



UNIVERSITÀ  
DEGLI STUDI  
DI PALERMO

# Testing

Ingegneria del Software  
a.a. 2017/18

# Test - concetti di base

- I test hanno lo scopo di dimostrare che un programma fa ciò che intende fare e di scoprire i difetti del programma prima della sua messa in servizio.
- Quando si esegue il test del software, si esegue un programma utilizzando dati artificiali.
- Verificate i risultati dell' esecuzione del test per verificare la presenza di errori, anomalie o informazioni sugli attributi non funzionali del programma.
- Può rivelare la presenza di errori NON la loro assenza.
- I test fanno parte di un processo più generale di verifica e convalida, che comprende anche tecniche di validazione statica.

# Obiettivi del processo di test del software

- Dimostrare allo sviluppatore e al cliente che il software soddisfa i suoi requisiti.
- Per il software personalizzato, ciò significa che nel documento dei requisiti deve essere effettuata almeno una prova per ogni requisito. Per i prodotti software generici, significa che ci dovrebbero essere test per tutte le caratteristiche del sistema, più combinazioni di queste caratteristiche, che saranno incorporate nella versione del prodotto.
- Scoprire situazioni in cui il comportamento del software è errato, indesiderabile o non conforme alle sue specifiche.
- La verifica dei difetti riguarda l' eliminazione di comportamenti indesiderati del sistema come crash, interazioni indesiderate con altri sistemi, calcoli errati e corruzione dei dati.

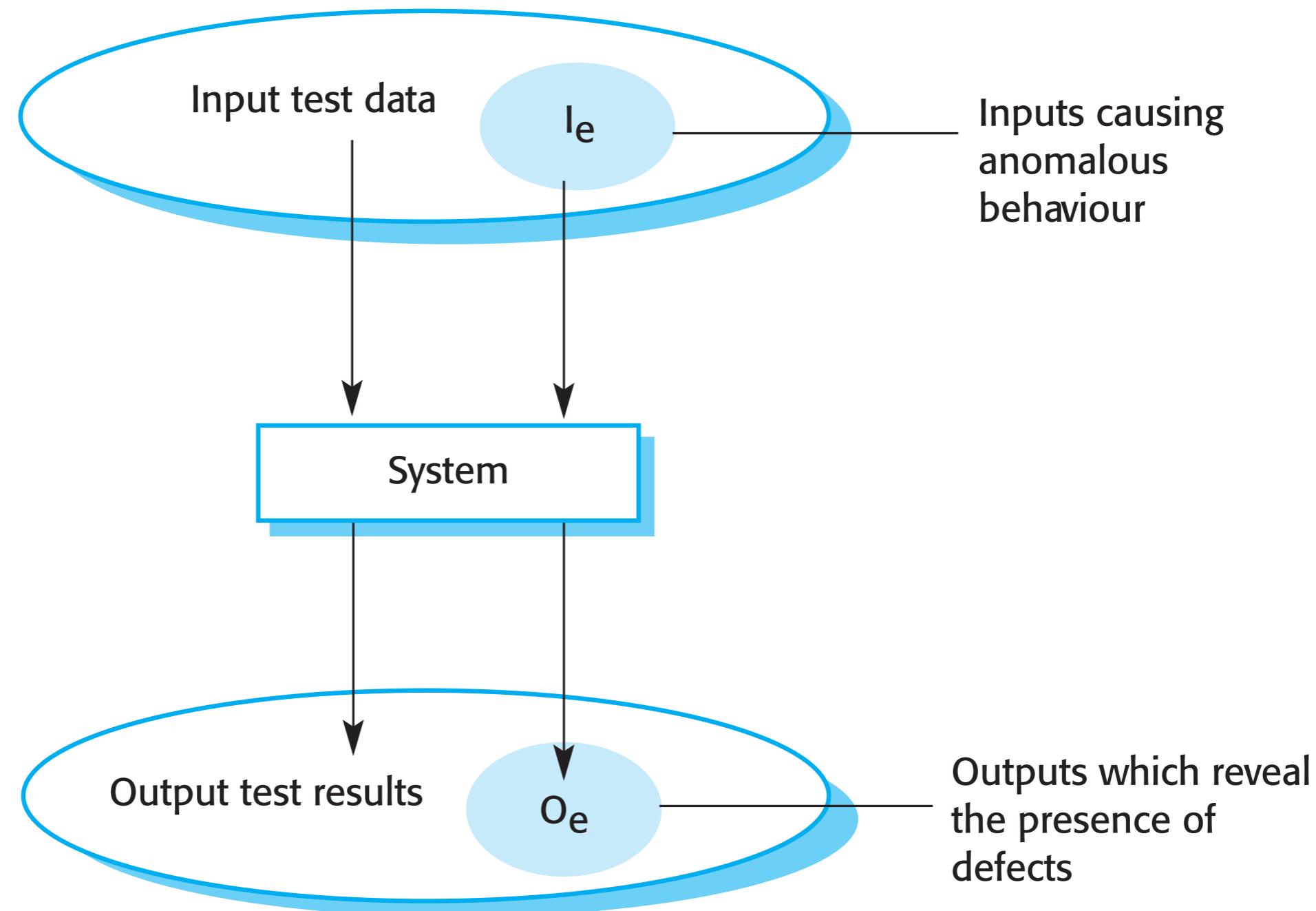
# Obiettivi del testing

- Il primo obiettivo porta ai test di convalida
- Ci si aspetta che il sistema funzioni correttamente utilizzando una serie di casi di test (*test cases*) che riflettano l' uso previsto del sistema.
- Il secondo obiettivo porta alla verifica dei difetti
- I *test cases* sono progettati per esporre i difetti. I casi di test nelle prove dei difetti possono essere deliberatamente oscuri e non devono necessariamente riflettere il modo in cui il sistema viene normalmente utilizzato.

# Obiettivi del processo di test del software

- Test di convalida
  - dimostrare allo sviluppatore e al cliente del sistema che il software soddisfa i suoi requisiti
  - Un test di successo dimostra che il sistema funziona come previsto.
- Test dei difetti
  - Rilevare guasti o difetti del software se il suo comportamento non è corretto o non è conforme alle sue specifiche.
  - Un test di successo è un test che fa sì che il sistema funzioni in modo errato e quindi espone un difetto nel sistema.

# Modello di input-output per il test di programmi



# Verifica vs Convalida (V&V)

- Verifica:
  - “Stiamo costruendo il prodotto correttamente? ”.
  - Il software deve essere conforme alle sue specifiche.
  
