



Famiglie di Prodotti Software

Piergiorgio Di Giacomo
Firenze, 09 giugno 2005

Contenuti della presentazione:

- Parte I:
 - Introduzione alle famiglie di prodotti nell'industria
 - Concetto di riutilizzo di componenti
 - Riuso non sistematico vs. Riuso sistematico
 - Fattori critici per la creazione di famiglie di prodotti software.
 - il modello BAPO e le sue 4 dimensioni
- Parte II:
 - Approccio classico al reuse
 - Approccio strutturato: Ingegneria del Dominio e delle applicazioni
 - Concetti base dell'ingegneria del dominio:
 - analisi,
 - modellazione e
 - implementazione
 - Il FODA:
 - Definizioni
 - Notazione grafica e suo significato
 - Gestione della variabilità
 - Limiti del FODA
 - UML come soluzione per superare i limiti del FODA
- PARTE III:
 - Caso di studio in Bosch

PARTE PRIMA

Forse non tutti sanno che...

- Alcuni tra i modelli di auto più vendute sul mercato condivide molte componenti:



Forse non tutti sanno che...



Le seguenti sigle indicano lo stesso motore:

- **1.9 TDI**
- **1.9 TDI**
- **1.9 TDI**
- **1.9 TDI**

Le sigle in realtà indicano che: la cilindrata e la maggior parte delle componenti non cambiano, eppure aggiornamenti sul software di controllo e altri accorgimenti lo rendono “riutilizzabile” in diversi contesti.

Forse non tutti sanno che...

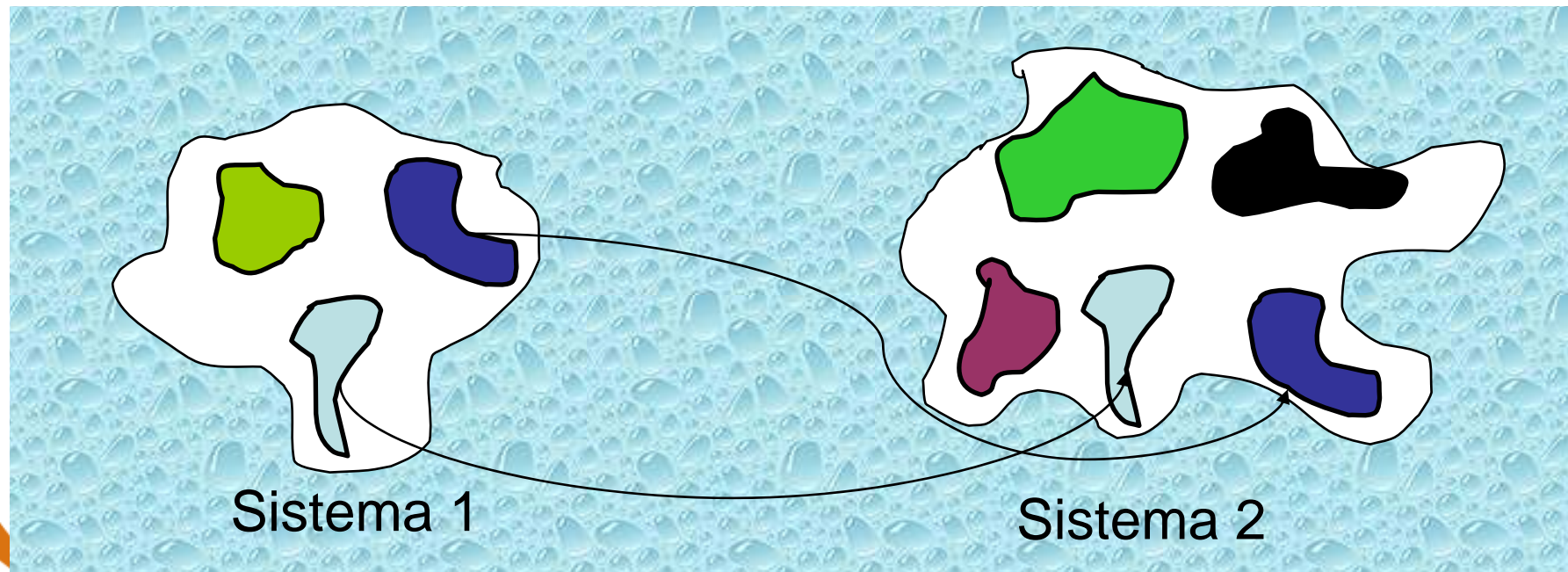
Nel 1993 a Bologna (e poi ne 1994 al salone di Ginevra) Giugiaro presentava la Fiat Lucciola, un possibile sostituto della Cinquecento.



Il prototipo, non molto diverso dalla attuale Matiz, sembra fu però scartato perchè le sue dimensioni non rendevano possibile il riutilizzo del pianale della Fiat Cinquecento. Il progetto fu accantonato e sostituito dalla attuale Seicento

Riuso del software

- Per riuso si intende l'uso di applicazioni software esistenti per costruire nuovi sistemi



Riuso sistematico: famiglie di prodotti

Linea guida:

*Prodotti ricchi di funzionalità, di
alta qualità che abbiano tempi
di sviluppo rapidi e costino
poco*

- Tre obiettivi conflittuali non possono essere raggiunti appieno nello stesso programma di sviluppo:
 - Portata e complessità del prodotto
 - Qualità
 - Costi & tempi di realizzazione
- => Separare i componenti riutilizzabili, organizzarli in piattaforme software e lasciare che siano queste a prendersi cura dei problemi a lungo termine

