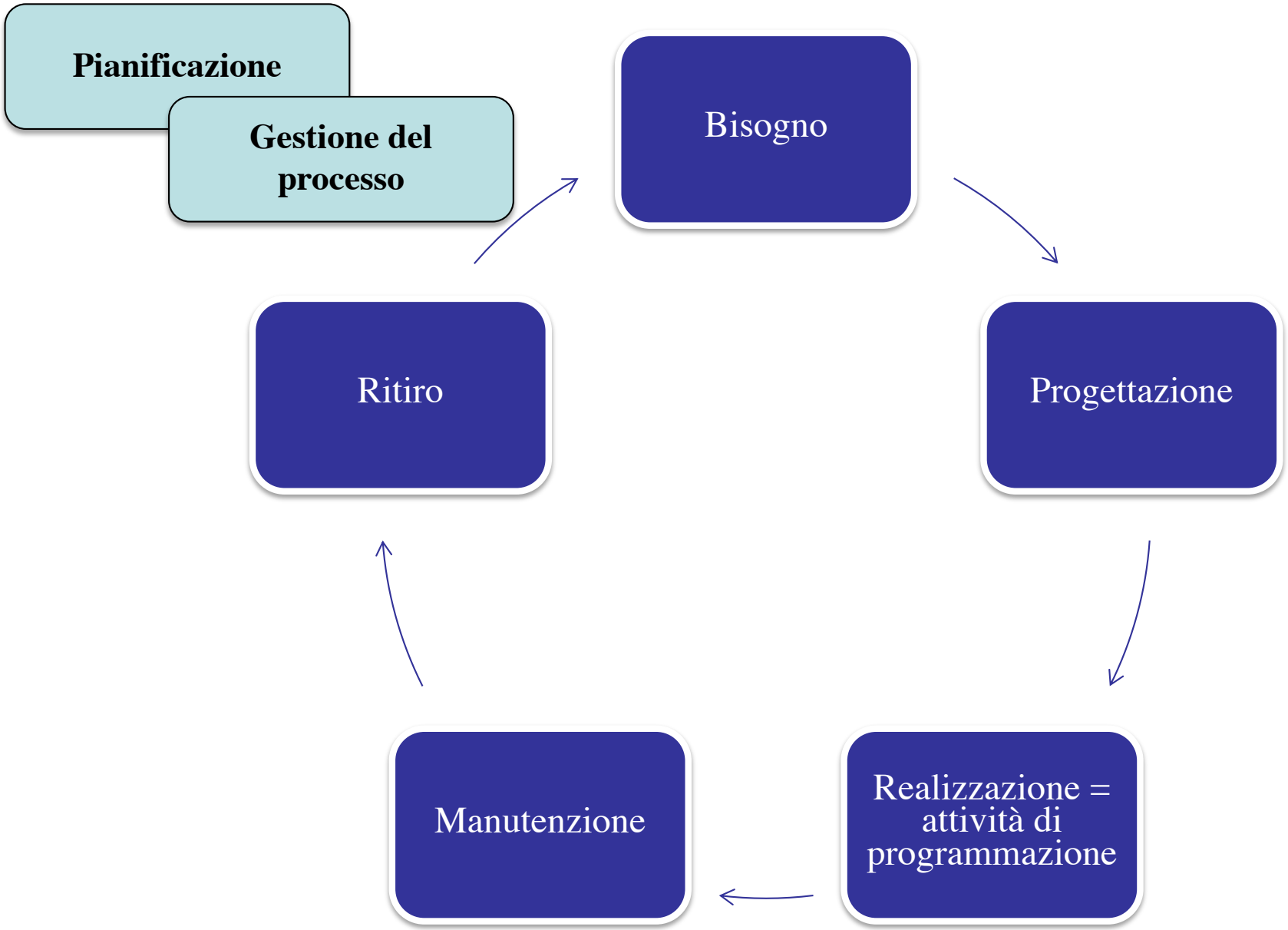
- Capacità del software di coesistere e cooperare con altri sistemi

- Facilitato dall'utilizzo di interfacce aperte
- Sistema aperto: collezione estendibile di applicazioni sviluppate in modo indipendente ma che funzionano come un sistema integrato

INTEROPERABILITÀ E SANITÀ

INTEROPERABILITY = Ability of different systems to work cooperatively allowing different users to share information and resources

LE MACROFASI DEL CICLO DI VITA



ATTIVITÀ DI PRODUZIONE

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

- **Studio di fattibilità**
- **Acquisizione, analisi e specifica dei requisiti**
- **Progettazione dell'architettura**
- **Codifica e test dei moduli**
- **Integrazione e test del sistema**
- **Rilascio, installazione e manutenzione**
- **Attività di supporto**

OGNI ATTIVITÀ PRODUCE UN OUTPUT

STUDIO DI FATTIBILITÀ

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

- Stabilisce se:
  - Il prodotto può essere realizzato
  - È conveniente realizzarlo
  - Quali strategie possono essere adottate per realizzarlo
  - Valutare risorse/costi delle diverse alternative
- 3 parti:
  - Definizione del problema e analisi di dominio quanto più approfondito possibile
  - Definizione degli scenari di soluzione
  - Modalità di sviluppo delle alternative, costi e date di consegna

OUTPUT =
Documento di studio di fattibilità
che deve contenere la descrizione delle 3 parti