- Convalida:
  - “Stiamo costruendo il prodotto giusto? ”.
  - Il software dovrebbe fare ciò che l' utente ha veramente bisogno

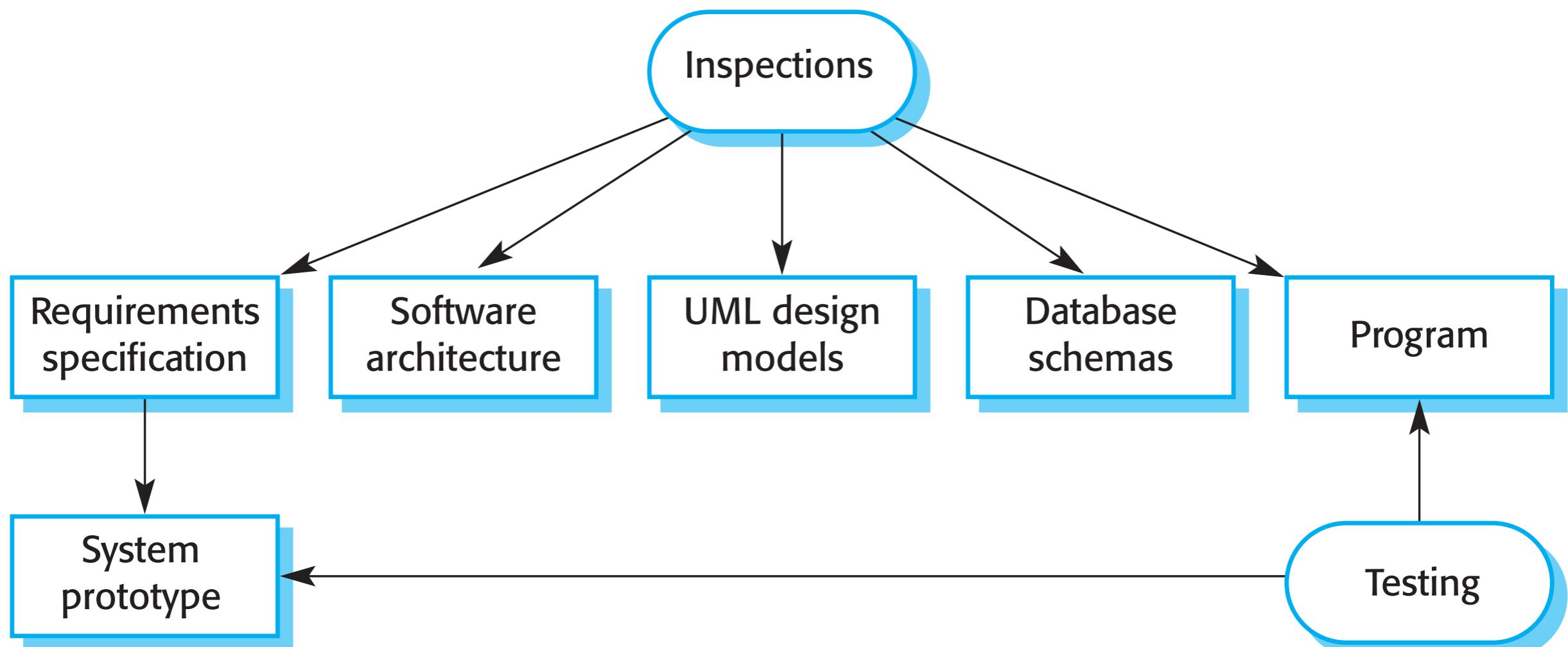
# Verifica Convalida (V&V)

- L' obiettivo di V&V è stabilire con certezza che il sistema è "adatto allo scopo" ....con un buon grado di confidenza
- Dipende dallo scopo del sistema, dalle aspettative degli utenti e dall'ambiente di marketing
  - Scopo del software
    - Il livello di fiducia dipende dalla criticità del software per un'organizzazione.
  - Aspettative degli utenti
    - Gli utenti possono avere scarse aspettative per alcuni tipi di software.
  - Ambiente di marketing
    - Ottenere un prodotto sul mercato in anticipo può essere più importante che trovare difetti nel programma.

# Ispezioni e test

- Ispezioni del software: si occupa dell' analisi della rappresentazione statica del sistema per rilevare i problemi (verifica statica)
- Può essere integrato da documenti basati su strumenti e analisi del codice.
- Test del software: si occupa di sperimentare ed osservare il comportamento del prodotto (verifica dinamica)
- Il sistema viene eseguito con i dati di prova e ne viene osservato il comportamento operativo.

# Ispezioni e test nei vari stadi del processo



# Ispezioni software

- Coinvolgono persone che esaminano la rappresentazione della fonte con l' obiettivo di scoprire anomalie e difetti.
- Le ispezioni non richiedono l' esecuzione di un sistema, pertanto possono essere utilizzate prima dell'attuazione.
- Esse possono essere applicate a qualsiasi rappresentazione del sistema (requisiti, progettazione, dati di configurazione, dati di prova, ecc.)
- Esse hanno dimostrato di essere una tecnica efficace per scoprire gli errori del programma.

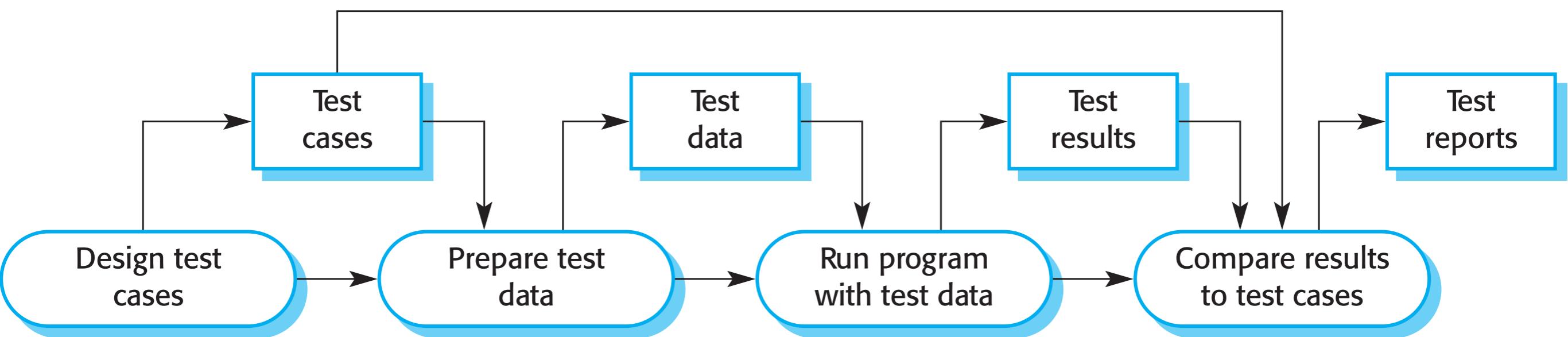
# Vantaggi delle ispezioni

- Durante il test, gli errori possono mascherare (nascondere) altri errori. Poiché l' ispezione è un processo statico, non è necessario preoccuparsi delle interazioni tra gli errori.
- Le versioni incomplete di un sistema possono essere ispezionate senza costi aggiuntivi. Se un programma è incompleto, è necessario sviluppare cablaggi di prova specializzati per testare le parti disponibili.
- Oltre alla ricerca di difetti del programma, un' ispezione può anche prendere in considerazione attributi qualitativi più ampi di un programma, come la conformità agli standard, la portabilità e la manutenibilità.

