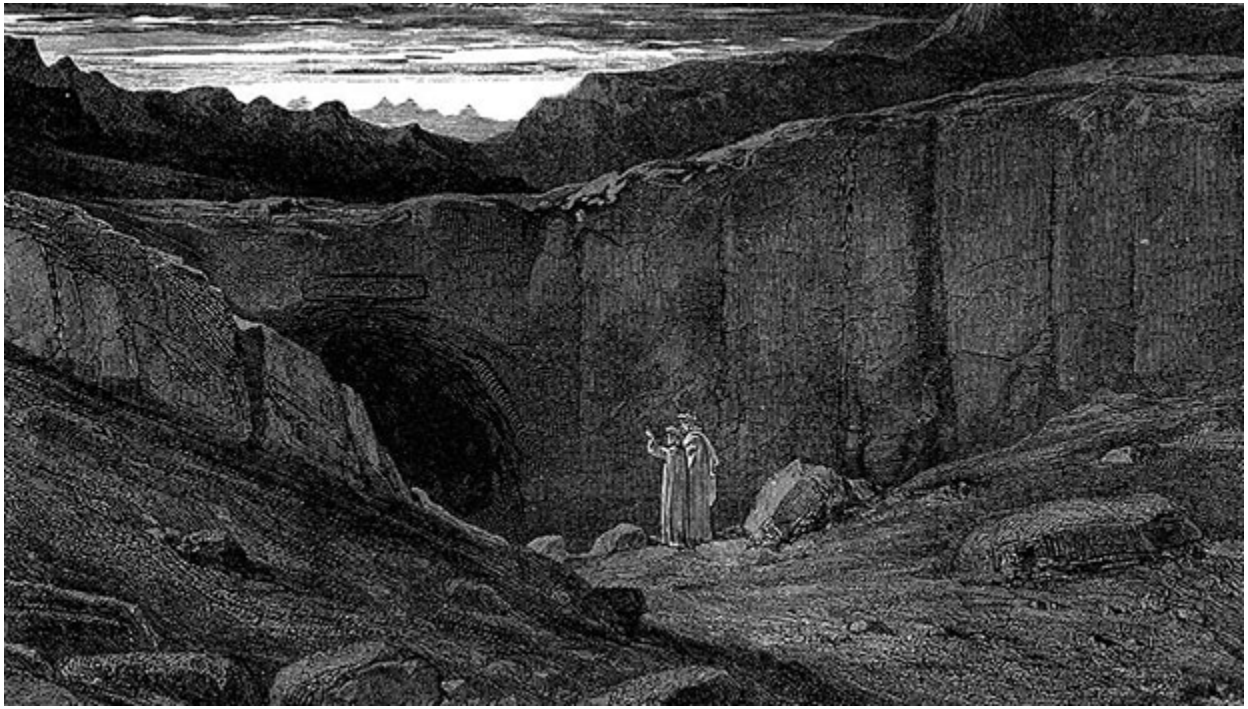
Introduzione alla fase e ai concetti di verifica e validazione

Roberta Gori, Laura Semini, thanks to Mauro Pezzè & Michal Young
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

Lasciate ogni speranza o voi che entrate



- Il problema della verifica di correttezza è indecidibile

Un risultato fondamentale I

- Nel 1937 Alan Turing ha dimostrato che alcuni problemi non possono essere risolti da un algoritmo (programma)

- dim per assurdo: supponiamo esista un algoritmo, che prende in ingresso un qualsiasi altro algoritmo e input d e stabilisca se l'algoritmo termina su d (restituendo true) o se non termina (restituendo false).

Problema della terminazione

```
// halts() restituisce true se il suo input termina, false altrimenti
boolean C(a, d):
    return halts(a(d));
```

Posso chiamare C(a,a)

```
// loop() è una funzione che non termina
boolean K(a):
    if C(a,a) loop();
    else return false;
```

chiamo `K(K)`

questo algoritmo termina, restituendo il valore `false`, solo se l'algoritmo `K` con input `K` non termina.

`K(K)` termina se e solo se `K(K)` non termina. Contraddizione!

Non esiste un algoritmo C !