REQUISITI E SPECIFICA DEI REQUISITI

- REQUISITI = definizione di COSA l'utente richiede al software (proprietà e comportamenti richiesti)
- SPECIFICA DEI REQUISITI = descrizione precisa dei requisiti →
  - Descrive una certa entità in termini di servizi forniti e proprietà da esibire (i.e., interfaccia)
  - Il grado di precisione richiesto per una specifica dipende in generale dallo scopo della specifica.
  - Descrive che cosa deve fare un sistema e quali proprietà deve avere
  - NON descrive come deve essere costruito

OUTPUT =
Documento di specifica dei requisiti
Versione preliminare del manuale utente
Piano di test

TIPOLOGIE DI REQUISITI

Requisiti dell'utente: descrizione del dominio applicativo e obiettivi dell'implementazione

- Quali utenti?
- Quali aspettative?
- Quali entità? Quali relazioni? (dati e relazioni tra i dati)
- Come interagiscono i dati col sistema? (controllo)

Requisiti funzionali

- Cosa farà il sistema?

Requisiti non funzionali

- Attributi di qualità (efficienza, scalabilità, etc)

Requisiti del processo di sviluppo e manutenzione

- Procedure di test e e verifica
- Priorità di sviluppo delle funzioni richieste
- Possibili cambiamenti che il sistema subirà

PROGETTAZIONE DELL'ARCHITETTURA

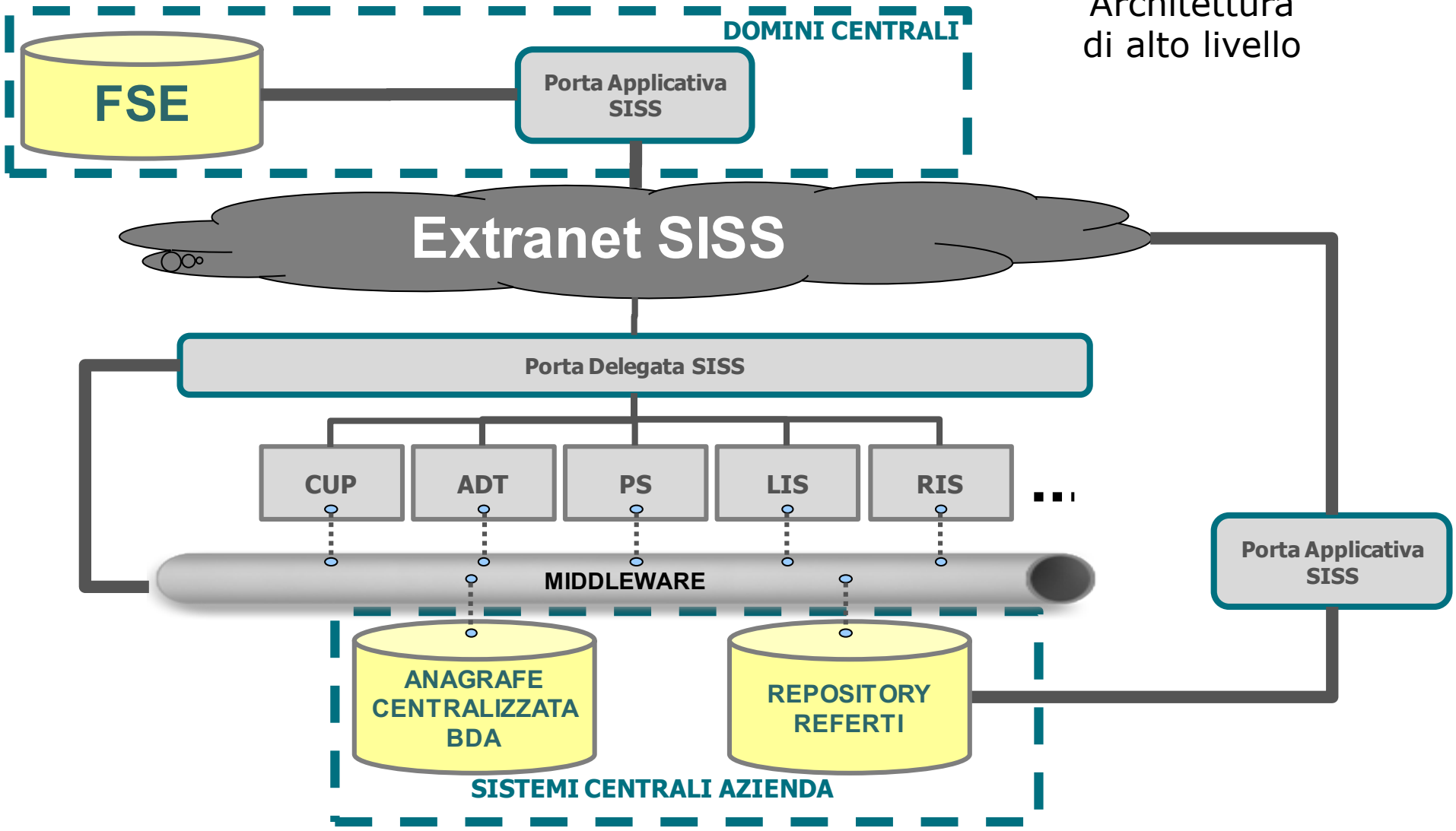
- Stabilisce **COME** deve essere fatto il sistema
- Costruzione di un modello = descrizione astratta di un sistema
- Descrive le componenti del sistema, le interfacce e le relazioni tra le parti
- Si parte da un'architettura di alto livello per poi scendere a livelli di dettaglio successivi
- Esistono linguaggi formali (ad es UML)