# Ispezioni e testing

- Le ispezioni e i test sono complementari e non si oppongono alle tecniche di verifica.
- Entrambi devono essere utilizzati durante il processo V & V.
- Le ispezioni possono verificare la conformità con una specifica ma non la conformità con i requisiti reali del cliente.
- Le ispezioni non possono verificare le caratteristiche non funzionali come prestazioni, usabilità, ecc.

# Modelli di processo del test del software



# Stadi del Test

- Test di sviluppo, in cui il sistema viene testato durante lo sviluppo per scoprire bug e difetti.
- Test della release, in cui un team di test separato verifica una versione completa del sistema prima che sia rilasciato agli utenti.
- Test utente, in cui gli utenti o i potenziali utenti di un sistema testano il sistema nel proprio ambiente.



Test di sviluppo

# Test di sviluppo

- I test di sviluppo comprendono tutte le attività di test che vengono svolte dal team che sviluppa il sistema.
- Test dell' unità, dove vengono testate le singole unità di programma o classi di oggetti. La verifica dell' unità deve concentrarsi sulla verifica della funzionalità di **oggetti** o **metodi**.
- Test dei componenti, in cui sono integrate più unità singole per creare componenti composti. Le prove sui componenti devono concentrarsi sulle **interfacce** dei componenti.
- Test del sistema, in cui alcuni o tutti i componenti di un sistema sono integrati e il sistema viene testato nel suo complesso. I test di sistema dovrebbero concentrarsi sulle **interazioni tra i componenti**.

# Test delle unità

- Il test dell' unità è il processo di prova dei singoli componenti isolatamente.
- Si tratta di un processo di prova dei difetti.
- Le unità possono essere:
  - Funzioni o metodi individuali all' interno di un oggetto
  - Classi di oggetti con diversi attributi e metodi
  - Componenti composti con interfacce definite utilizzate per accedervi.

# Test delle classi di oggetti

- La copertura completa del test di una classe comprende
  - Prova di tutte le operazioni associate a un oggetto
  - Impostazione e interrogazione di tutti gli attributi degli oggetti
  - Provare l' oggetto **in tutti gli stati possibili.**
- L' ereditarietà rende più difficile progettare test di classe dell' oggetto in quanto le informazioni da testare non sono localizzate.



# Interfaccia dell'oggetto Stazione Meteorologica

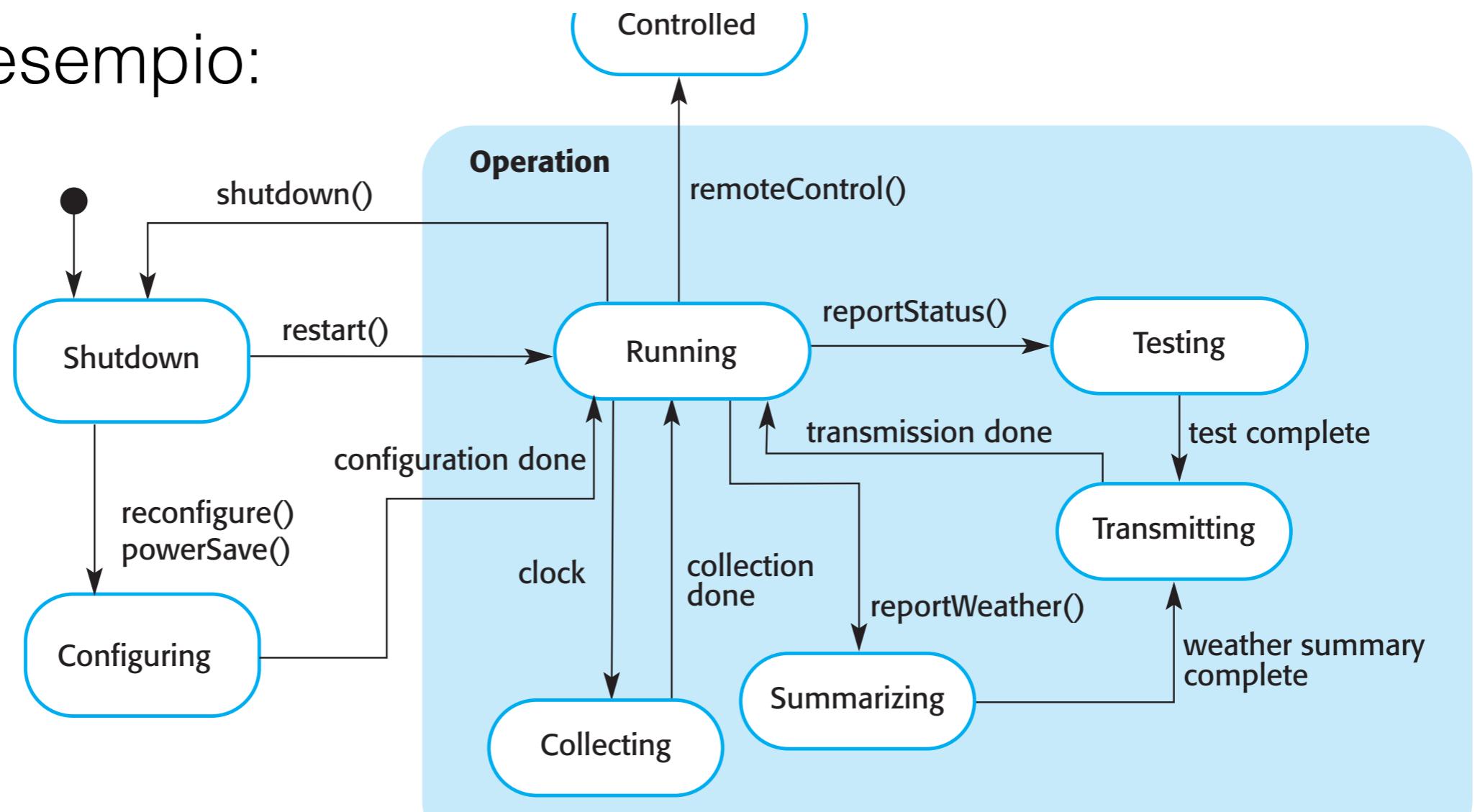
## **WeatherStation**

**identifier**

**reportWeather ()**  
**reportStatus ()**  
**powerSave (instruments)**  
**remoteControl (commands)**  
**reconfigure (commands)**  
**restart (instruments)**  
**shutdown (instruments)**

# Test della Stazione Meteorologica

- Per testare gli stati della stazione meteorologica si può usare un suo modello a stati
  - per esempio:



# Test della Stazione Meteorologica

- Necessità di definire casi di test per reportWeather, calibrare, testare, avviare e spegnere.
- Utilizzando un modello di stato, identificare le sequenze di transizioni di stato da testare e le sequenze di eventi che causano tali transizioni.
- Ad esempio:
- Spento -> Esecuzione-> Spento -> Configurazione-> Esecuzione> Test -> Trasmissione -> Esecuzione-> Raccolta-> Esecuzione-> Sintesi -> Trasmissione -> Esecuzione

# Test automatici

- Quando possibile, i test dell' unità devono essere automatizzati in modo che i test vengano eseguiti e controllati senza intervento manuale.
- Nei test automatizzati dell' unità, per scrivere ed eseguire i test del programma si fa uso di un framework di automazione di test (come JUnit).
- I framework di test delle unità forniscono classi di test generici che si estendono per creare casi di test specifici.
- Possono quindi eseguire tutti i test che avete implementato e riferire, spesso attraverso alcune GUI, sul successo di altri test.

