

A large, stylized orange graphic of a person with arms raised, surrounded by yellow circles, serving as a background for the title.

# Architettura del software: dai requisiti ai casi d'uso



**GUISA**  
Gruppo Utenti Italiani  
Solution Architect

# Lorenzo Barbieri



- Sono un Senior Trainer/Consultant in ObjectWay SpA ([www.objectway.it](http://www.objectway.it)), specializzato in architetture Microsoft .NET, Windows, SQL Server, Visual Studio Team System, Virtual PC/Virtual Server
- Collaboro con UGIdotNET, INETA, Team System Rocks!, Windowserver.it e sono tra i soci fondatori di GUISA
  - [lorenzo.barbieri@objectway.it](mailto:lorenzo.barbieri@objectway.it)
  - [www.geniodelmale.info](http://www.geniodelmale.info)

# Non parleremo di UML

- Se non per quanto riguarda gli Use Case e solo da un punto di vista prettamente pratico.
- Per maggiori informazioni su UML si può vedere la sessione di Riccardo Golia disponibile qui (previa registrazione):  
[http://www.ugidotnet.org/workshops/workshops\\_detail.aspx?ID=9ed3a5e7-a69c-4258-af7b-6ec3a8bcd025](http://www.ugidotnet.org/workshops/workshops_detail.aspx?ID=9ed3a5e7-a69c-4258-af7b-6ec3a8bcd025)

# Cos'è un requisito

1. **Condizione o capacità** che occorre all'**utente** per risolvere un problema o raggiungere un **obiettivo**
2. **Condizione o capacità** che deve essere raggiunta o posseduta dal **sistema** per soddisfare un **contratto**, standard, specificazione o altro documento formale
3. La **rappresentazione documentata** di una condizione o capacità (1 o 2)

# Un po' di teoria

- La Norma **ISO9126**, pubblicata nella sua prima versione nel 1991, ha definito il modello dei requisiti qualitativi del Software.
- Secondo tale modello i requisiti sono raggruppabili in 6 "caratteristiche" e in 21 "sottocaratteristiche".

# Funzionalità

- Capacità di soddisfare esigenze esplicite od implicite.
  - Idoneità = funzionalità appropriate per specificati compiti
  - Accuratezza = precisione dei risultati
  - Interoperabilità = capacità di interagire con altre applicazioni
  - Sicurezza = protezione da utilizzi non autorizzati
  - Concordezza = aderenza a standard o regolamentazioni legislative

# Disponibilità

- Capacità di fornire una continuità di servizio
  - Maturità = mancanza di interruzioni per malfunzionamenti
  - Tolleranza = ridotto degrado in caso di malfunzionamenti
  - Recupero = capacità e velocità di ripristino dopo interruzioni

# Usabilità

- Facilità di utilizzo da parte degli utenti
  - Comprensione = acquisizione di adeguato livello di conoscenza
  - Apprendimento = velocità di familiarizzazione
  - Utilizzabilità = facilità di uso e controllo
  - Attrattiva = livello di gradimento nell'utilizzo

# Efficienza

- Capacità di fornire prestazioni adeguate
  - Tempo Risposta = reattività agli stimoli dell'utente
  - Utilizzo risorse = utilizzo adeguato delle risorse del sistema

# Manutenibilità

- Facilità di manutenzione correttiva e evolutiva
  - Analizzabilità = facilità di diagnosi e identificazione componenti
  - Modificabilità = facilità di inserimento di modifiche
  - Stabilità = limitazione di effetti indesiderati derivanti da modifiche
  - Collaudabilità = facilità di testare le modifiche apportate

# Portabilità

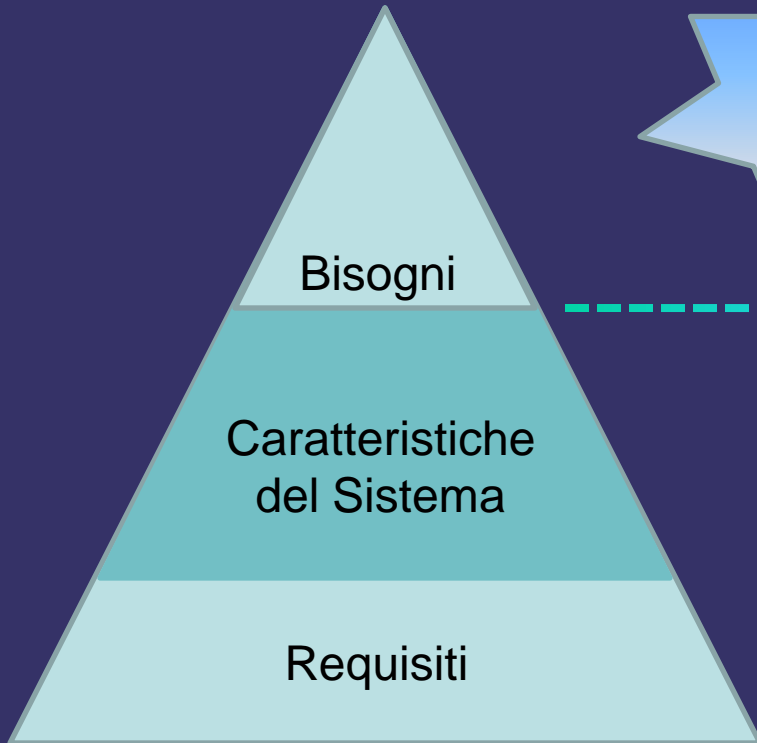
- Trasferibilità da un ambiente all'altro
  - Adattabilità = facilità di adeguamento ad un nuovo ambiente
  - Installabilità = velocità e completezza di installazione
  - Coesistenza = capacità di risiedere con altre applicazioni nello stesso ambiente
  - Sostituibilità = capacità di rimpiazzare un'altra applicazione con simili funzionalità