Fattori critici per il riuso del software

- Decisioni guidate dal mercato (non dalla tecnologia)
- La creazione di una famiglia di prodotti deve essere trattata come un investimento
- Definizione puntuale degli obiettivi e del dominio di appartenenza
- Pianificazione di un processo di misura delle prestazioni e di miglioramento
- Pianificazione dei cambi nel ciclo di vita dei singoli prodotti

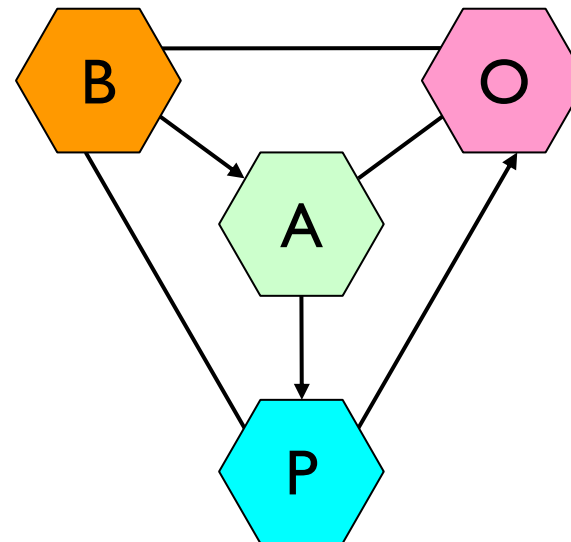
Definizione del dominio di appartenenza

- **Non sempre** il riuso del software porta vantaggi e quasi mai in ugual misura
- Il riuso richiede:
 - Una domanda potenziale per applicazioni simili in uno stesso dominio
 - Conoscenze avanzate ed esperienza nella produzione di sistemi software
 - Avere successo con tecniche di riuso richiede un certo **tempo di apprendimento**

Il modello BAPO

- Il successo di una famiglia di prodotti software dipende da quattro fattori:

- Business
- Organizzazione
- Architettura
- Processo



- Il controllo di questi quattro aspetti è essenziale per il successo

BAPO-B: Il riuso inteso come investimento

- Le iniziative volte al riuso del software richiedono una analisi **costi-benefici**
 - Return On Investment, Net Present Value, etc..
- Considerare I seguenti **fattori**:
 - Nuove opportunità di mercato
 - Tempo di arrivo sul mercato
 - Competitività dei prezzi
 - Tempo di vita dei singoli sistemi sul mercato
- I **Rischi** devono essere considerati e controllati



BAPO-A: gestione dell'architettura

- Apesti caratterizzanti:
 - Livello di riuso degli assets: fino a che punto si vogliono usare o riusare gli assets di un dominio in ciascun prodotto finale
 - Ingegnerizzazione di una architettura software: creazione di una architettura software comune e suo utilizzo
 - Gestione delle variabilità: l'uso esplicito di punti di variazione nella configurazione dei prodotti e stabilimento di meccanismi formali per la loro gestione (vedi PARTE II)