OUTPUT =
Documento di specifica del progetto

ESEMPIO: FSE - ARCHITETTURA DI ALTO LIVELLO

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Architettura
di alto livello



CODIFICA E TEST DEI MODULI

- Fase di sviluppo:
  - Trasparenza
  - Leggibilità (si possono seguire standard anche aziendali)
  - Modificabilità (è bene usare sistemi di versioning)
- Si effettua il debug
- Si effettua il test dei moduli
- Si può effettuare una verifica del codice per vedere se è conforme allo standard

OUTPUT =
Codice e documentazione del codice

INTEGRAZIONE E TEST DEL SISTEMA

- Attività in cui il sistema viene assemblato → a volte è inclusa nella precedente
- Vengono effettuati test di integrazione e test di sistema (alpha test)

OUTPUT =
Sistema completo per il rilascio

RILASCIO, INSTALLAZIONE E MANUTENZIONE

- Fase post-sviluppo
- Rilascio = consegna del prodotto
  - Ad un gruppo ristretto di utenti → beta test
  - A tutti gli utenti previsti
- Installazione = setup dell'architettura run-time
- Manutenzione (circa 60% del costo totale)
  - Correttiva
  - Adattiva
  - Perfettiva→ quella che si stima consumare le maggiori risorse (50% delle risorse di manutenzione)

ALTRE ATTIVITÀ DI SUPPORTO

Documentazione

- Di tutte le attività effettuate
- Per garantire la trasparenza

Verifica

- **Validazione (o convalida)** → valutazione della rispondenza (correttezza e qualità) del prodotto rispetto ai requisiti (informali)
- **Verifica** → valutazione della rispondenza alle specifiche (formali)
- Applicabile a tutte le attività che costituiscono il processo di sviluppo

Gestione

- Delle attività
- Delle responsabilità e delle risorse umane
- Del prodotto (nuove versioni, rilascio, etc)

MODELLI DI PROCESSO DI SVILUPPO

- Processo di sviluppo → modo di organizzare le attività del ciclo di vita
- Permette di assegnare risorse alle varie attività
- Permette di fissare le deadline
- FASE
  - intervallo di tempo in cui si svolgono certe attività
  - Ciascuna attività può essere ripartita fra più fasi.
- MODELLO DI PROCESSO → descrizione generica di una famiglia di processi simili, che realizzano il modello in modi diversi.

MODELLI DI PROCESSO

Modello Waterfall (a cascata)