# Contesto



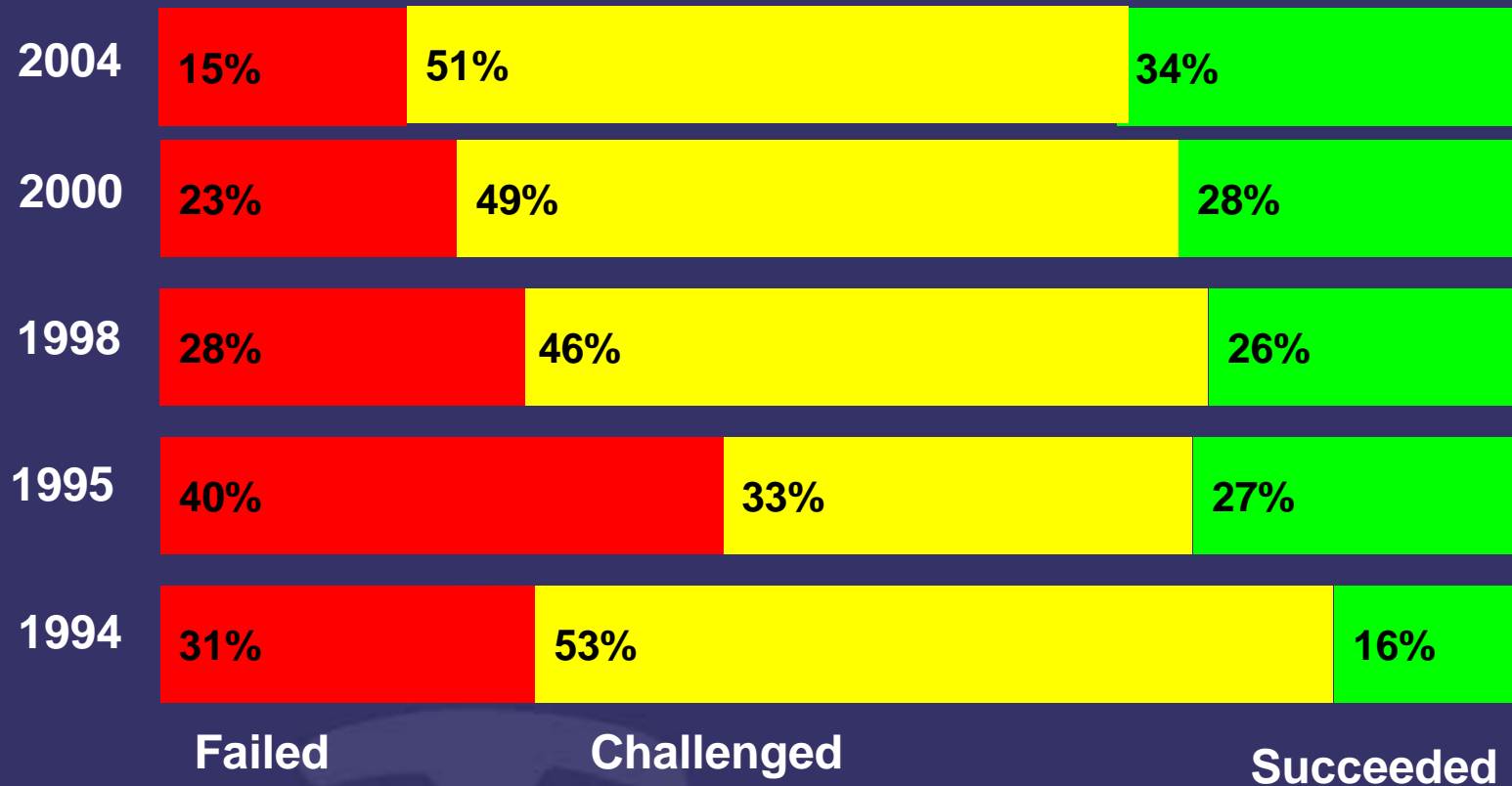
Problema

Soluzione



Analisi, Design, Implementazione,...

# Stato di salute dei progetti IT



This chart depicts the outcome of the 30,000 application projects in large, medium, and small cross-industry U.S. companies tested by The Standish Group since 1994.

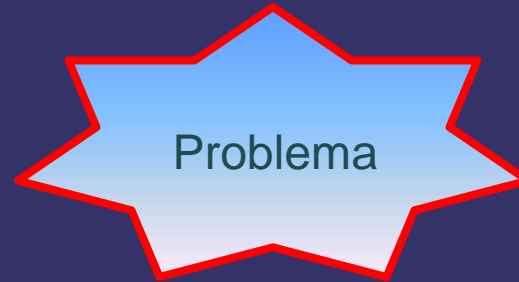
# Fattori per il successo

- Progetti di successo:
  - Coinvolgimento degli utenti 16%
  - Supporto dal management 14%
  - Chiarezza dei requisiti 12%
- Progetti "problematici":
  - Mancanza di coinvolgimento degli utenti 13%
  - Specifiche incomplete 13%
  - Specifiche mutevoli nel tempo 12%
- Progetti falliti:
  - Requisiti incompleti 13%
  - Mancanza di coinvolgimento degli utenti 12%
  - Specifiche mutevoli nel tempo 8%