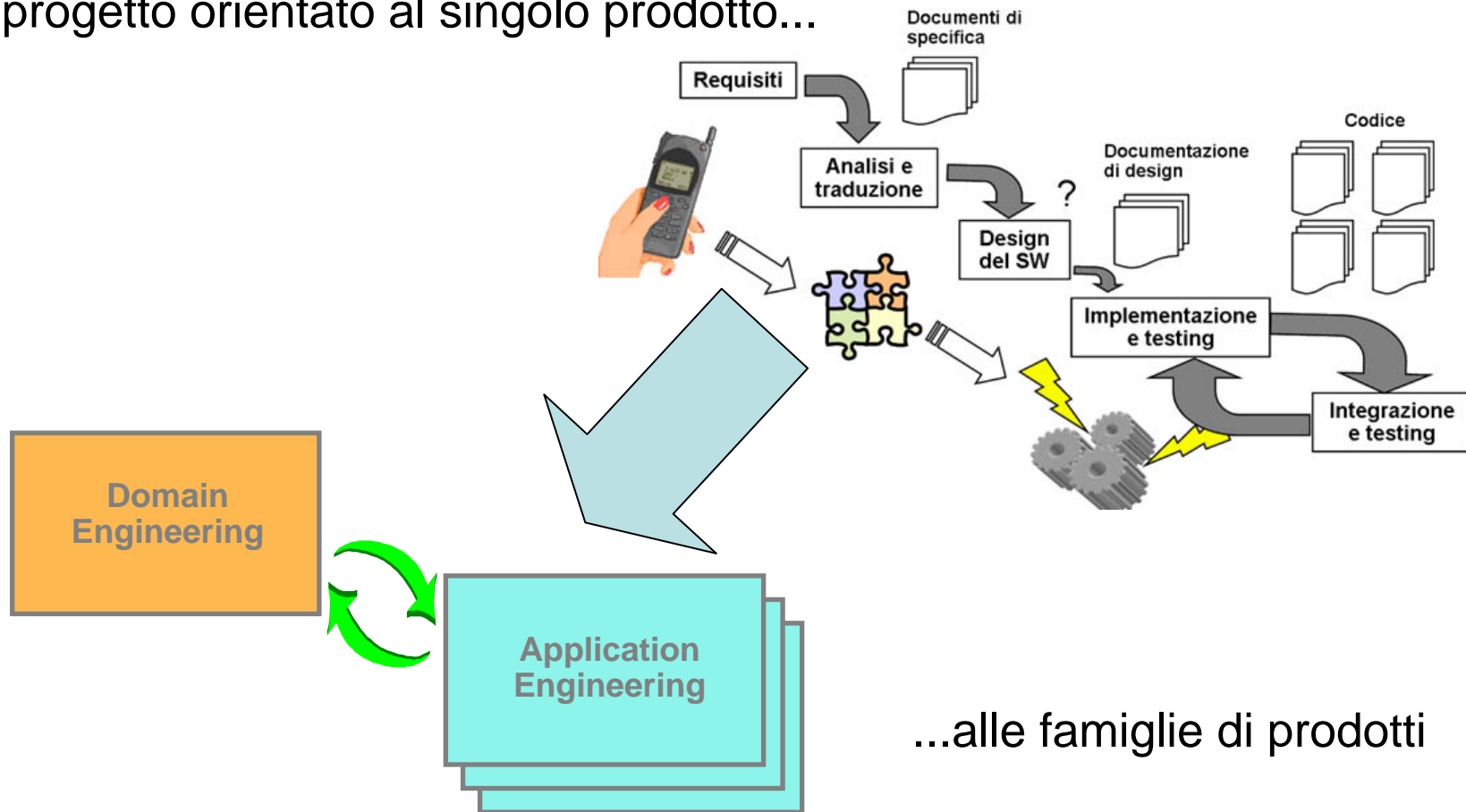
BAPO-P: Uso di processi formali

- Il riuso **cambia** in modo significativo il processo di sviluppo del software
- L'adozione di tecniche di riuso e l'impiantazione di un processo formale sono complementari l'una all'altra
- Per avere successo, il riuso sistematico richiede un certo livello di maturità nei processi di una organizzazione



Cambio nel ciclo di vita

Dal progetto orientato al singolo prodotto...



BAPO-O: definizione della infrastruttura di supporto

- Ruoli e responsabilità
 - I compiti di gestione della famiglia di prodotti devono essere assegnati in modo preciso. Deve esistere una struttura decisionale per il supporto della creazione, evoluzione e mantenimento della famiglia di prodotti software
- Struttura
 - Bisogna fornire una infrastruttura aziendale capace di mettere in pratica le decisioni prese in tempi brevi ed in modo efficiente
- Schemi di collaborazione
 - Bisogna far sì che il personale comprenda gli obiettivi da raggiungere e ne abbia una visione chiara e comune
 - Data la complessità dell'obiettivo finale, bisogna creare degli schemi di collaborazione tra i vari gruppi di lavoro.

PARTE SECONDA

Famiglie di prodotti software: approccio classico non sistematico

Copia e incolla

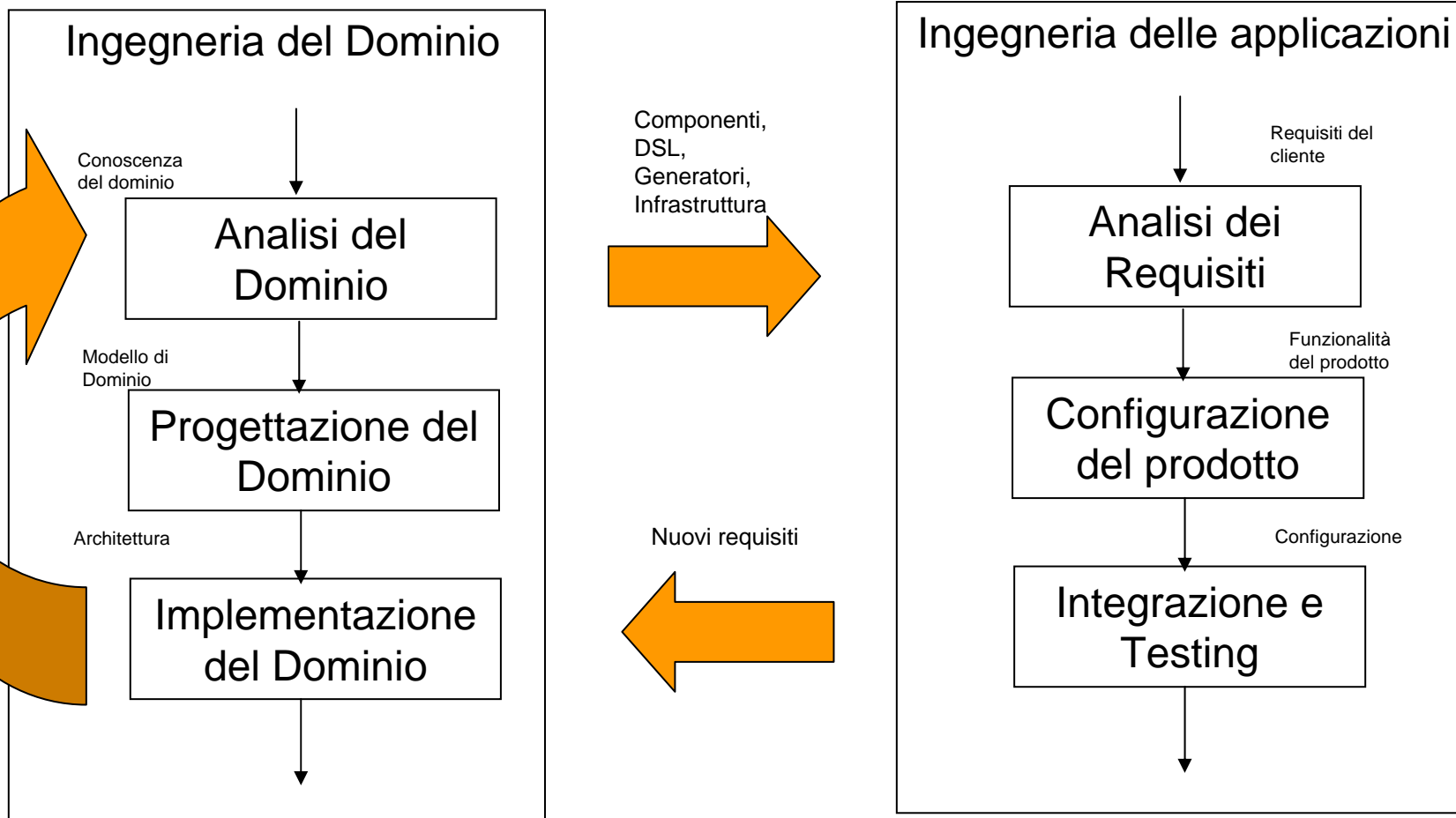


- **Velocità di derivazione e penetrazione del mercato**
- **Bassi costi di sviluppo**
- **Elevato numero di dispositivi**
- **Assenza di controllo**
- **Elevata ridondanza tra i prodotti**