- Modello lineare
- Modello con feedback
- Modello a V

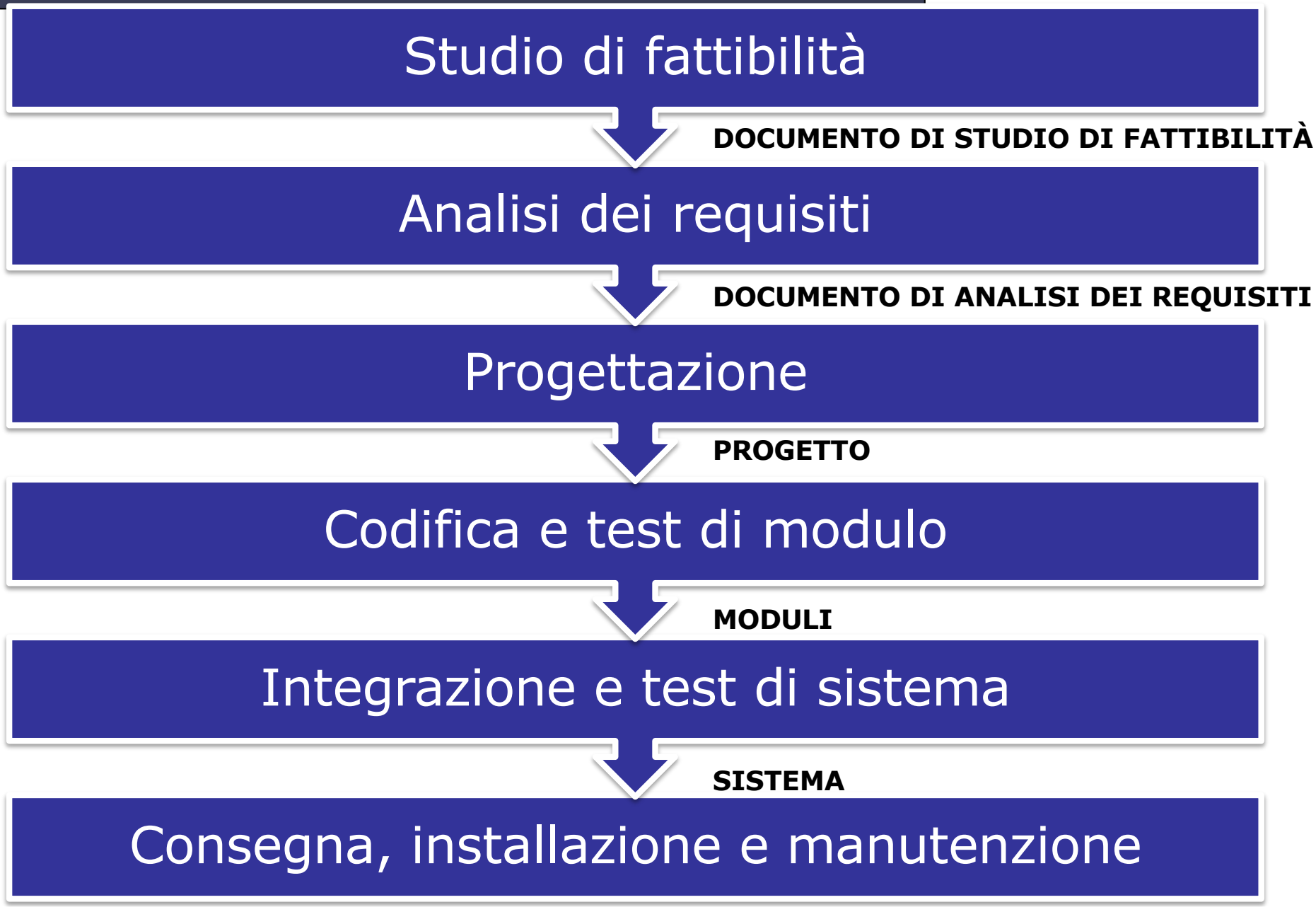
Modelli evolutivi

- Modelli basati su prototipi
- Unified process

Modelli trasformazionali

Modelli a spirale

MODELLO WATERFALL



CARATTERISTICHE

- Elaborato all'inizio degli anni '70
- È il modello di riferimento su cui si basano tutti gli altri modelli di processo
- Cascata lineare:
  - Ogni fase deve essere completata prima di passare alla fase successiva
  - l'output del passo precedente costituisce l'input del passo successivo
- Output = **DELIVERABLE**
- Tutte le fasi vengono completate avendo come target l'intero sistema
- Il modo in cui le fasi vengono affrontate dipende dal tipo di sistema

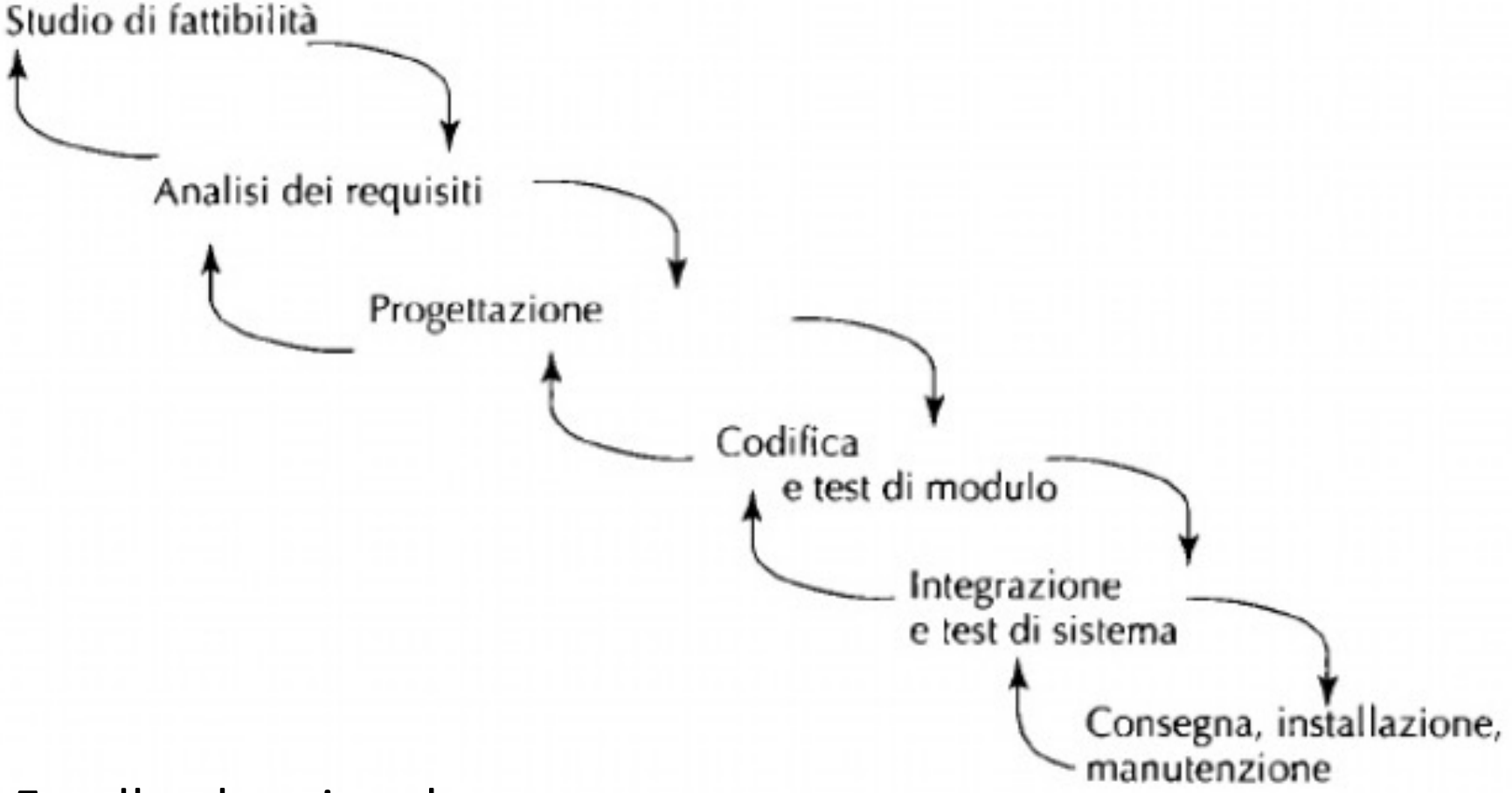
VANTAGGI

- Introduce una disciplina di progettazione dopo il sistema code&fix (→ programmo e correggo)
- Introduce i concetti di pianificazione e gestione
- Rimanda l'implementazione dopo che la fase di progettazione è stata completata