# I Requisiti vanno gestiti:

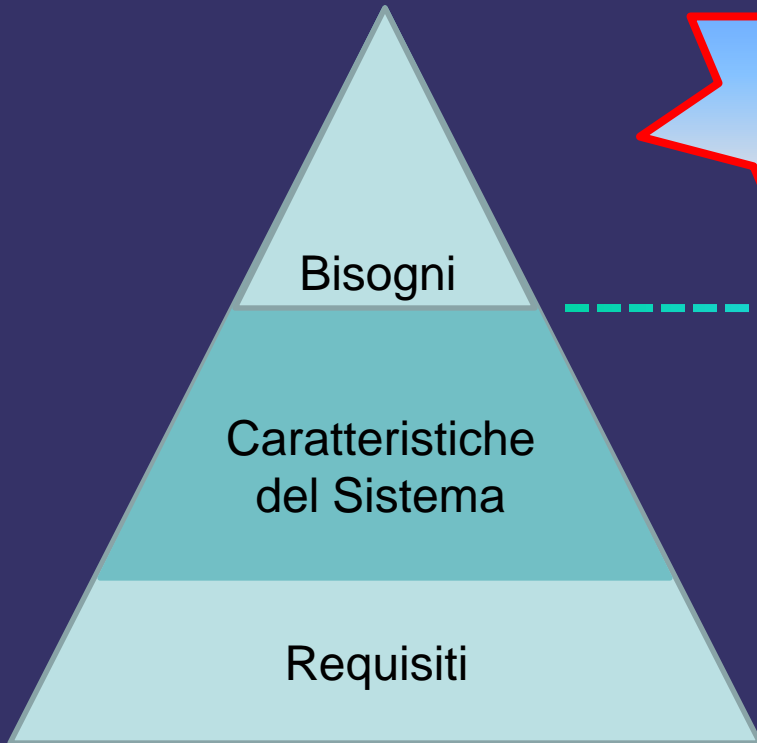
- Come si vede la mancanza di requisiti o la mancanza di gestione è tra le cause principali del fallimento o dei problemi ai progetti.

# Definiamo il problema



Problema

Soluzione



Analisi, Design, Implementazione,...

# Problema

- Descrivere regole e “comportamenti” del *giuoco* del calcio



# Definizione

- Bisogna ridurre la distanza tra la percezione delle cose e cosa si desidera realmente:
  - Partita tra amici all'oratorio
  - 
  - Partita di Champions League
- La definizione del problema serve a far chiarezza sugli aspetti dubbi

# Cerchiamo di capire bene

- Lo sforzo iniziale di comprensione del problema serve a evitare la sindrome del “Si... Ma...” al termine del progetto:
  - Si, soddisfa i requisiti che abbiamo documentato...
  - Ma, NON risolve il problema...

# Come definire bene il problema

- Conviene creare un documento di Vision (RUP, MSF, etc...)
  - Identificare il “vero” problema
  - Identificare gli Stakeholder
  - Identificare i vincoli e i confini del sistema
  - Identificare i bisogni degli Stakeholder

# Un possibile template

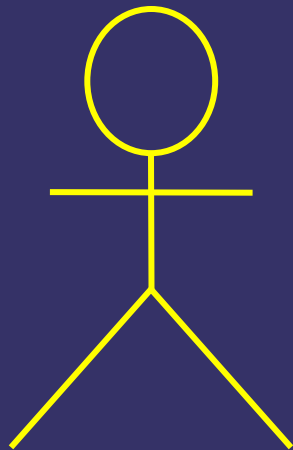
- Possiamo ad esempio usare questo template minimale per partire a definire il nostro documento di Vision:

<b>Il problema è</b>	(descrivere il problema)
<b>e coinvolge</b>	(gli stakeholders coinvolti)
<b>L'impatto è</b>	(qual è l'impatto negativo del problema)
<b>Una soluzione soddisfacente offrirebbe</b>	(elencare i benefici principali di una soluzione)

# A volte non è semplice definire i confini del sistema

- Non sempre è semplice definire i confini di un sistema.
- Per chiarirsi meglio i confini può essere utile introdurre il concetto di Attore