Configurazione

- **Le variabilità sono inserite semplicemente attivando o disattivando parti del codice**
- **Difficile manutenzione**
- **Grossi sforzi realizzativi**
- **Affidabilità non sempre eccellente**

Industrializzazione del software



Due processi distinti

- Ingegneria del Dominio
 - **Analisi**: delimitazione del dominio e definizione di un insieme di requisiti riutilizzabili e configurabili per i sistemi nel dominio
 - **Progettazione**: Sviluppo di una architettura comune ai sistemi nel dominio e definizione di un piano di produzione degli stessi
 - **Implementazione**: Implementazione degli assets riutilizzabili, dei linguaggi specifici del dominio (DSL), dei generatori di codice, di una infrastruttura di supporto e del processo produttivo
- Ingegneria delle applicazioni
 - **Produzione di sistemi concreti** utilizzando gli assets sviluppati nel processo di ingegneria del dominio

Ingegneria del dominio: definizioni

- **Dominio (def. I):** spazio del problema per una famiglia di applicazioni con requisiti simili e soluzioni simili per tali requisiti.
- **Dominio (def. II):** un'area della conoscenza o un'attività caratterizzata da un insieme di concetti e una terminologia comprensibile agli addetti ai lavori dell'area in questione.

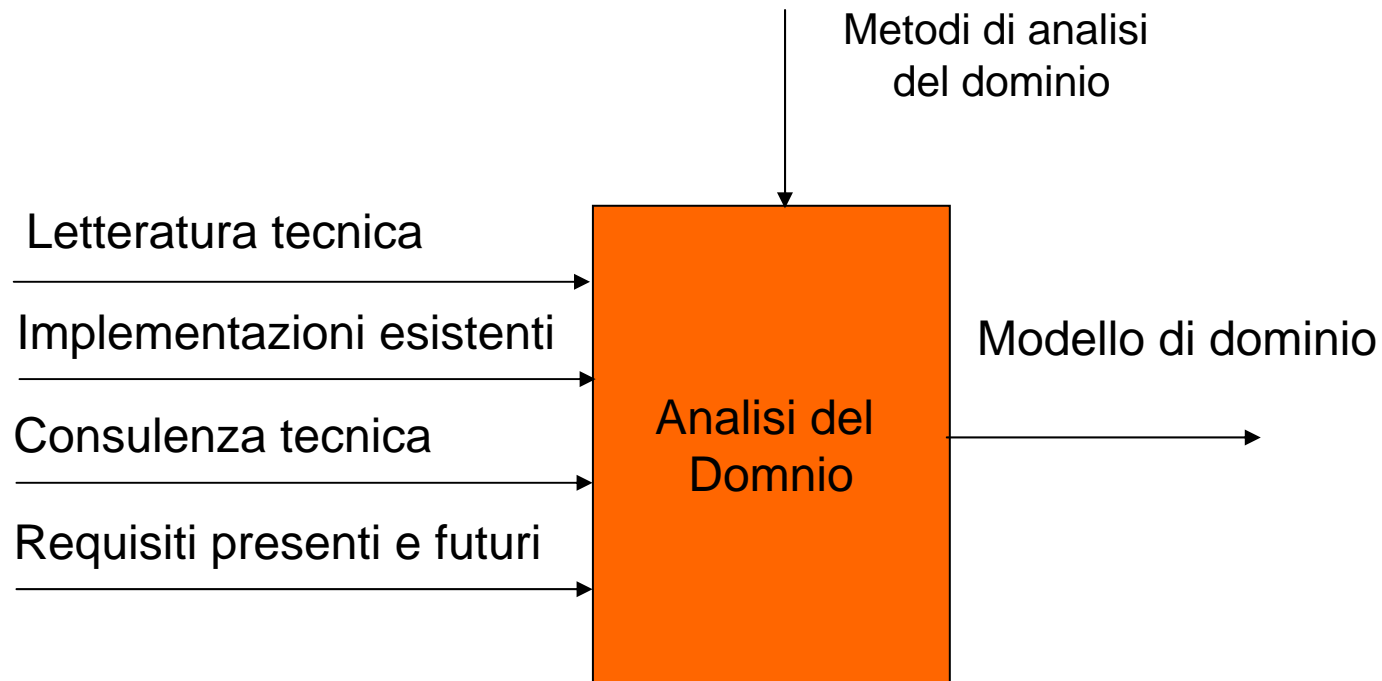
Caratteristiche:

- Un dominio può essere una particolare funzione, una classe di utenti o una intera famiglia di prodotti.
- Un dominio normalmente definisce un insieme di sistemi simili o correlati
- Un dominio definisce anche un insieme di componenti riutilizzabili (assets) che possono essere condivisi da tutti i sistemi del dominio.
- Per ridurre la loro complessità i domini possono essere suddivisi in sottodomini, separati da limiti precisi.