Un risultato fondamentale II

- In particolare,dobbiamo concludere che non esiste un programma P che per ogni programma e input D, dice se il programma Q sull'input D termina o no (Halting Problem).

- Purtroppo non e' solo un risultato teorico: quasi tutte le proprieta' interessanti dei programmi incorporano l'halting problem.

Indecidibilità

- Quindi, non esiste alcun programma P che prende in input altri programmi e PER OGNUNO di questi decide in tempo finito se è corretto o meno
  - Esistono programmi che è possibile dimostrare corretti in tempo finito

```
public void printHW(){
            System.out.println("Hello, World");
}
```

  - Ne esistono altri per cui ciò non è possibile.

E' indecidibile per esempio:

- Dire se un qualsiasi programma C va in ciclo infinito su un certo ingresso o termini la sua esecuzione

```
public void printC(myHome Home) {
    Calzini calzini = myhome.calzini;
    while (not appaiati(calzini) ) calzini=appaia(calzini);
    System.out.println("Hello, World");
    }
```

E' anche p. es. indecidibile:

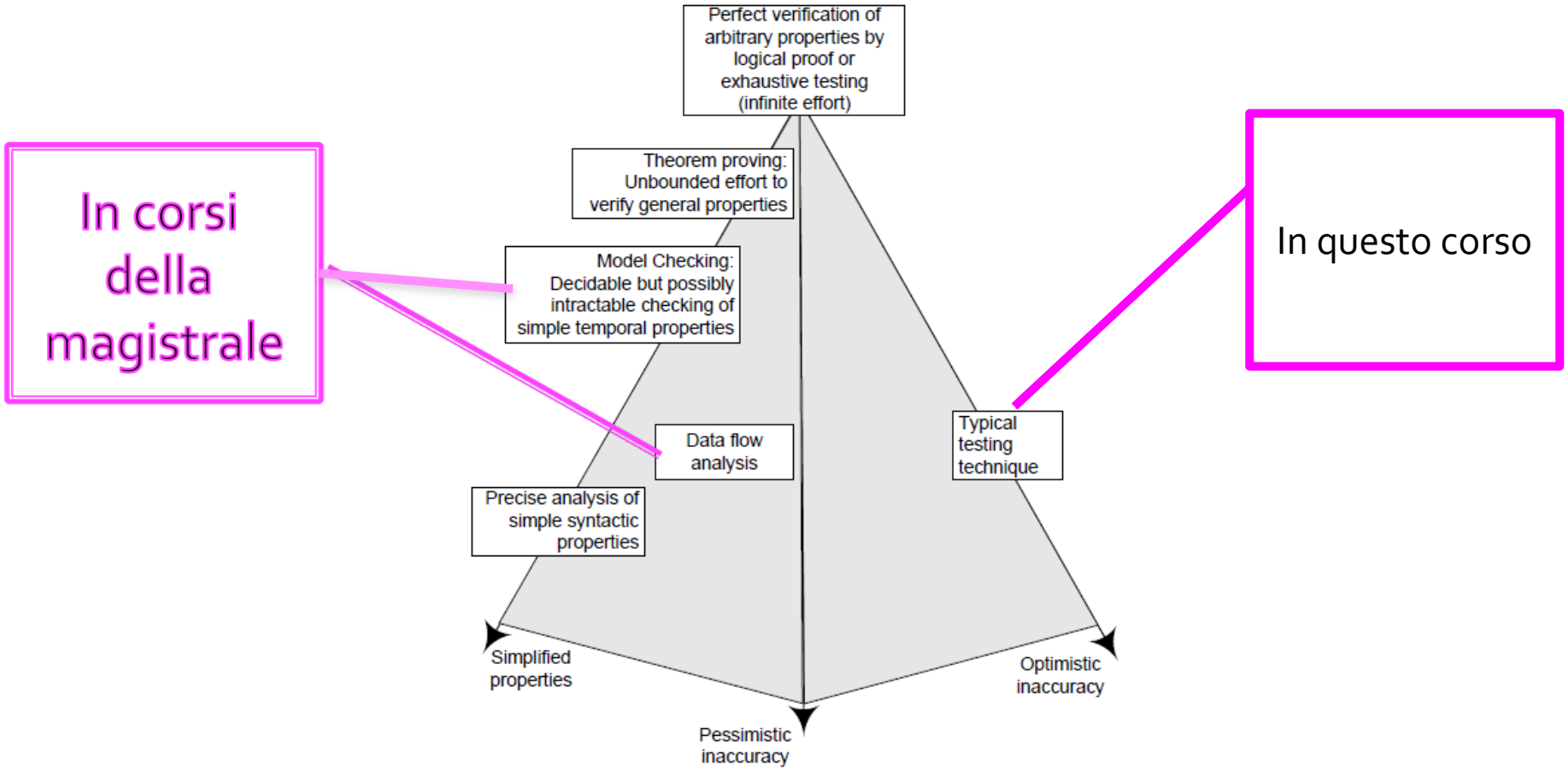
- Dire se due programmi C producono lo stesso risultato in corrispondenza degli stessi dati di ingresso
  - Per esempio i due metodi visti

```
public void printHW(myHome Home){
              System.out.println("Hello, World");
   }

public void printC(myHome Home) {
        Calzini calzini = myhome.get(calzini);
        while (not appaiati(calzini) ) calzini=appaia(calzini);
        System.out.println("Hello, World");
  }
```