# Attori

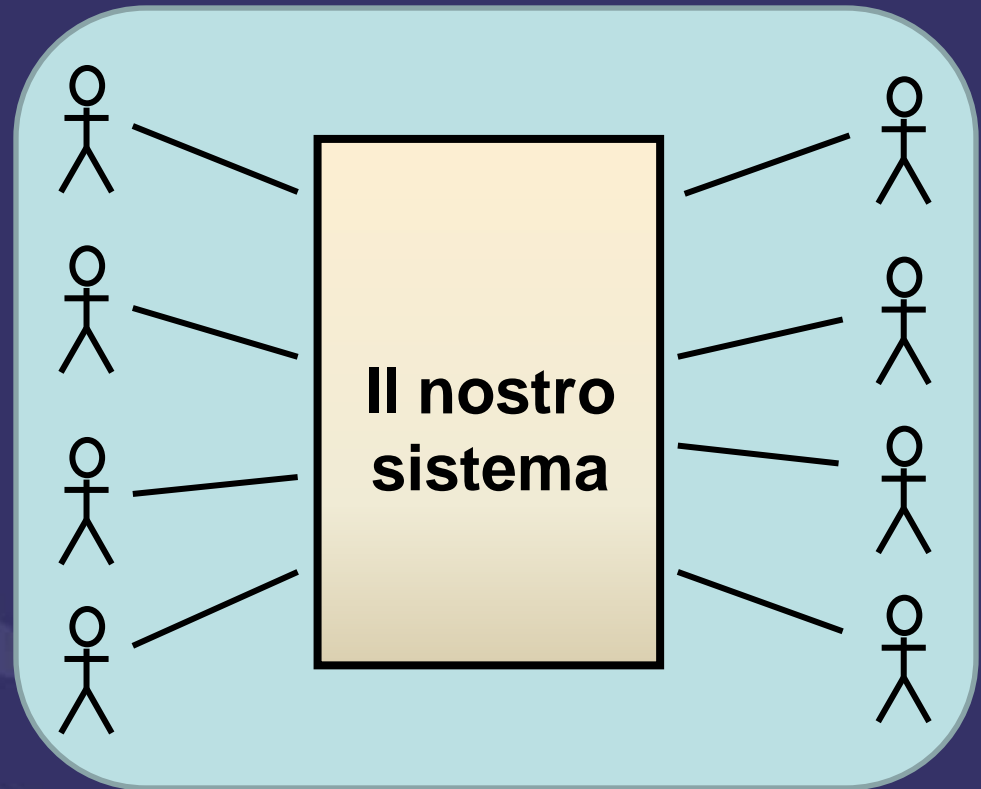


**Attore**

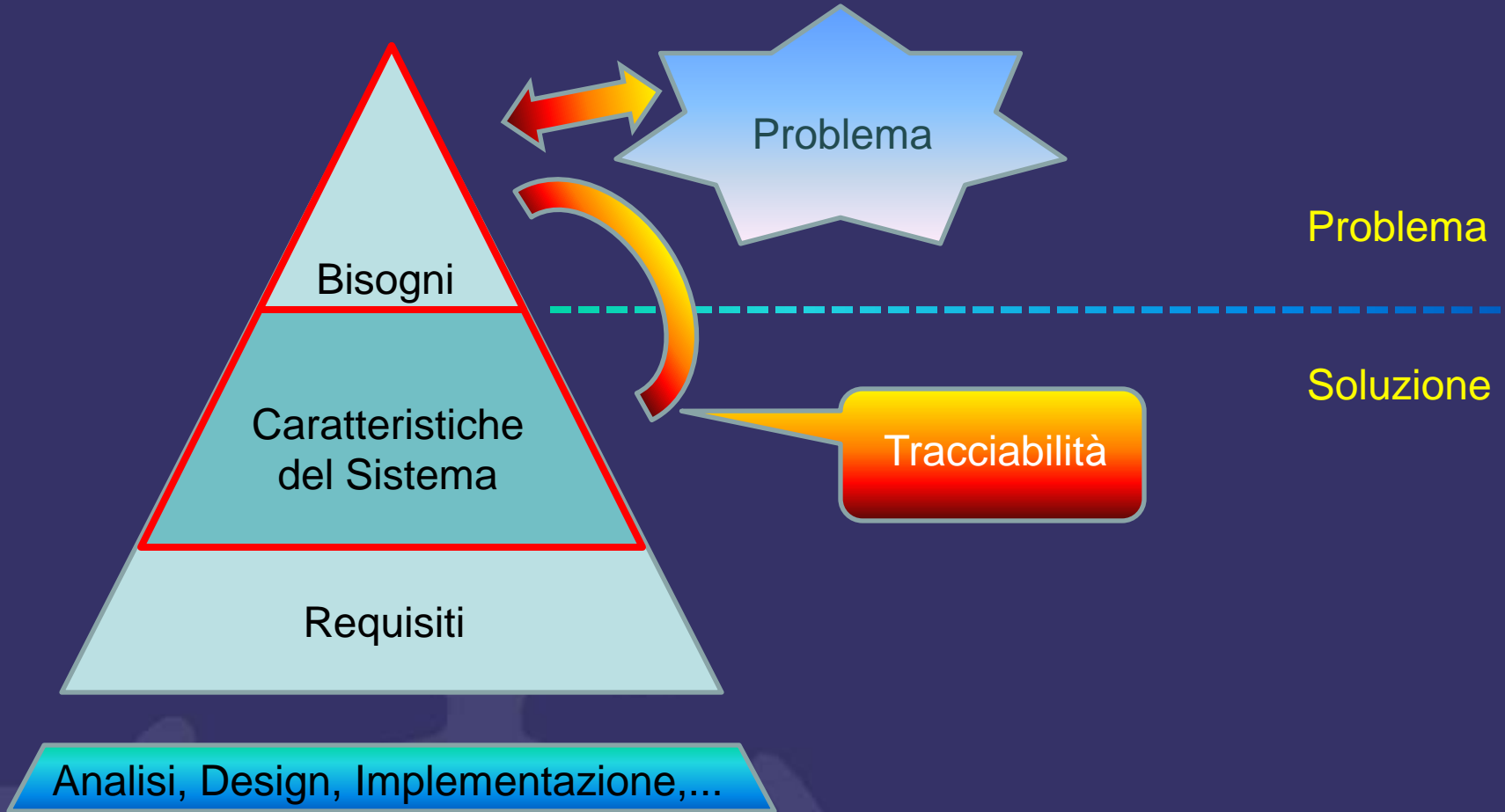
- Un attore è un **umano**, una **macchina** o un **altro sistema**
- Un attore **usa** il sistema o è coinvolto durante l'uso del sistema – ha una interazione diretta col sistema
- Gli attori **non sono parte del nostro sistema**
- Gli attori umani sono un **sottoinsieme degli stakeholder**