LIMITI

- Modello ideale
- Modello lineare→ non consente di avere feedback finché l'intero processo non è terminato
- Modello rigido→ i risultati restano "congelati" finché non si passa alla fase successiva
- Modello monolitico→
  - Orientato ad una singola fase di rilascio (del sistema completo)
  - La fase di rilascio può avvenire anche mesi/anni dopo la fase di progettazione
- Non permette visibilità del prodotto fino alla sua completa implementazione
- Difficile stimare a priori i costi e le risorse necessarie
- Il documento di specifica dei requisiti rappresenta un vincolo per l'intera progettazione e realizzazione → non tiene conto di possibili raffinamenti/variazioni da parte dell'utente
- Basato su documentazione (deliverables) → può portare ad uno stile di lavoro troppo burocratico
- Non risponde al principio di anticipazione del cambiamento

MODELLO WATERFALL CON FEEDBACK



- Feedback a singolo passo
- Migliorato rispetto al waterfall classico ma ancora limitato

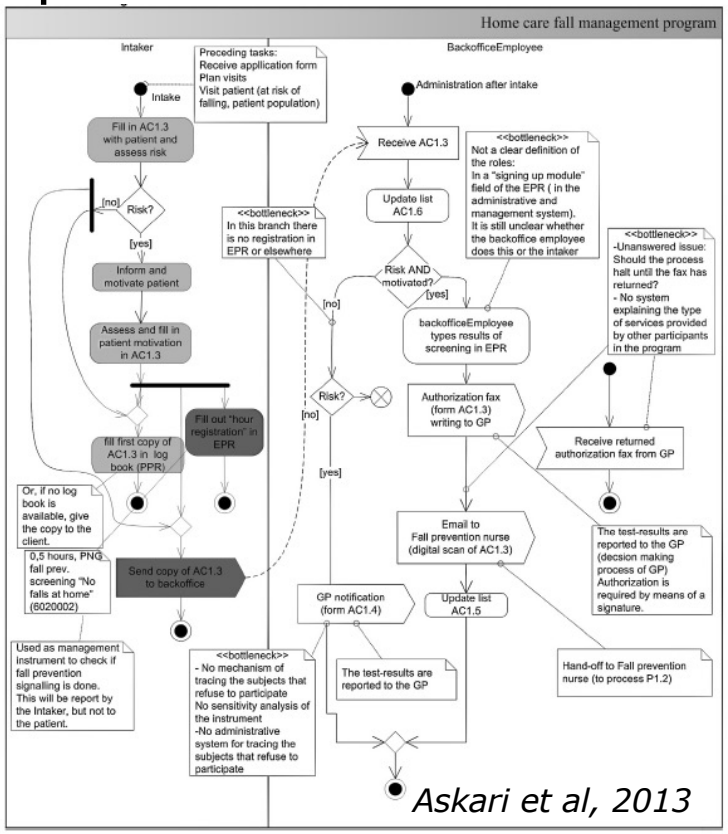
UNIVERSITÀ
DEGLI STUDI DI TRIESTE

PROGETTAZIONE DI SISTEMI DI INFORMATICA MEDICA E MODELLAZIONE DEI PROCESSI

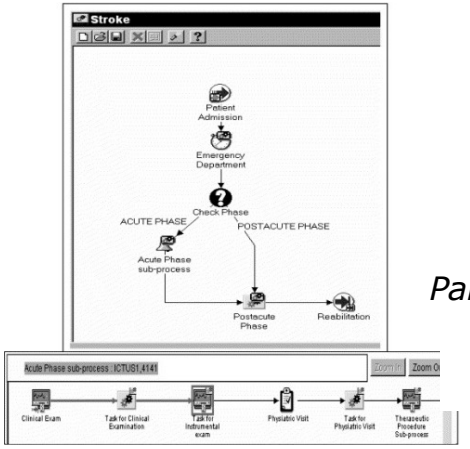
PROCESS MODELING

UNIVERSITÀ DEGLI STUDI DI TRIESTE

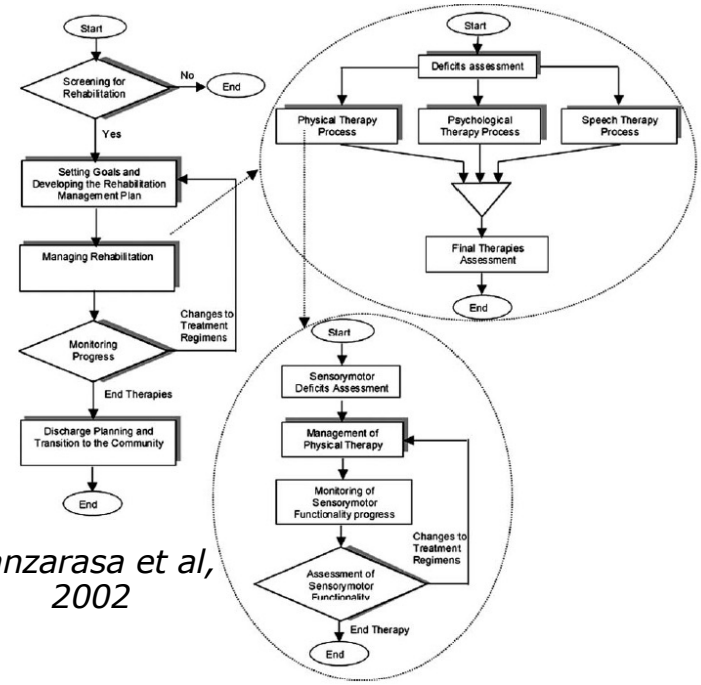
Model → abstraction used to represent and describe the process under examination