# Componenti dei Test automatici

- Una parte di setup, dove si inizializza il sistema con il test case, vale a dire gli ingressi e le uscite previste.
- Una parte di chiamata, dove si chiama l' oggetto o il metodo da testare.
- Una parte di asserzione, in cui si confronta il risultato della chiamata con il risultato atteso.
- Se l' affermazione viene giudicata vera, il test ha avuto successo se falsa, allora è fallita.



# Scelta del Test Case per le unità

- I casi di test dovrebbero dimostrare che, se usato come previsto, il componente che si sta testando fa ciò che si suppone venga fatto.
- Se il componente presenta dei difetti, questi devono essere evidenziati da casi di prova.
- Questo porta a 2 tipi di casi di test dell' unità:
  - Il primo dovrebbe riflettere il normale funzionamento di un programma e dovrebbe mostrare che il componente funziona come previsto.
  - L' altro tipo di test dovrebbe essere basato sull' esperienza di test in caso di problemi comuni. Dovrebbe utilizzare ingressi anomali per verificare che questi siano elaborati correttamente e non causino un arresto anomalo del componente.



# Test Case

---

<b>Attributes</b>	<b>Description</b>
<code>name</code>	Name of test case
<code>location</code>	Full path name of executable
<code>input</code>	Input data or commands
<code>oracle</code>	Expected test results against which the output of the test is compared
<code>log</code>	Output produced by the test

---

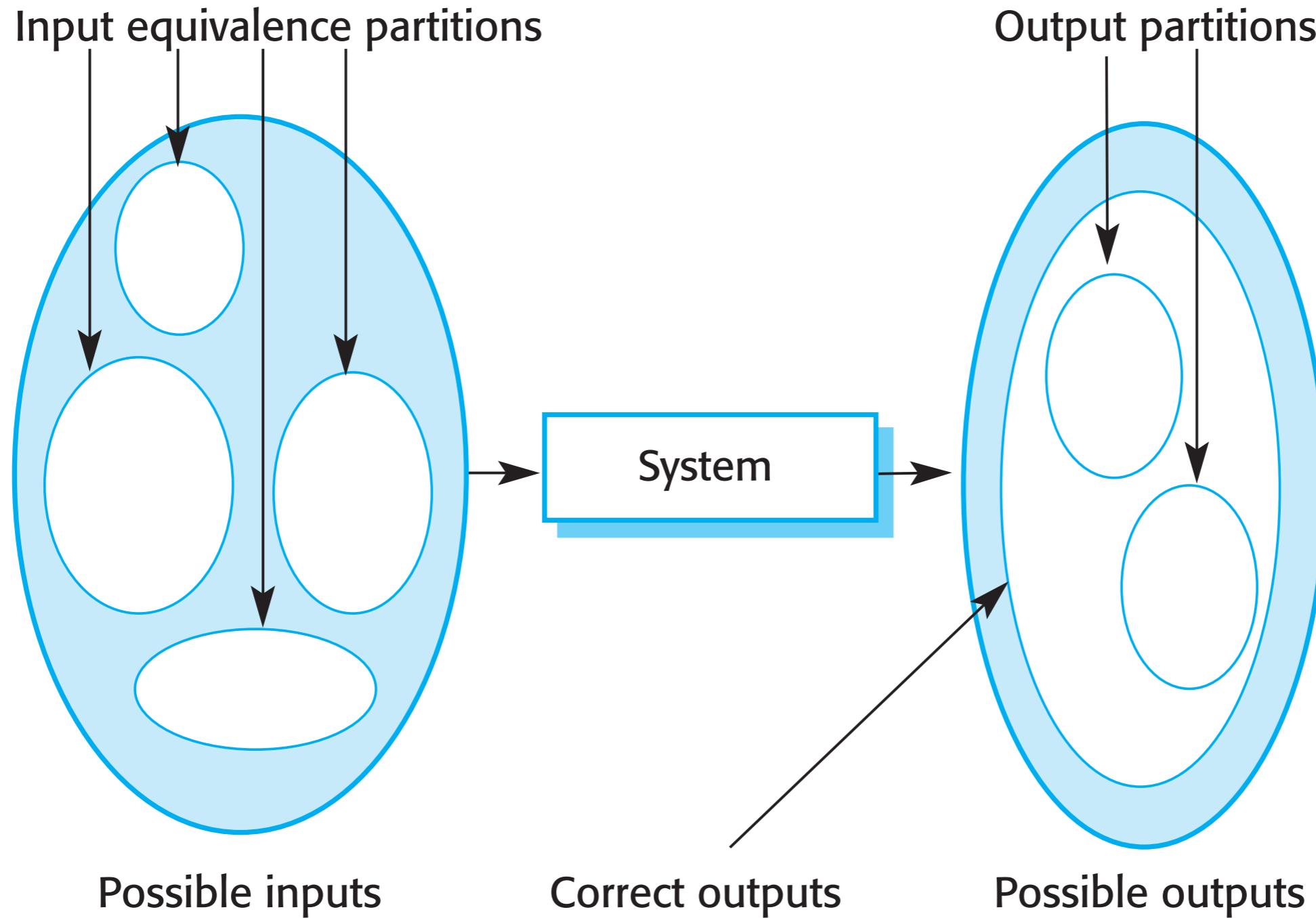
# Strategie di test

- Si possono usare due strategie per il test delle unità
- Test delle partizioni, in cui si identificano gruppi di input che hanno caratteristiche comuni e devono essere elaborati nello stesso modo.
- I test devono essere scelti all' interno di ciascuno di questi gruppi.
- Test basati su linee guida, in cui si utilizzano le linee guida per scegliere i casi di test.
- Queste linee guida riflettono l' esperienza precedente dei tipi di errori che i programmatori spesso fanno nello sviluppo dei componenti.