# Identifichiamo i nostri attori:

- Attaccante
- Centrocampista
- Difensore
- Portiere
- Arbitro
- Guardalinee
- Pubblico



# Bisogni e Caratteristiche



# Cosa sono i requisiti dell'utente?

- **Bisogno**: “una opportunità o un aspetto di un problema di business, o di un problema personale, che induce uno stakeholder a considerare di usare un nuovo sistema (costruendolo, comprandolo o affittandolo)”
- **Caratteristica del sistema**: “un servizio che il sistema fornisce per soddisfare uno o più bisogni ”

# Esempio:

- N001: La partita deve durare un tempo definito
- N002: Il numero di giocatori deve essere definito
- Nxxx: Bisogna definire cosa può fare il portiere
  
- Etc...

# Esempio:

- F001: La partita dura 90'
- F002: La partita è divisa in 2 tempi da 45'
- F003: Ogni squadra dispone di 10 giocatori più 1 portiere
- Fxxx: Il portiere può parare con le mani
- Fxxy: Il portiere non può toccare la palla con le mani se...

# Come "elicitare" i requisiti utente:

- Interviste
- Questionari
- Brainstorming & sintesi delle idee
- Role Playing
- Storyboarding e prototyping
- Revisione di specifiche dei requisiti fornite dal cliente
- Osservazione diretta o indiretta

# Ostacoli alla elicitazione dei requisiti utente

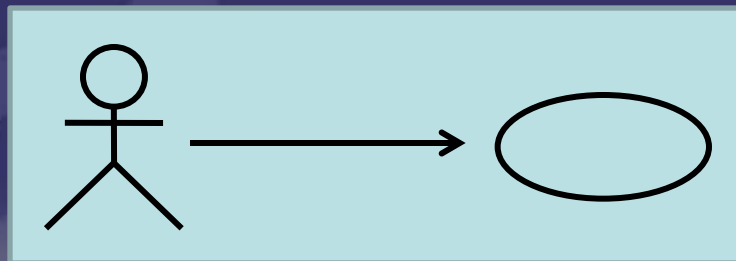
- Troppa fretta
  - “Non vedo l’ora di usarlo, terminalo al piu’ presto e installalo su tutti i client”
  - Risposta: “Adesso che ci penso, cosa ne diresti se ...? Ma cosa succederebbe se...?”
- Sindrome delle “Rovine nascoste”
  - Più requisiti troviamo, più ci accorgiamo che ne mancano

# Ostacoli alla elicitazione dei requisiti utente

- **Sindrome "utente vs. sviluppatore"**
  - Gli stakeholder sanno cosa vogliono ma non sanno esprimerlo e dettagliarlo oppure non lo sanno proprio
  - Gli stakeholder pensano di sapere cosa vogliono finchè non gli date quello che loro pensavano di volere
  - Gli analisti pensano di capire i problemi degli utenti meglio degli utenti

# Introduzione agli Use Case

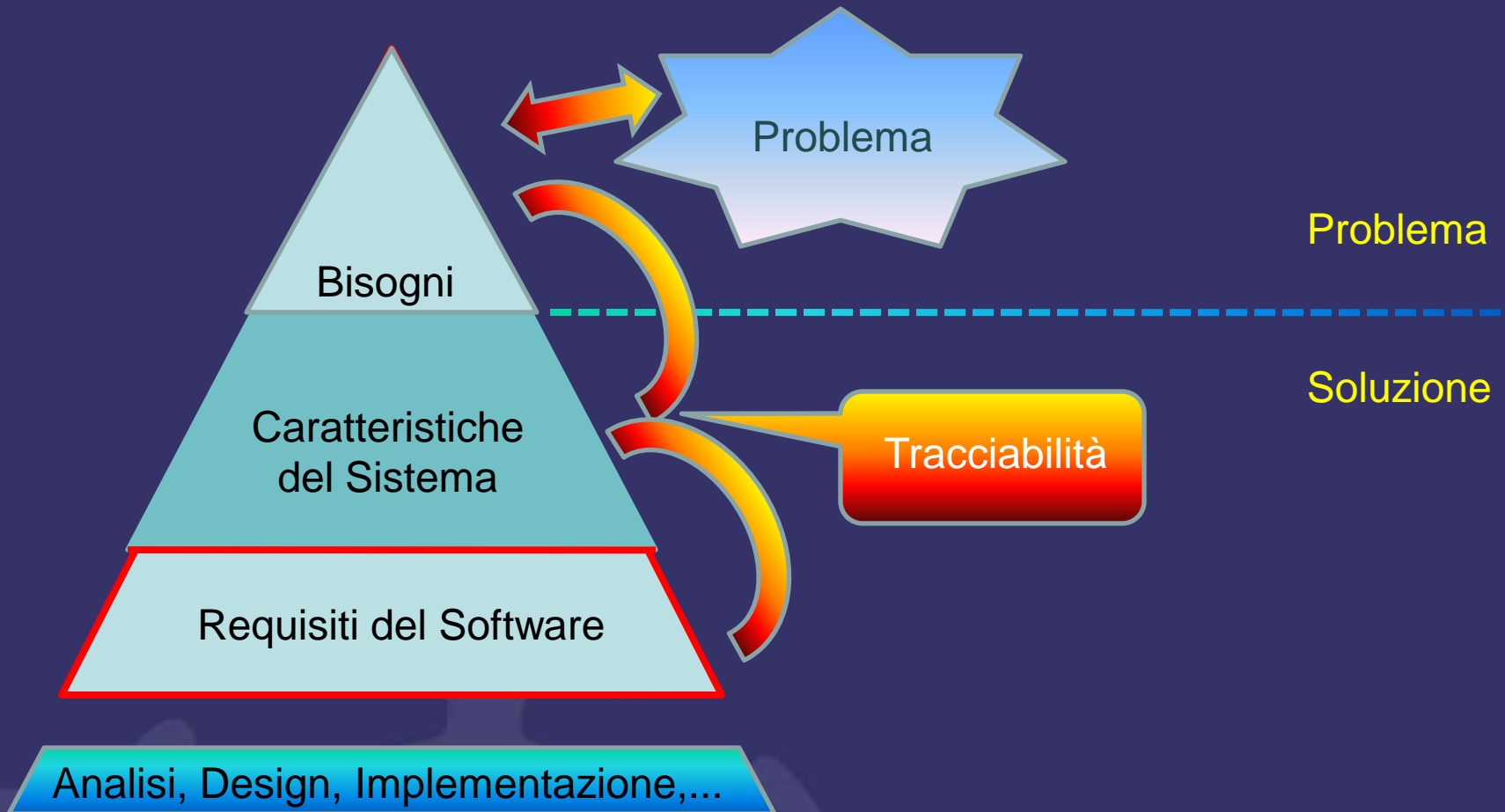
- Storie di uso del sistema, “user stories”
- Ogni caso d’uso è una collezione di scenari di uso
- Ogni caso d’uso è scatenato da almeno un attore
- Ogni caso d’uso descrive:
  - tutti i possibili di modi (scenari) per ottenere un preciso valore o obiettivo per un particolare attore
  - tutte le eccezioni che non permettono di ottenere il valore o l’obiettivo
- Lo stile con cui è scritto varia dall’informale/romanzato al formale/pseudocodice
  - deve essere comprensibile al target (in questa fase lo stakeholder)