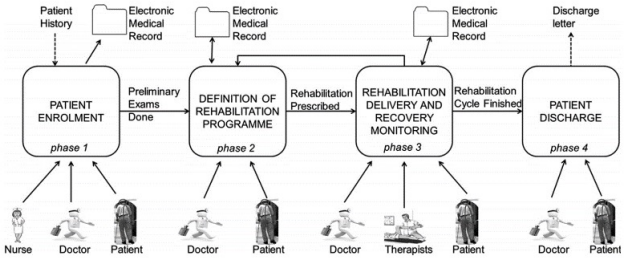
Askari et al, 2013



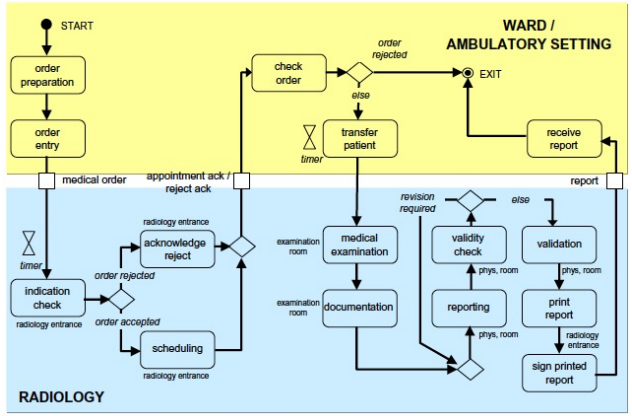
Quaglini et al, 2001



Panzarasa et al, 2002



Ferrante et al, 2013



Lenz & Reichert, 2007

HEALTHCARE PROCESSES

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Multiple stakeholders

Patient's response to treatment

Patient's engagement

Multiple systems

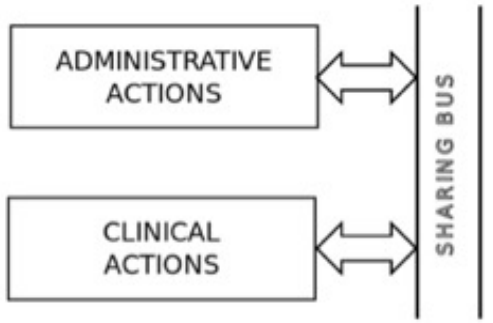
COMPLEXITY

Patient's empowerment

Evolving knowledge

Multiple action domains

Evidence based practices



EXPECTATIONS FROM HEALTHCARE PROCESS MODELING

CLINICAL EXPECTATIONS

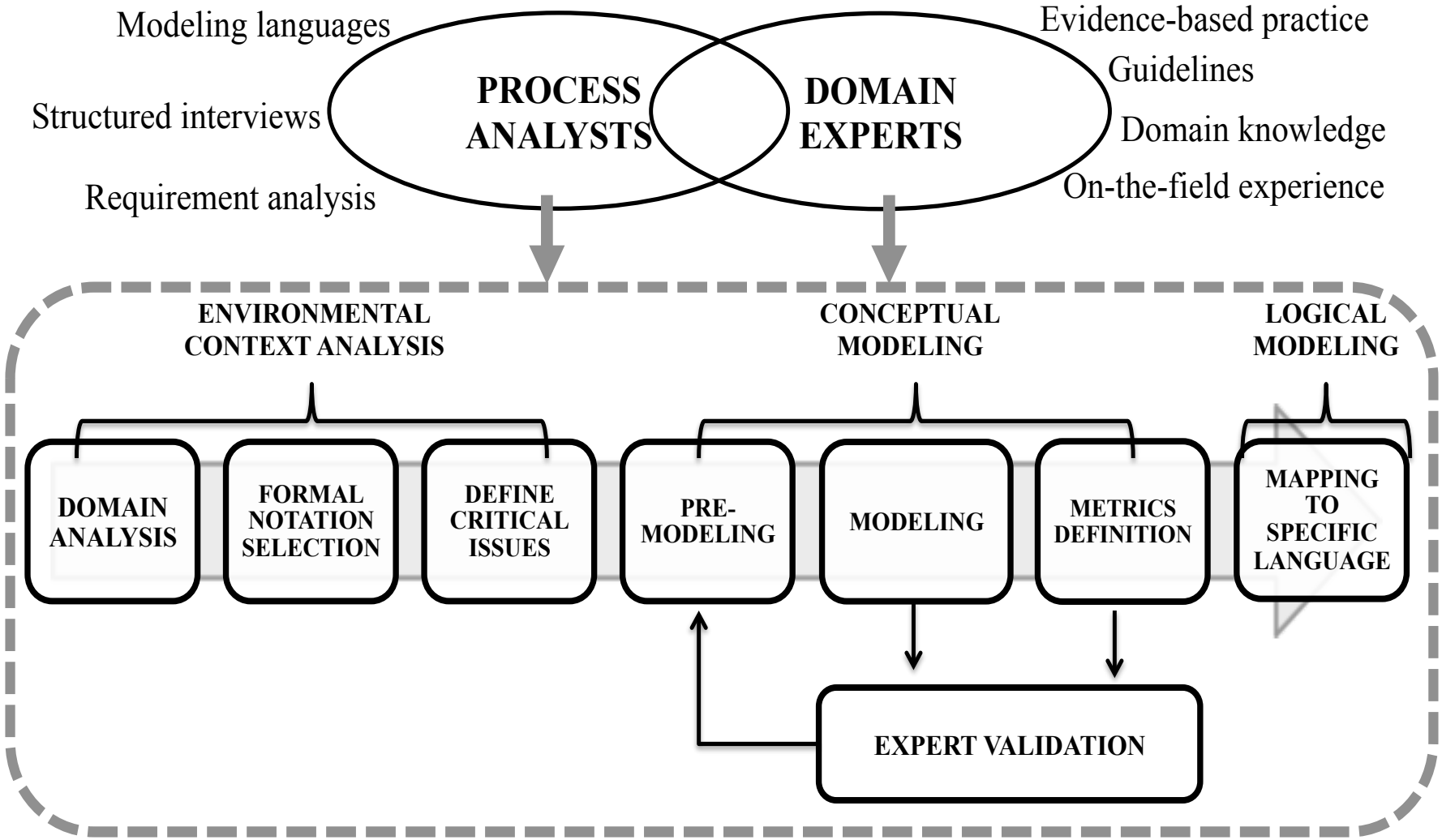
- Establishing shared protocols for patient's care.
- Facilitating adherence to the shared protocols, thus limiting problems due to incomplete communication or misunderstandings among different actors, ultimately increasing patient's safety.
- Monitoring deviances from the protocols, redundancies, and failures, thus early identifying problems that could lead to un-prevented errors.

ORGANIZATIONAL AND TECHNOLOGICAL EXPECTATIONS

- Fully understanding the information flow, thus identifying requirements and specifications for information system re-engineering and interoperability.