# Test delle partizioni

- I dati di input e i risultati di output spesso rientrano in classi diverse dove tutti i membri di una classe sono correlati.
- Ognuna di queste classi è una partizione di equivalenza o dominio in cui il programma si comporta in modo equivalente per ogni membro della classe.
- I casi di test devono essere scelti da ciascuna partizione.

# Partizioni di equivalenza



# Test delle partizioni

- Per identificare le partizioni si possono usare le specifiche del sistema o anche il codice
- black-box testing - non è necessario sapere come funziona il sistema
- white-box testing - si esamina il codice del programma per trovare altri possibili test

# Test con linee guida

- Per esempio nel caso delle sequenze:
  - Testare il software con sequenze che hanno un solo valore.
  - Utilizzare sequenze di dimensioni diverse in test diversi.
  - Prove di derivazione in modo da accedere ai primi, medi e ultimi elementi della sequenza.
  - Prova con sequenze di lunghezza zero.

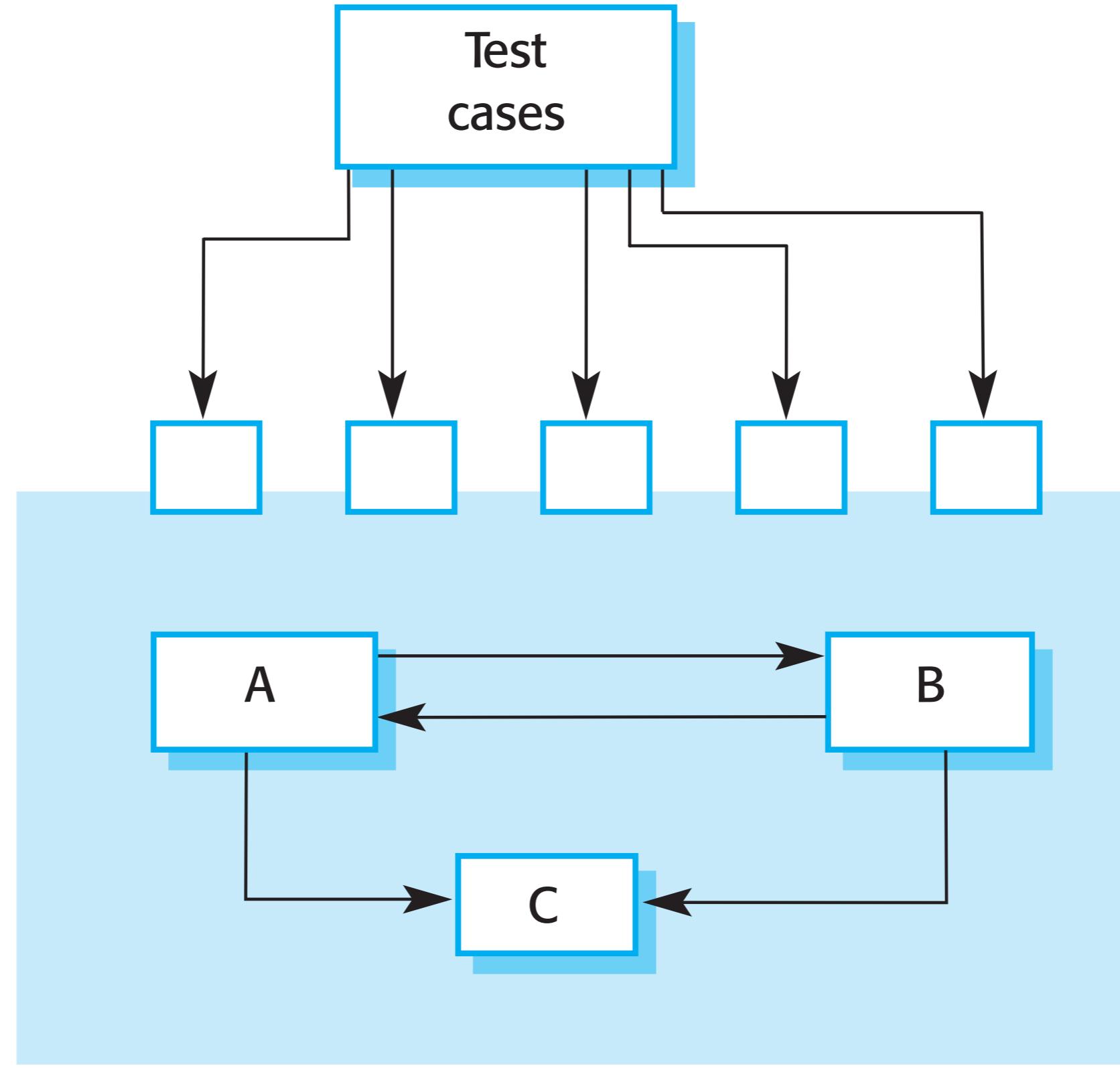
# Test con linee guida

- Scegliere gli ingressi che forzano il sistema a generare tutti i messaggi di errore
- Progettazione degli ingressi che causano un sovraccarico dei buffer di ingresso
- Ripetere più volte lo stesso ingresso o la stessa serie di ingressi
- Uscite non valide per forza da generare
- Il calcolo “per forza” risulta troppo grande o troppo piccolo.

# Test dei componenti

- I componenti software sono spesso componenti composti composti da più oggetti interagenti.
- Ad esempio, nel sistema delle stazioni meteo, il componente riconfigurazione include oggetti che trattano ogni aspetto della riconfigurazione.
- Si accede alla funzionalità di questi oggetti attraverso l'interfaccia componente definita.
- Il test dei componenti composti dovrebbe pertanto concentrarsi sulla dimostrazione che l' interfaccia dei componenti si comporta secondo le sue specifiche.
- Si può supporre che i test unitari sui singoli oggetti all' interno del componente siano stati completati.

# Test dell'interfaccia



# Test dell' Interfaccia

- Gli obiettivi sono individuare i guasti dovuti a errori di interfaccia o ipotesi errate sulle interfacce.
- Tipi di interfaccia
  - Interfacce parametri - I dati sono passati da un metodo o procedura all' altra.
  - Interfacce di memoria condivise -Il blocco di memoria è condiviso tra procedure o funzioni.
  - Interfacce procedurali - Il sottosistema comprende una serie di procedure che devono essere chiamate da altri sottosistemi.
  - Interfacce a passaggio di messaggi - Sottosistemi richiedono servizi da altri sottosistemi

# Errori nelle interfacce

- Uso improprio dell' interfaccia
  - Un componente chiamante chiama un altro componente e commette un errore nell' utilizzo della sua interfaccia, ad esempio parametri nell' ordine sbagliato.
- Incomprensione dell' interfaccia
  - Un componente call incorpora ipotesi sul comportamento del componente chiamato che sono errate.
- Errori di tempistica
  - La componente chiamante e quella chiamata funzionano a diverse velocità e si accede a informazioni superate.