# Gli use-case aiutano a elicitarre i bisogni

- Rappresentazione grafica sintetica
- Identificare chi interagisce col sistema
- Identificare quali interfacce il sistema avrà e verso chi
- Accordarsi col cliente sulle "macrofunzionalità" (feature)
- Raggruppare le "feature" per attore (obiettivi di ciascun attore)
- Aiutare a verificare che non si stanno tralasciando "feature"

# Definire i Requisiti Software



# Requisiti Utente vs. Requisiti Software

- I Requisiti Utente che abbiamo visto fino ad adesso sono definiti dall'utente (li scriviamo attraverso il processo di elicitazione)
- I Requisiti Software servono per descrivere in maniera completa il sistema
- Un Requisito Software:
  - descrive un comportamento del sistema
  - soddisfa un bisogno dell'utente, un vincolo, uno standard, etc...

# Come specificare i Requisiti Software

- I Requisiti Software possono essere specificati in vari modi:
  - Software Requirements Specifications tradizionali (testuali)
  - Use Case + Specifiche Addizionali
  - Scenari + Quality of Service Requirements
  - User Stories
  - Etc...
- A volte può essere conveniente un approccio "ibrido"

# Chi deve poter leggere una SRS

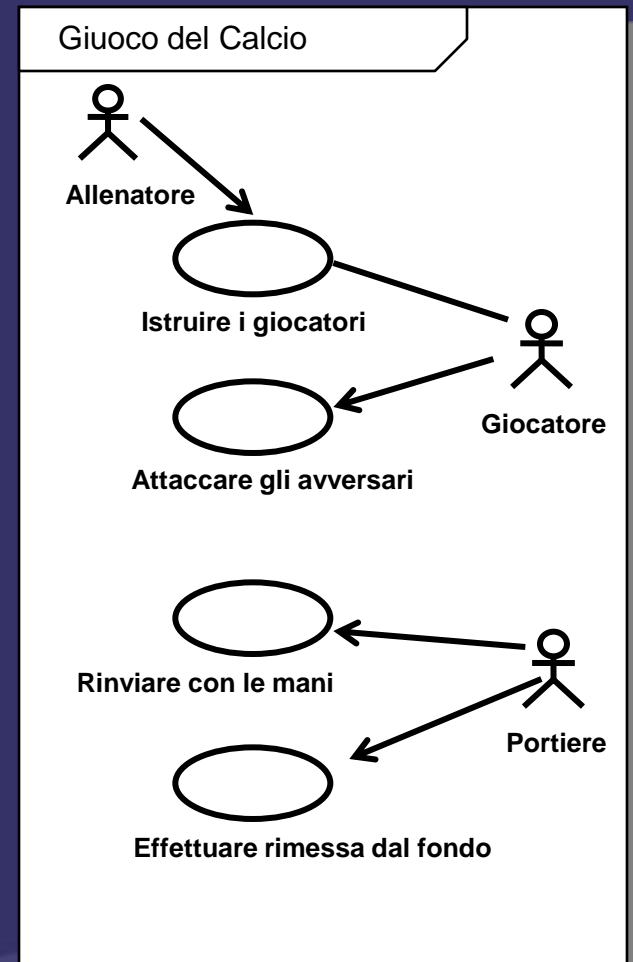
- A prescindere dalla strada scelta, una SRS deve essere leggibile da:
  - Stakeholder, committenti, etc... (è la base del contratto)
  - Team di progetto (è la base dell'analisi, dell'architettura e del design)
  - Team di test
  - Team che prepara la documentazione
- In pratica tutti i soggetti coinvolti nel progetto