Analisi del Dominio

- Per analisi del dominio si intende il processo teso a identificare, analizzare e rappresentante un **modello di dominio** e una architettura software a partire dallo studio di: sistemi esistenti, letteratura, tecnologia emergente e storia evolutiva del dominio di interesse.
- Gli obiettivi dell'analisi del dominio sono:
 - La selezione e definizione del dominio
 - La costruzione di un suo modello rappresentativo

Processo di analisi del domino



Una **Famiglia di prodotti Software** può essere vista come un insieme di sistemi software simili tra loro e appartenenti allo stesso dominio.

Modello di dominio

- Un modello di dominio è formato da:
 - Definizione del dominio: definisce i limiti del dominio, gli attori in gioco e ne caratterizza il contenuto fornendo esempi, controesempi e regole generiche per l'inclusione o l'esclusione
 - Terminologia: definisce il lessico del dominio
 - Modelli concettuali: descrivono i concetti nel dominio tramite un qualche formalismo atto allo scopo (es. linguaggio naturale o diagrammi descrittivi).

Metodi classici di analisi del dominio

- FODA – Feature-Oriented Domain Analysis (developed at Software Engineering Institute)
- ODM – Organization Domain Modeling (M. Simos)
- Draco (J. Neighbors)
- DARE – Domain Analysis and Reuse Environment (W. Frakes & R. Prieto-Diaz)
- DSSA – Domain-Specific Software Architecture (ARPA)
- FAST – Family-Oriented Abstraction, Specification, and Translation (D. Weiss)
- ODE – Ontology-based Domain Engineering (Falbo et al.)

FODA¹

Con il metodo FODA si utilizzano modelli di funzionalità (feature models) per catturare le variabilità e le commonalità dei sistemi in un dominio.

L'obiettivo del FODA è quindi quello di descrivere un dominio a partire dalle funzionalità che i sistemi al suo interno devono o possono avere. Il livello di decomposizione di ciascuna feature in componenti più piccole è dettato dalle necessità di riutilizzo in ciascun contesto (complessità di modellazione vs. granularità).

[1] <http://www.sei.cmu.edu/domain-engineering/FODA.html>

FODA

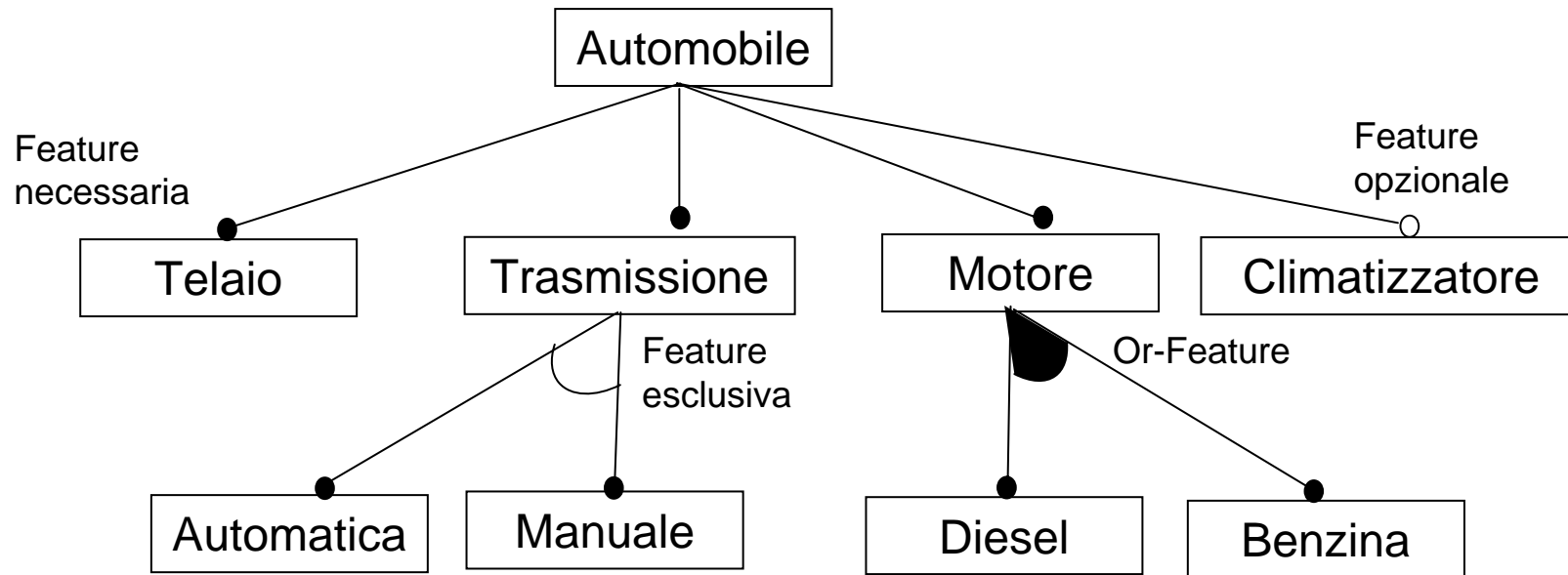
I modelli di features sono composti da:

- Diagramma di features: rappresenta una decomposizione gerarchica delle features del sistema e il loro tipo (necessarie, alternative, opzionali...)
- Definizioni: descrivono semanticamente le features
- Regole di composizione: descrivono quali combinazioni di features sono valide
- Logicità: ragioni per cui una feature dovrebbe essere inclusa/esclusa nel/dal sistema

In FODA si distinguono i seguenti tipi di feature:

- **Features necessarie:** Comuni a tutti i sistemi del dominio.
- **Feature mutuamente esclusive:** Ciascun sistema può avere solo una feature di un dato gruppo
- **Feature opzionali:** danno valore aggiunto al sistema, ma non sono indispensabili al suo funzionamento
- **Or-features:** Feature sostituibili con altre equivalenti.