# Guideline per il test delle interfacce

- Prove di progettazione in modo che i parametri di una procedura chiamata siano alle estremità della loro gamma.
- Verificare sempre i parametri del puntatore con puntatori nulli.
- Prove di progettazione che causano il guasto del componente.
- Utilizzare test di stress nei sistemi di trasmissione dei messaggi.
- Nei sistemi a memoria condivisa, variare l' ordine di attivazione dei componenti.

# Test di sistema

- Il collaudo del sistema durante lo sviluppo comporta l'integrazione di componenti per creare una versione del sistema e poi testare il sistema integrato.
- Il focus nel test di sistema è testare le interazioni tra i componenti.
- Il test di sistema verifica che i componenti siano compatibili, interagiscano correttamente e trasmettano i dati corretti al momento giusto attraverso le loro interfacce.
- I test di sistema testano il comportamento emergente di un sistema.

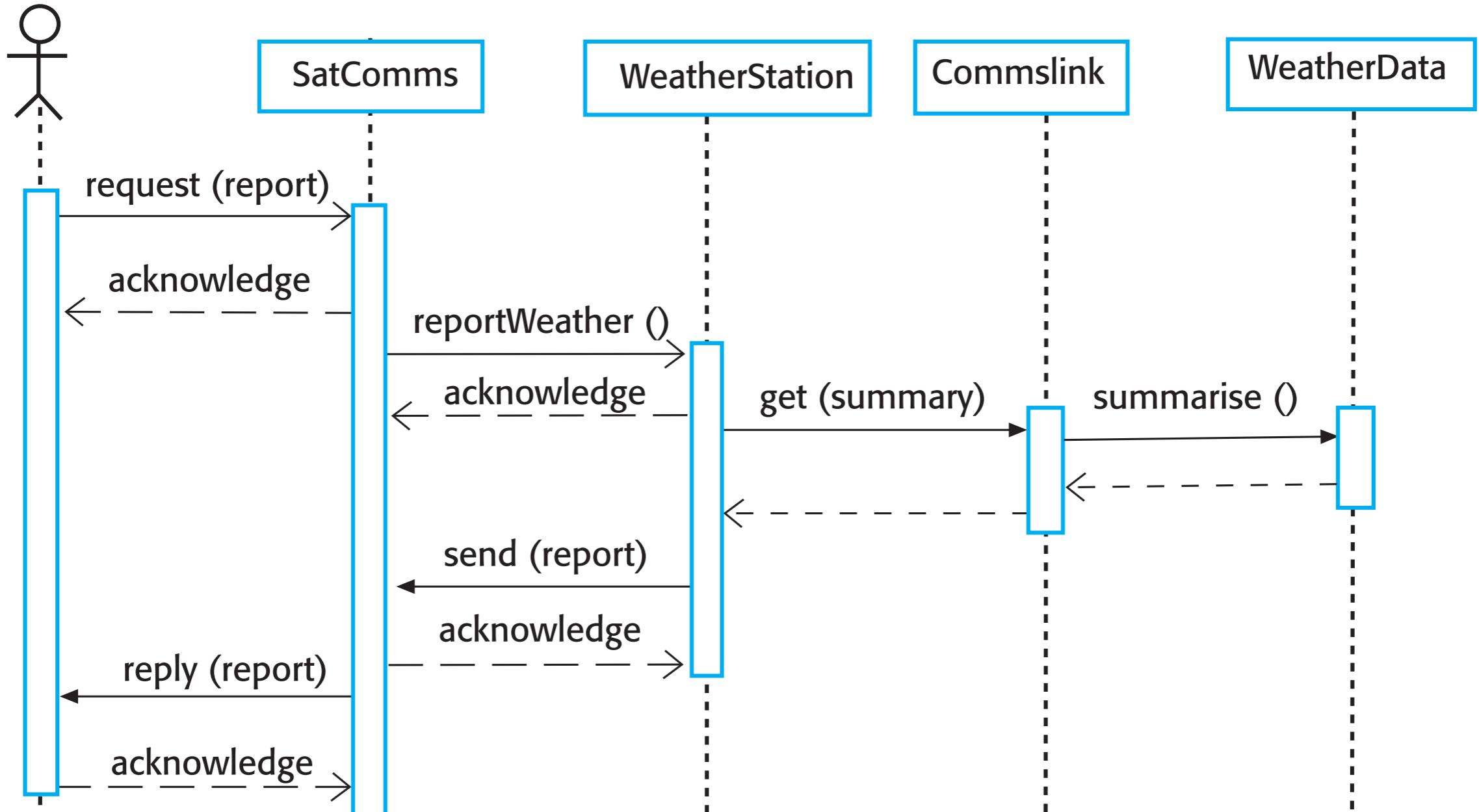
# Test di sistema e dei componenti

- Durante il collaudo dei sistemi, i componenti riutilizzabili che sono stati sviluppati separatamente e i sistemi pronti all' uso possono essere integrati con componenti di nuova concezione. Il sistema completo viene quindi testato.
- In questa fase possono essere integrati componenti sviluppati da membri o sottogruppi diversi del team. Il test di sistema è un processo collettivo piuttosto che individuale.
- In alcune aziende, i test di sistema possono coinvolgere un team di test separato senza il coinvolgimento di progettisti e programmatore.

# Use-case testing

- I casi d' uso sviluppati per identificare le interazioni di sistema possono essere utilizzati come base per i test di sistema.
- Ogni caso d' uso di solito coinvolge diversi componenti del sistema, quindi testare il caso d'uso costringe queste interazioni a verificarsi.
- I diagrammi di sequenza associati al case d' uso documentano i componenti e le interazioni che vengono testati.

information system



# Test case a partire dai sequence

- L'inserimento di una richiesta di relazione deve essere accompagnato da un riscontro. Alla fine, una relazione dovrebbe essere restituita dalla richiesta.
- È necessario creare dati riassunti che possano essere utilizzati per verificare che il report sia correttamente organizzato.
- Una richiesta di input per un report a WeatherStation comporta la generazione di un report riassuntivo.
- Può essere testato creando dati grezzi corrispondenti al riepilogo preparato per il test di SatComms e controllando che l'oggetto WeatherStation produca correttamente questo riepilogo. Questi dati grezzi vengono utilizzati anche per testare l'oggetto WeatherData.