# Specificare i requisiti funzionali tramite Use Case

- In questa fase gli Use Case saranno molto più dettagliati di quelli usati (opzionalmente) nella fase precedente:
  - Uno use case è un completo e significativo flusso di interazioni fra attori e sistema
  - Uno use case è iniziato da un attore che richiama una certa funzionalità del sistema, che reagisce per fornire qualcosa di valore all'attore
  - L'insieme di tutti gli use case definisce tutti i modi possibili di usare il sistema

# Use-Case Model

- Non esiste solo il Diagramma "grafico" degli Use Case
- Use Case Model Survey
  - Lista di diagrammi, attori e breve descrizione
- Use Case Specification (una per ogni Use Case)
  - Descrizione
  - Flussi (principale, secondari, eccezione)
  - Pre/Post Condizioni
  - Requisiti supplementari dello UC



# Gli Use Case non sono SOLO i disegni

- Il cuore della specifica è la Use Case Specification
- In molte realtà la parte grafica non viene utilizzata, si usa solo la Use Case Survey e le varie Use Case Specificatio

# Un possibile template per la Use Case Specification

- Nome use case e Breve Descrizione
- Flusso base / principale (flusso happy days)
- Flussi secondari
- Flussi di eccezione
- Requisiti speciali (opzionale): tutti i non funzionali legati allo specifico UC
  - Immagini, schermate, etc...
  - Requisiti di prestazione, scalabilità, etc...

# Un possibile template per la Use Case Specification

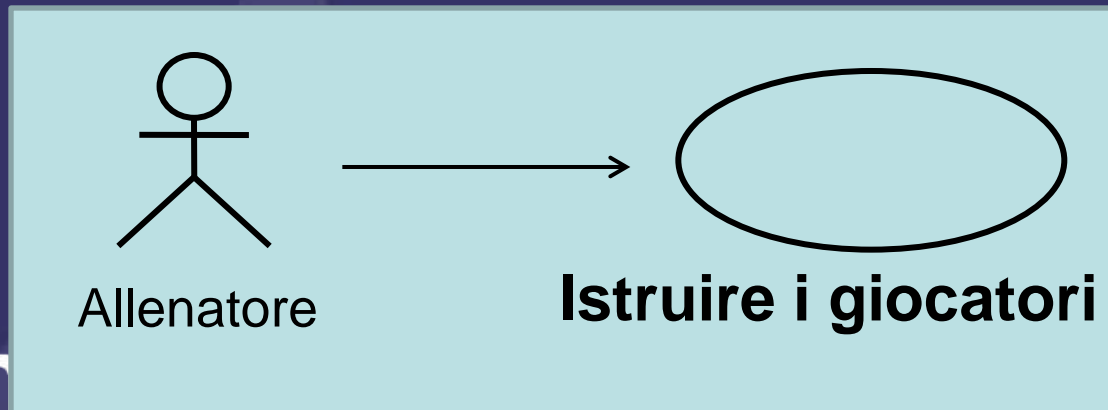
- Precondizione (opzionale): condizione che vincola l'avviamento dello use case (non è la causa che scatena lo UC)
- Postcondizioni (opzionali): definisce lo stato del sistema quando lo use case termina
- Extension Points: descrizioni degli eventuali punti di estensione nei flussi

# Partire da template standard

- Per gli UC esistono molti buoni template
  - RUP contiene vari template per UC
  - Cockburn propone un formato compatto facile da leggere
  - Constantine e Lockwood un formato essenziale
- Vantaggi della standardizzazione
  - Sfruttare il lavoro di altri
  - Si è produttivi velocemente
  - Tutti sanno dove ritrovare le informazioni
  - I documenti sono più facili da leggere
- Attenzione a scegliere il template con il giusto livello di dettaglio

# Che nome scegliere

- Ogni UC dovrebbe avere un nome che indichi il valore che fornisce
- Meglio usare verbi infiniti e transitivi
- Deve essere leggibile usando l'attore principale come soggetto



# Flussi – Come scriverli

- Stile e linguaggio comprensibile al cliente/utente
- Descrivere il flusso degli eventi, non solo le funzionalità.
- Non descrivere cosa accade fuori del sistema o in altri UC
- Evitare terminologia vaga
- Adottare un livello di dettaglio omogeneo
- Valutare l'uso di diagrammi grafici per chiarire meglio il flusso (Sequence Diagram, etc..)