Modello di Features: un esempio



Variabilità e loro gestione

- In base alla posizione nell'albero delle feature si possono distinguere:
 - **Features atomiche:** non possono essere ulteriormente suddivise.
 - **Features composte:** Sono definite a partire da altre feature
 - **Feature comuni:** sono le feature presenti in tutte le istanze di uno stesso concetto. Tutte le feature obbligatorie, che dipendono da un concetto, sono comuni. I figli obbligatori di feature comuni sono anch'essi comuni.
- La variabilità di un sistema nel diagramma delle feature si esprime usando feature feture opzionali, esclusive o di tipo or. Tali funzionalità si chiamano: feature variabili, mentre I nodi che le contengono sono dette “**punti di variazione**”

Variabilità e loro gestione

- Le feature variabili descrivono la variabilità totale di un dato sistema, esse sono:
 - opzionali, esclusive, inclusive, or-features
- I nodi di un diagramma che presentano sottoparti variabili sono detti punti di variazione (variation points, VP)
 - VP omogenei vs non omogenei: tutte le sottoparti sono dello stesso tipo
 - VP singolari vs. VP non singolari: a partire da essi può essere selezionata *almeno* una o *solo* una subfeature

Limiti del FODA

- Manca una gestione puntuale delle cardinalità
- Ha un solo livello di astrazione (il diagramma delle feature dipende dalla piattaforma in uso)
- La configurabilità dei prodotti è di difficile gestione
- I Diagrammi di sistemi complessi sono di difficile comprensione
- Non è universale. I diagrammi non sono chiari ai non addetti ai lavori
- Manca un meccanismo per l'individuazione delle features
- Non esprime chiaramente alcune possibili interdipendenze tra features

La modellazione con UML

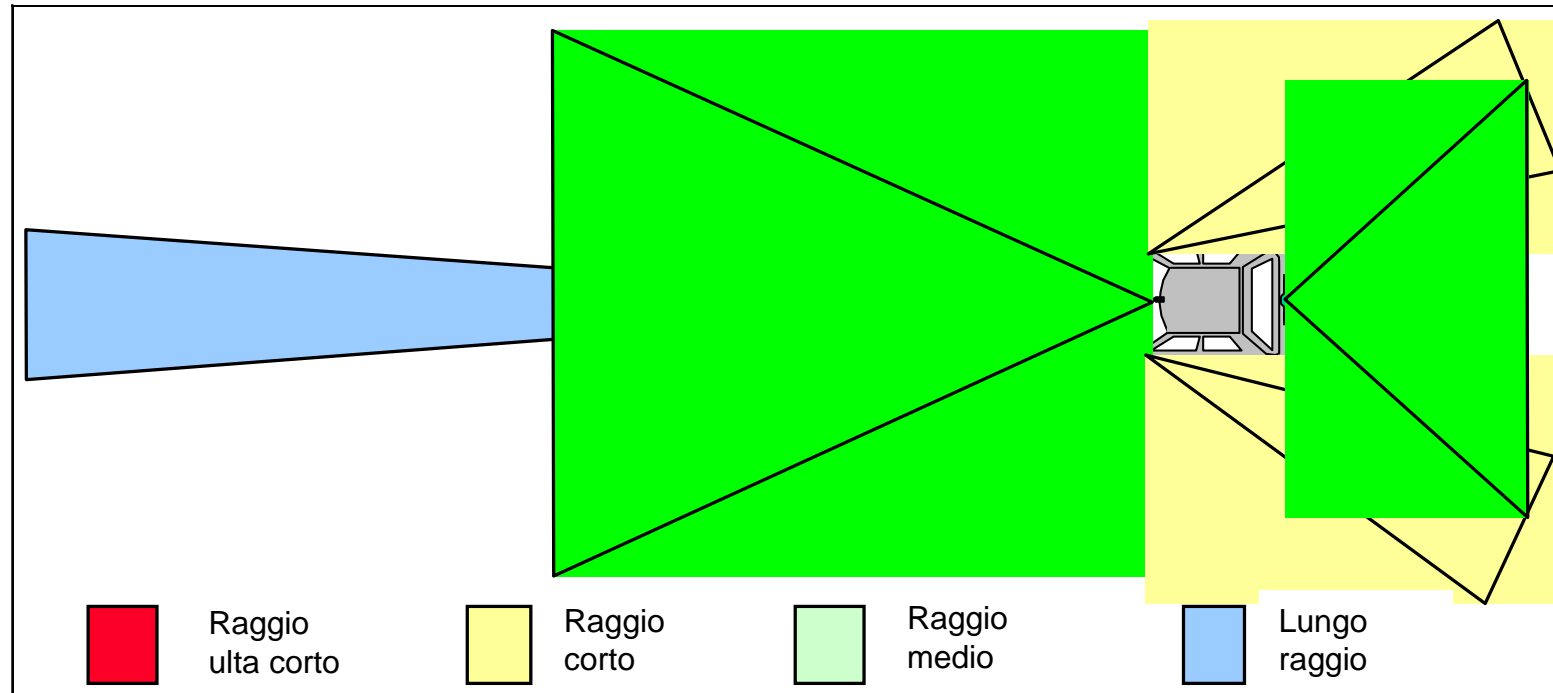
- UML (Unified Model Language) è un linguaggio di modellazione con cui è possibile superare i limiti del FODA;
- Vantaggi:
 - È uno standard supportato dalla OMG
 - È indipendente dalla metodologia usata durante la fase di analisi del dominio
 - È più “potente” dei semplici diagrammi di funzioni espressi nel FODA
 - Può modellare sistemi in modo dipendente o indipendente dalla piattaforma di utilizzo
 - È estendibile: può modellare elementi specifici di un dato dominio
 - Include meccanismi per l’individuazione delle features e la identificazione delle loro interdipendenze (diagrammi dei casi d’uso)

PARTE TERZA

Caso di Studio

Introduzione

Picture © Bosch



Car Periphery Systems (CPS)

Applicazioni del CPS

Il sistema CPS é o potrebbe essere usato per le seguenti applicazioni:

Stop and Go (S&G): mantiene una distanza minima predeterminata con il veicolo che ci precede accelerando o frenando in modo appropriato (utile negli ingorghi).