- Detecting process weaknesses thus designing corrective measures.
- Optimizing the use of resources.

MODELING METHODOLOGY OVERVIEW

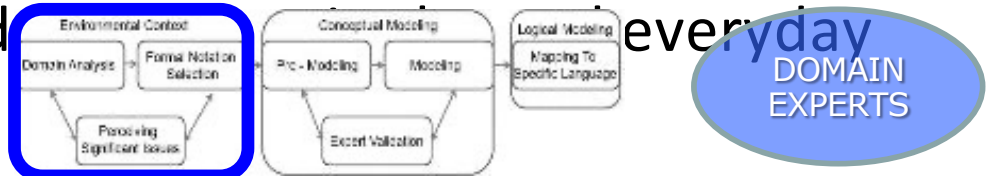
UNIVERSITÀ
DEGLI STUDI DI TRIESTE



DOMAIN ANALYSIS

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

- Identifying and analyzing the available sources of information to fully understand the domain of interest
- Evidence-based knowledge → international guidelines and recommendations
- Local domain →
  - local practices
  - specific clinical pathways already in use locally
  - focus groups and interviews to the medical staff and/or the patient/caregivers, to highlight the personal experience of the actors involved in the process
- Description of the information systems already in use → helps planning the model d… …everyday practice.



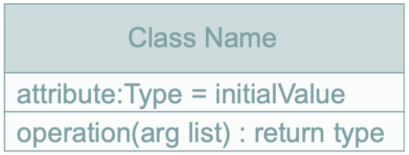
SELECTION OF THE FORMAL NOTATION: UML

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

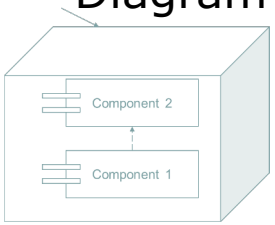
"The Unified Modeling Language (UML) is a graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems".