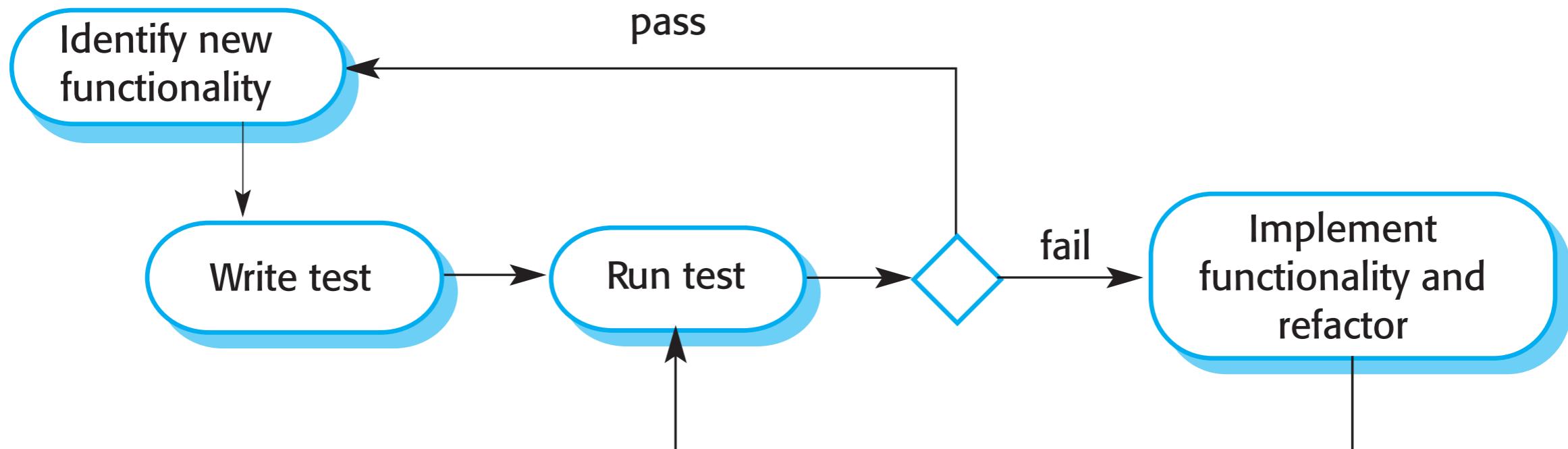
# Politiche di testing

- È impossibile effettuare test di sistema completi, pertanto è impossibile elaborare politiche di test che definiscano la copertura richiesta per i test di sistema.
- Esempi di politiche sperimentali:
  - Tutte le funzioni del sistema a cui si accede attraverso i menu devono essere testate.
  - È necessario testare le combinazioni di funzioni (ad es. formattazione del testo) a cui si accede tramite lo stesso menu.
  - In caso di inserimento da parte dell' utente, tutte le funzioni devono essere testate sia con l' inserimento corretto che con quello errato.

# Sviluppo guidato dai test

- Lo sviluppo test-driven development (TDD) è un approccio allo sviluppo di programmi in cui è possibile eseguire test e sviluppare codice contemporaneamente.
- I test sono scritti prima del codice e passare i test è il motore critico di sviluppo.
- Si sviluppa codice incrementale, insieme a un test per tale incremento. Non si passa all' incremento successivo finché il codice sviluppato non supera il test.
- Il TDD è stato introdotto come parte di metodi agili come la Programmazione Estrema. Tuttavia, può essere utilizzato anche in processi di sviluppo pianificati.

# Sviluppo guidato dai test



# Sviluppo guidato dai test

- Iniziare identificando l' incremento di funzionalità necessario. Normalmente questo dovrebbe essere piccolo e attuabile in poche righe di codice.
- Scrivere un test per questa funzionalità e implementarlo come test automatico.
- Eseguire il test, insieme a tutti gli altri test che sono stati implementati. Inizialmente, non è stata implementata la funzionalità in modo che il nuovo test fallisca.
- Implementare la funzionalità e ripetere il test.
- Una volta che tutti i test sono stati eseguiti con successo, si passa all' implementazione del successivo chunk di funzionalità.

# Benefici dello sviluppo test-driven

- Codice di copertura
  - Ogni segmento di codice che si scrive ha almeno un test associato, quindi tutto il codice scritto ha almeno un test.
- Test di regressione
  - Una serie di test viene sviluppata in modo incrementale man mano che viene sviluppato un programma.
- Debug semplificato
  - Quando un test fallisce, dovrebbe essere chiaro dove sta il problema. Il nuovo codice scritto deve essere controllato e modificato.
- Documentazione del sistema
  - I test stessi sono una forma di documentazione che descrive cosa dovrebbe fare il codice.

# Test di regressione

- Il test di regressione testa il sistema per verificare che le modifiche non abbiano "rotto" il codice operativo precedente.
- In un processo di test manuale, le prove di regressione sono costose ma, con le prove automatiche, sono semplici e semplici. Tutti i test vengono ripetuti ogni volta che si modifica il programma.
- Il test di regressione controlla quali modifiche non abbiano introdotto nuovi bug nel sistema e che il nuovo codice interagisca con il precedente nel modo voluto.

# Release testing

- Il test della release è il processo di prova di una particolare release di un sistema destinato ad essere utilizzato al di fuori del team di sviluppo.
- L' obiettivo primario del processo di test di release è convincere il fornitore del sistema che il sistema è abbastanza buono per l' uso.
- Le prove di release, pertanto, devono dimostrare che il sistema offre le sue funzionalità, prestazioni e affidabilità specifiche e che non presenta problemi durante il normale utilizzo.
- Le prove di rilascio sono solitamente un processo di prova in scatola nera, in cui le prove sono derivate solo dalle specifiche del sistema.

# Test delle release e di sistema

- La prova di rilascio è una forma di prova del sistema.
- Differenze importanti:
  - Un team separato che non è stato coinvolto nello sviluppo del sistema, dovrebbe essere responsabile delle prove di rilascio.
  - Il test di sistema da parte del team di sviluppo dovrebbe concentrarsi sulla scoperta di bug nel sistema (defect test). L' obiettivo delle prove di rilascio è quello di verificare che il sistema soddisfi i suoi requisiti e sia sufficientemente valido per l' uso esterno (prove di convalida).

# Test basato sui requisiti

- Principio generale: i requisiti devono essere tentabili
- i requisiti devono essere scritti in modo che possa essere progettato un apposito test
- Le prove basate sui requisiti comprendono l' esame di ciascun requisito e lo sviluppo di una o più prove.
- Requisiti del sistema di manutenzione:
  - *Se un paziente è notoriamente allergico a un determinato medicinale, la prescrizione di tale medicinale deve dare luogo a un messaggio di avvertenza per l' utente del sistema.*
  - *Se un prescrivente sceglie di ignorare un avviso di allergia, deve motivare il motivo per cui tale avviso è stato ignorato.*

# Requirements Tests

- Creare una cartella clinica del paziente senza allergie note. Prescrivere i farmaci per le allergie note. Verificare che il sistema non invii un messaggio di avviso.
- Creare una cartella clinica del paziente con un' allergia nota. Prescrivere il farmaco che il paziente è allergico e controllare che l'avviso sia emesso dal sistema.
- Creare una cartella clinica in cui registrare le allergie a due o più farmaci. Prescrivere entrambi questi farmaci separatamente e verificare che l'avvertimento corretto per ogni farmaco sia rilasciato.
- Prescrivere due farmaci a cui il paziente è allergico. Verificare che siano stati emessi correttamente i due avvisi.
- Prescrivere un farmaco che faccia in modo che il sistema emetta un avvertimento e non considerare quell'avvertimento. Verificare che il sistema richieda all' utente di fornire informazioni che spieghino perché l'avviso è stato annullato.