Parking Assistance (PA): mostra all'utente la distanza dal veicolo che gli sta davanti, dietro o di lato, emettendo un segnale sonoro quando si é troppo vicini ad esso.

Pre-Crash (PC): individua gli ostacoli con cui si sta rischiando una collisione (oggetti in arrivo al **vehicle envelope**) e calcola le conseguenze e le probabilità della stessa. In tal modo gli incidenti possono essere identificati prima del loro verificarsi, permettendo un uso migliore degli airbag migliorando il tempo di risposta e la velocità di accelerazione degli airbags.

Applicazioni of CPS

Le seguenti applicazioni sono invece incluse nel dominio del CPS ma non ancora implementate nel presente:

- Autonomous parking (AP)
- Automatic Cruise Control (ACC)
- Traffic sign detection (TSD)

Nel caso in esame il CPS ha incorporato anche I seguenti prodotti di vecchia generazione (legacy products):

- Parking Warner (PW): emette un suono quando ci si avvicina troppo ad un ostacolo in fase di parcheggio.
- Air Bag Firer (ABF): attiva gli airbags in seguito ad un incidente (senza possibilità di prevedere la collisione).

CPS – Esempio di glossario

Closing velocity: la differenza di velocità tra il veicolo dell'utente e un ostacolo in avvicinamento

Crash angle: l'angolo tra il vettore della velocità del centro di gravità dell'ostacolo e quella del veicolo

Vehicle envelope: un dato volume che ingloba l'auto. Esso si estende davanti, sopra di lato e dietro, ma non al di sotto del veicolo (si divide in *inner envelope* e *outer envelope*)

Sensor: un dispositivo in grado di catturare informazione circa la posizione degli ostacoli in un raggio predefinito

Esempio di definizione di dominio

- **Clienti:** sviluppatori di applicazioni per CPS.
- **Limitazioni:** Ogni veicolo può montare un numero massimo di sensori. Tutte le applicazioni che possono essere installate su un veicolo non possono possedere un set esclusivo di sensori.
- **Organizzazioni coinvolte:** Driving Security dept., Software Development dept., Systems Development dept.
- **Tecnologia in uso:** I sensori sono basati su tecnologia a ultrasuoni, possono fornire informazioni sulle distanze tra oggetti, direzione, velocità e accelerazione. L'accuratezza delle informazioni è in relazione con il raggio del sensore (dai sensori a raggio ultra corto: fino a 2 metri, fino a quelli a raggio lungo: 150 metri).

Esempi di *commonalities*

Sistema

C1: Tutti I veicoli sono equipaggiati con un set minimo di sensori.

C2: La posizione e la velocità di un ostacolo sono calcolate come la media pesata delle letture di vari sensori.

Sensori

C3: Ciascun sensore é equipaggiato con un driver software e un identificatore unico

C4: Ogni sensore deve indicare il suo raggio di portata e la sua accuratezza rispetto ad esso.

Esempi di *variabilities*

V1: Il tipo di veicolo su cui si installa il CPS può variare.

V2: Il numero, il tipo e il posto su cui sono installati i sensori può variare.

V3: L'algoritmo di misurazione della posizione e velocità degli oggetti può variare. In particolare, i pesi usati nei calcoli variano in accordo alla accuratezza e al numero dei sensori sul veicolo.

V4: La lingua in cui deve essere scritta la documentazione varia in base alle necessità del cliente finale.

V5: La frequenza delle misurazioni ambientali può variare.

Procedimento di identificazione

- Per le *commonalities*
 - Sono stati esaminati i sistemi legacy già esistenti alla ricerca di feature essenziali: è stata stilata una lista per ciascun sistema e si è visto cosa avevano in comune vari sistemi
 - La granularità (o livello di definizione) delle feature comuni è stato definito in modo tale da rispecchiare i reali bisogni del cliente finale.
- Per le *variabilities*
 - Le variabilities sono state derivate dalle commonalities.
 - Per verificare la necessità di una variabilità, le variabilità sono state divise in caratteristiche essenziali e variabili di configurazione.
 - Tale divisione ha reso possibile la definizione di un livello di dettaglio tale da evitare il dispararsi del numero delle variabilità selezionabili (profondità ridotta dell'albero delle features).