STATIC VIEWS

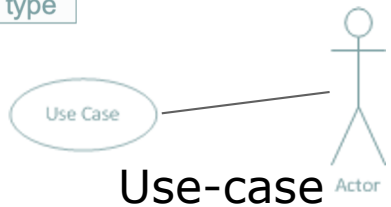
DYNAMIC VIEWS



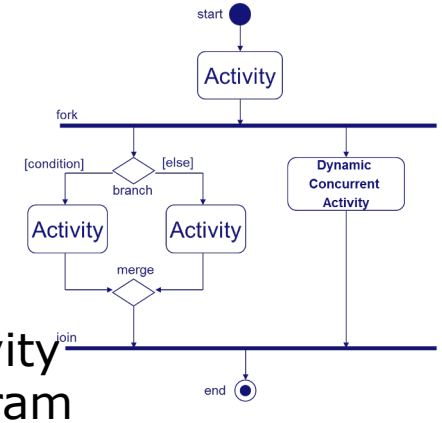
Class Diagram



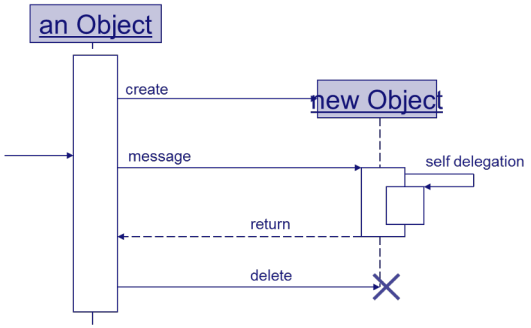
Component Diagram



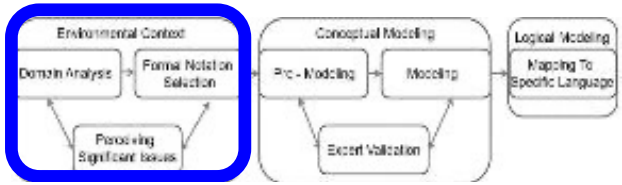
Use-case Diagram



Activity Diagram



Sequence Diagram



ANALYST

UML: ADVANTAGES

- Graphical language →
  - Enables the communication between domain experts and analysts
  - Easy to understand by non-experts of computer sciences
- Provides different views on the process→
  - Static and dynamic diagrams
  - Structural, Behavioural, and Interaction diagrams

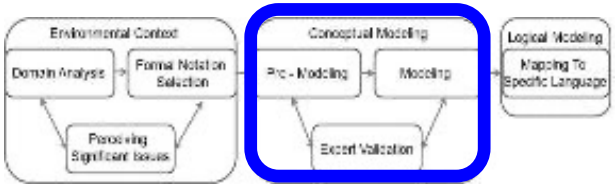
- The final UML model can be used as specification for the development of an IT system
- The use of UML promotes modularity and facilitates future changes
- It allows the evaluation of the whole system, also at the deployment level

PRE-MODELING

- Provide a high-level process description (process phases)
  - Functional aspects (main activities of the process, objects and data items managed)
  - Organizational aspects (agents, roles, skills, availabilities, authorizations required to enact the process) □ understand actors' responsibilities on the main activities business aspects (goals to be achieved).

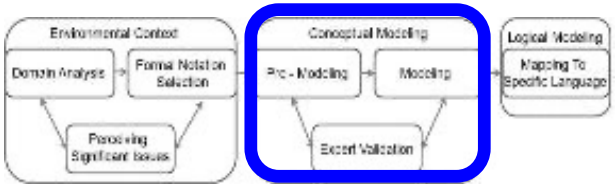
- Define a list of goals of the process modeling effort

- Identification of outcomes and integration with patient-reported outcome measures



MODELING

- The modeling step starts from the previously collected information and produces a conceptual model of the process according to the formal notation adopted
- The conceptual model comprises:
  - the schema of the process
  - its variables
  - the specification of the expected exceptions
  - the specification of the transactions
  - the access control model
  - the description of the interactions with the external information system.
- The results of the modeling are then validated by the domain experts, before continuing with the design



MODEL EVALUATION

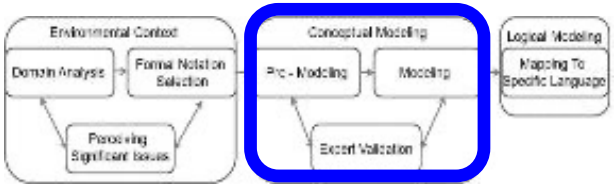
Assess if the model is "syntactically correct"

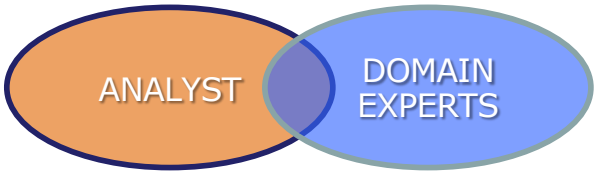
- Verify the internal consistency of the model.
- Assess its usability as a starting point for the logical modeling towards the software implementation.

Assess if the model is "semantically correct"

- Iterative approach based on focus groups or external validators.
- Analysis by actor type to validate the flow of information in the simplest activities of the process.
- Analysis with multiple actors to test the whole model.
- The model is final when a total agreement between experts and analysts is reached.

Askari et al. 2013





METRICS AND GQM

- METRIC = a quantitative measure of the degree to which a system, component, or process possesses a given attribute
- Metrics for the evaluation of health ITs cannot be directly derived from the model itself → the model can be the basis for identifying the outcome variables to be introduced into e-management techniques as metrics for evaluation.
- Goal Question Metrics (GQM) →
  - allows selecting metrics with a top-down and goal-oriented approach
  - the identification of the metrics starts from the definition of goals
  - The definition of the goals is done during the conceptual modeling phase
- GQM has three levels →
  - Goal: Conceptual level, defines the main purposes of a work to be measured;
  - Question: Operational level, defines a set of questions useful for achieving the goals;
  - Metric: Quantitative level, defines a set of metrics for answering the questions in a measurable way.

MODEL IMPLEMENTATION

- Mapping to a specific executable language
- Building a system implementing the model
- Create new modules of an already existing system to implement the process