# Test degli scenari

George è un' infermiera specializzata nella salute mentale. Una delle sue responsabilità è quella di visitare i pazienti a casa per verificare che il loro trattamento sia efficace e che non soffrano di effetti collaterali dei farmaci.

In un giorno di visite domiciliari, George si collega al sistema Mentcare e lo utilizza per stampare il suo programma di visite domiciliari per quel giorno, insieme a informazioni sintetiche sui pazienti da visitare. Chiede che i record per questi pazienti siano scaricati sul suo computer portatile. Viene chiesto per la sua frase chiave di crittografare i record sul laptop.

Uno dei pazienti che visita è Jim, che viene curato con farmaci per la depressione. Jim sente che il farmaco lo sta aiutando ma crede che abbia l' effetto collaterale di tenerlo sveglio durante la notte. George guarda il disco di Jim e viene richiesto per la sua frase chiave per decrittografare il disco. Controlla il farmaco prescritto e ne interroga gli effetti collaterali. L'assenza di sonnolenza è un effetto collaterale noto, così egli nota il problema nel record di Jim e suggerisce che lui visita la clinica per avere il suo farmaco cambiato. Jim è d' accordo così George entra in un prompt a chiamarlo quando torna alla clinica per fissare un appuntamento con un medico. George conclude la consultazione e il sistema ri-cripta il record di Jim.

Dopo aver terminato le sue consultazioni, George ritorna alla clinica e carica i documenti dei pazienti visitati nella banca dati. Il sistema genera una lista di chiamate per George di quei pazienti che egli deve contattare per informazioni di follow-up e fissare appuntamenti in clinica.

# Caratteristiche testate dallo scenario

- Autenticazione tramite login con il sistema.
- Download e caricamento di specifiche cartelle paziente su un computer portatile.
- Programmazione delle visite domiciliari.
- Crittografia e decodifica delle cartelle cliniche dei pazienti su un dispositivo mobile.
- Registrare il recupero e la modifica.
- Collegamenti con la banca dati sulle droghe che mantiene informazioni sugli effetti collaterali.
- Il sistema per la richiesta di chiamata.

# Test delle prestazioni

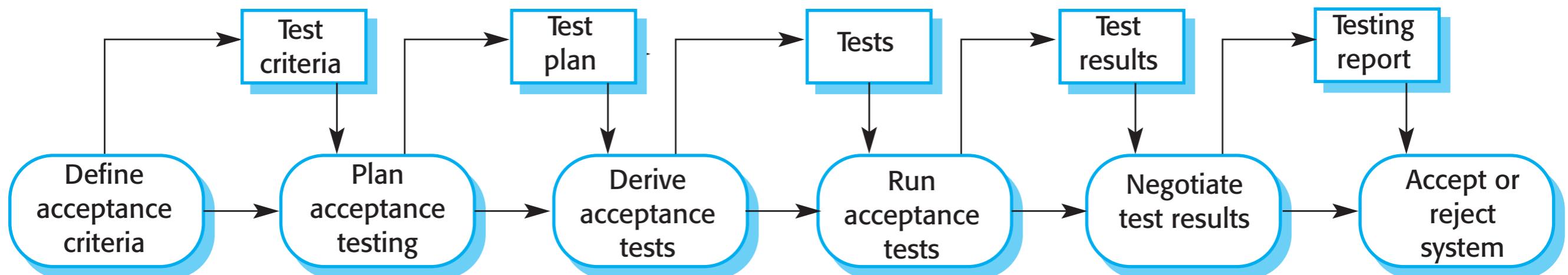
- Parte dei test di rilascio può comprendere la verifica delle proprietà emergenti di un sistema, quali le prestazioni e l' affidabilità.
- I test devono riflettere il profilo di utilizzo del sistema.
- I test delle prestazioni solitamente prevedono la pianificazione di una serie di test in cui il carico aumenta costantemente fino a quando le prestazioni del sistema non diventano inaccettabili.
- Lo **stress test** è una forma di prova delle prestazioni in cui il sistema è deliberatamente sovraccaricato per testare il suo comportamento in caso di guasto.

# Test degli utenti

- Il test dell' utente o del cliente è una fase del processo di test in cui gli utenti o i clienti forniscono input e consulenza sul test del sistema.
- Il test dell' utente è essenziale, anche quando sono stati effettuati test completi del sistema e di rilascio.
- Ciò è dovuto al fatto che le influenze dell'ambiente di lavoro dell' utente hanno un effetto importante sull'affidabilità, le prestazioni, l' usabilità e la robustezza di un sistema. Questi non possono essere replicati in un ambiente di prova.

# Test degli utenti

- Test alfa
  - Gli utenti del software collaborano con il team di sviluppo per testare il software presso il sito dello sviluppatore.
- Test beta
  - Una release del software è messa a disposizione degli utenti per permettere loro di sperimentare e sollevare problemi che scoprono con gli sviluppatori di sistema.
- Prove di accettazione
  - I clienti testano un sistema per decidere se è pronto o meno per essere accettato dagli sviluppatori di sistema e distribuito nell'ambiente del cliente.



# Test di accettazione

- Definire criteri di accettazione
- Test di accettazione del piano
- Prove di accettazione derivate
- Eseguire i test di accettazione
- Negoziare i risultati dei test
- Sistema di respingimento/accettazione



# Key Points

- I test possono solo mostrare la presenza di errori in un programma. Non possono dimostrare che non vi siano guasti.
- I test di sviluppo sono di competenza del team di sviluppo software. Un team separato dovrebbe essere responsabile di testare un sistema prima che venga rilasciato ai clienti.
- I test di sviluppo includono test unitari, in cui si testano singoli oggetti e metodi di test dei componenti in cui si testano gruppi correlati di oggetti e test di sistema, in cui si testano sistemi parziali o completi.

# Key Points

- Durante il test del software, si dovrebbe cercare di “rompere” il software utilizzando esperienza e linee guida per scegliere i tipi di casi di test che sono stati efficaci nel scoprire i difetti in altri sistemi.
- Dove possibile, dovreste scrivere test automatizzati. I test sono integrati in un programma che può essere eseguito ogni volta che viene apportata una modifica a un sistema.
- Lo sviluppo test-first è un approccio allo sviluppo in cui i test sono scritti prima del codice da testare.
- Il test dello scenario consiste nell' inventare uno scenario d' uso tipico e nell' utilizzarlo per ricavare i casi di test.
- Il test di accettazione è un processo di test dell' utente in cui l'obiettivo è decidere se il software è abbastanza buono per essere distribuito e utilizzato nel suo ambiente operativo.