Risoluzione e gestione delle variabilità

- La creazione dell'albero delle features é stata gestita tramite un modello decisionale (decision model DM).
- Un decision model riunisce tutte le decisioni aperte che caratterizzano il dominio insieme ai corrispondenti intervalli di possibili valori associati a ciascuna decisione.
- Ciascuna decisione é stata rappresentata con una domanda (una variabile) e un intervallo di possibili risposte (valori della variabile)
- In seguito il decision model é stato rappresentato tramite un albero, secondo lo schema FODA

Schema di rappresentazione del DM

Types:

Basic types: integer, real, Boolean

Intervals: [1-10]

Enumeration types: (x | y | z)

Aggregation types: x: type

y: type

z: type

Set types: set {x, y, z}

Collection types: collection {x, y, z}

Restrizioni aggiuntive:

– Opzionale o necessario

– Valore di default

Decision model del CPS

VehicleType: (car | bus | truck)

Nbsensors: [1-12]

Sensor

id: string

type: (USR, SR, MD, LR)

location

side: (front | rear | driver | co-pilot)

position (central | right | left)

sampleFreq: real

SensorsInstalled: nonempty set of Sensor

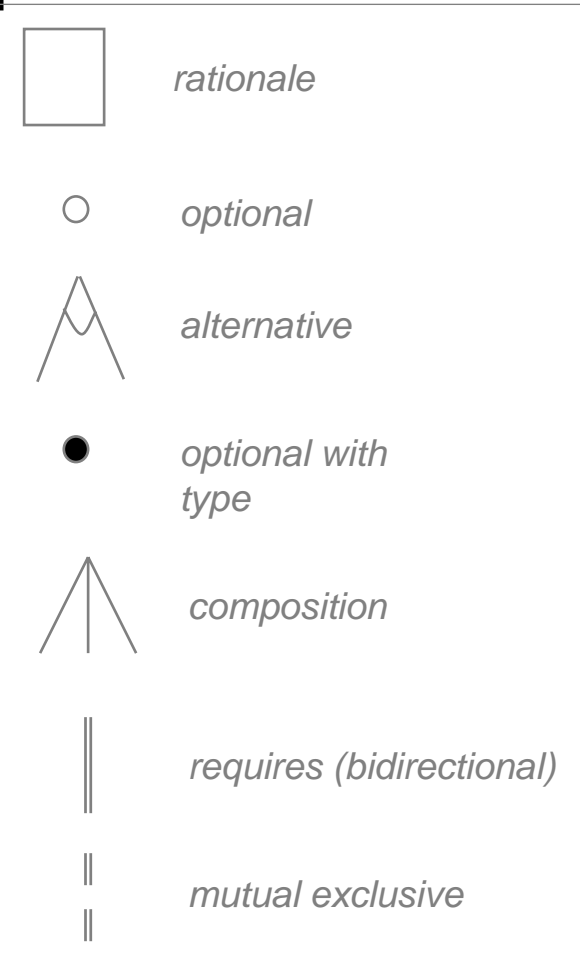
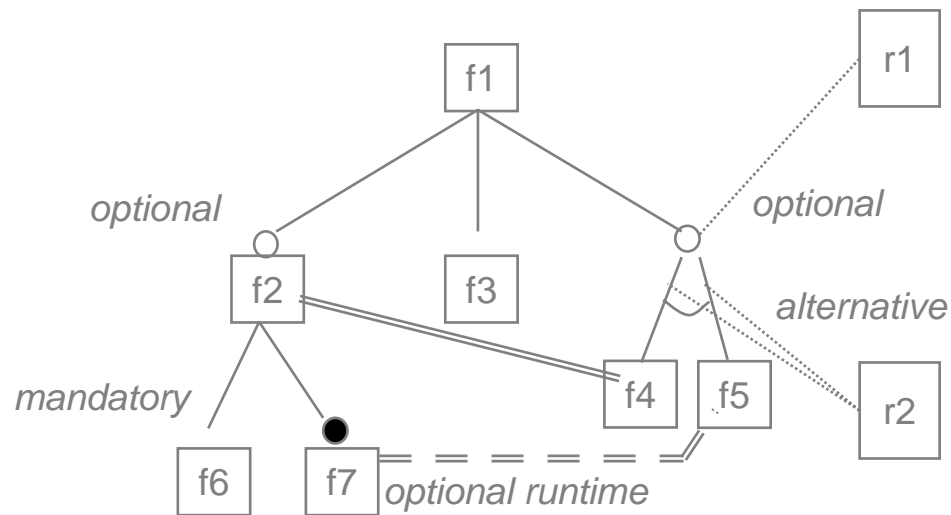
Algorithm: (weighted | notWeighted)

DocumentationLanguage: (English | German | Spanish)

Restriction:

VehicleType = car and Nbsensors <=10

Rappresentazione del DM tramite una struttura ad albero di tipo FODA



Definizione delle features riusabili

Per la risoluzione delle variabilità sono state impiegate due tecniche diverse:

- Run time binding
 - Libreria di componenti riusabili
 - Libreria di servizi riutilizzabili. I servizi sono compilati e usati dall'utente in fase di run-time
- Construction time binding
 - I componenti sono stati creati sotto forma di “code templates” che devono essere riempiti durante la fase di creazione di ogni singolo componente finale. Il processo può essere (in parte) completato in modo automatico (code generation). Il vantaggio di questa tecnica è che può essere utilizzata per tutti i componenti riusabili del dominio e non solo il codice (es. requisiti, documentazione, specifiche, ecc.)

Esempio: variazioni nei documenti

The vehicle carries 3 sensors on the front, 3 on the back and
if VehicleType = car then 2 else 3 end if on the sides.
if VehicleType <>car then “include a brief explanation of why 3 sensors are necessary on the sides” end if

The range of the sensors is as follows:

for each sensor in SensorsInstalled

Sensor sensor.id is

Case sensor.type = USR then ultra-short range

Case sensor.type = SR then short range

Case sensor.type = MR then medium range

Case sensor.type = LR then long range

The algorithm to calculate objects position and relative speed
if Algorithm = notWeighted then does not use **else** uses **end if**
weights to compute the average of sensor information.

DOMANDE?

Piergiorgio Di Giacomo
R&D Area
Software Engineer
Piergiorgio.DiGiacomo@esi.es

Parque Tecnológico, # 204
E-48170 Zamudio
Bizkaia (Spain)
Tel.: +34 94 420 95 19
Fax: +34 94 420 94 20
www.esi.es