# Dove specificare i requisiti Non Funzionali

- Se sono specifici del singolo Use Case:
  - All'interno dei flussi
  - All'interno della sezione requisiti speciali
- Se sono generici per il sistema
  - Nel documento di Specifiche Addizionali

# Cosa non deve contenere una SRS?

- Design - Come soddisfare i requisiti
- Test - Come sapere se i requisiti sono stati soddisfatti
- Piani di progetto – Quando e come i requisiti saranno implementati

# Vantaggi della modellazione con Use Case

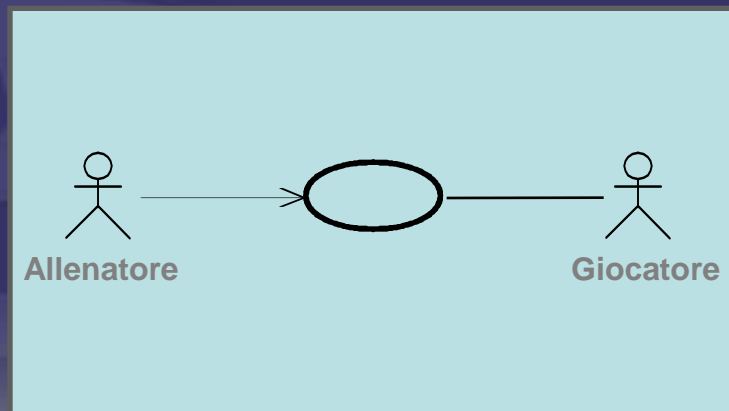
- Permette di identificare al meglio il ruolo degli attori del sistema
- Semplifica gli accordi con il cliente con riferimento ai requisiti del sistema
  - La terminologia è facilmente comprensibile
- Aiuta la comprensione del sistema da parte dello sviluppatore
- Aiuta a controllare che vengano identificati tutti i requisiti (sindrome delle Macerie Nascoste)
- Identifica le interfacce del sistema (UI, SW, HW)
  - attenzione a non usarlo “direttamente” per la scomposizione in sottosistemi

# Prima di concludere

- Diamo qualche “dettaglio” in più sui formalismi grafici usati negli Use Case Diagram

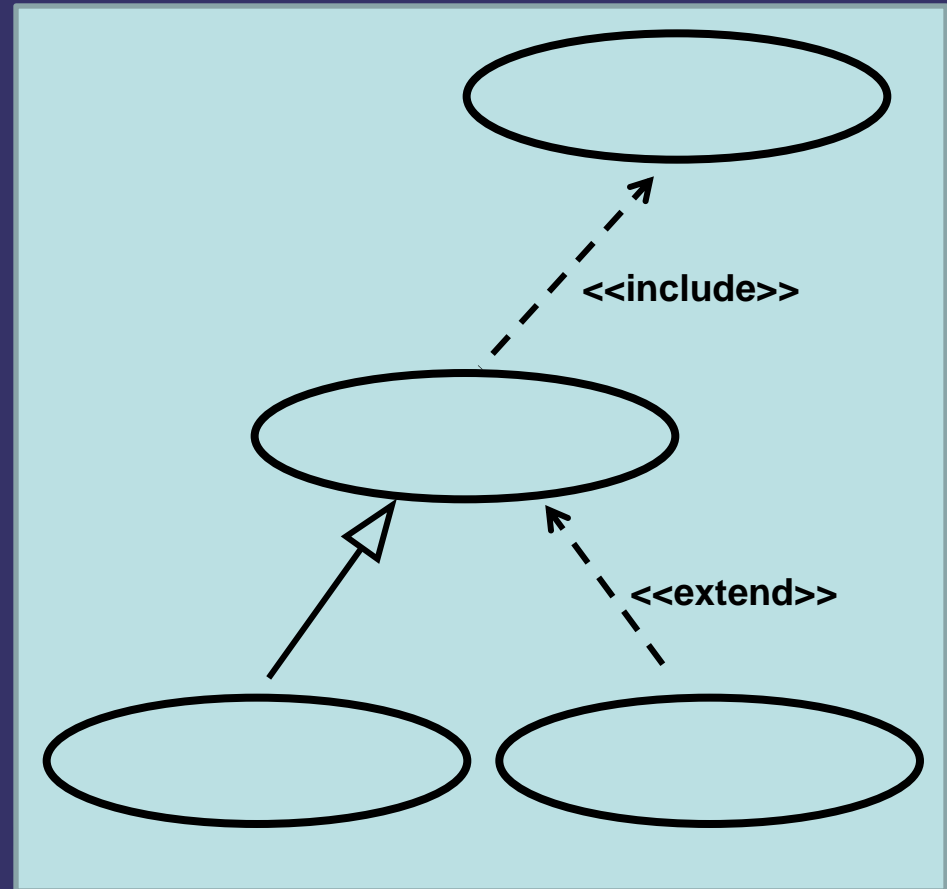
# Relazione tra attore e Use Case

- La freccia di comunicazione definisce l'attore che inizia lo use case.
- I messaggi di ritorno sono impliciti.
- Una linea senza freccia suppone una comunicazione nei due sensi
- Usare la freccia solo se serve veramente



# Relazioni fra Use-Case

- Include
  - Per isolare parte del comportamento di uno use case, che **dipende dal risultato dell'include, non dal flusso** (encapsulation).
- Extend
  - Per isolare parti **opzionali** o **eccezionali** di uno use case (condizioni nell'*abstract*)
- Generalizzazione
  - Per mostrare che lo use case **condivide scopo, struttura e comportamento**.
  - E' possibile anche generalizzare Attori



# Una piccola aggiunta

- Il modello degli Use Case ha ispirato molti modelli successivi per la raccolta e la gestione dei requisiti
- Microsoft Solutions Framework si basa sulla definizione di Scenari e Persone

# Differenze

- Uno Scenario è un singolo percorso dell'utente nel sistema
  - E' molto più granulare
  - E' più facile da "stimare"
  - Il numero di Scenari può "esplodere"
- Una Persona è una "profilazione" degli utenti reali in base non solo a caratteristiche funzionali, ma anche a caratteristiche "comportamentali" e "demografiche"

Ricordatevi di compilare il modulo di Feedback

**DOMANDE?**