Ciò premesso

- Vediamo cosa possiamo fare



Roadmap

- Concetti e terminologia
- Visualizzare il "quadro generale" della qualità del software nel contesto di un progetto di sviluppo software e organizzazione:
  - attività di verifica e di validazione (V&V: verification and validation) del software
  - la selezione e la combinazione di attività di V&V all'interno di un processo di sviluppo software.

Attività di verifica e di convalida

- L'attività di verifica può variare tra:
  - verifica altamente ripetitiva di prodotti non critici per i mercati di massa
  - prodotti altamente personalizzati o altamente critici
- L'appropriatezza di una attività di verifica dipende
  - dalla disciplina ingegneristica
  - dal processo di costruzione
  - dal prodotto finale
  - dai requisiti di qualità

Il sw ha alcune caratteristiche che rendono V & V particolarmente difficile

- requisiti di qualità diversi
- il sw è sempre in evoluzione
- distribuzione irregolare dei guasti
- non linearità, esempio:
  - Se un ascensore può trasportare un carico di 1000 kg, può anche trasportare qualsiasi carico minore:
  - se una procedura ordina correttamente un set di 256 elementi, potrebbe non riuscire su un set di 255 o 53 o 12 elementi, nonché su 257 o 1023.

Dipendenza dai linguaggi usati

- nuovi approcci di sviluppo possono introdurre nuovi tipi di errori
  - deadlock o race conditions per il software distribuito
  - problemi dovuti al polimorfismo o al binding dinamico nel software object-oriented

Varietà di approcci: non ci sono ricette fisse

- I progettisti di test devono:
  - scegliere e programmare la giusta combinazione di tecniche
    - per raggiungere il livello richiesto di qualità
    - entro i limiti di costo
  - progettare una soluzione specifica che si adatta
    - al problema
    - ai requisiti
    - all'ambiente di sviluppo

Non ci sono ricette fisse ma fatevi guidare da queste cinque domande

1. Quando iniziare verifica e convalida? Quando sono complete?
2. Quali tecniche applicare?
3. Come possiamo valutare se un prodotto è pronto per essere rilasciato?
4. Come possiamo controllare la qualità delle release successive?
5. Come può essere migliorato il processo di sviluppo?

1 Quando iniziare verifica e convalida? Quando sono complete?

- Il testing non è una fase finale dello sviluppo software
  - L'esecuzione dei test è solo una piccola parte del processo di verifica e convalida
- V & V iniziano non appena decidiamo di creare un prodotto software, o anche prima
- V & V durano molto oltre la consegna dei prodotti
  - per tutto il tempo in cui il software è in uso
  - per far fronte alle evoluzioni e agli adattamenti alle nuove condizioni

1: Quando iniziare verifica e convalida?

- Dallo studio di fattibilità
  - Lo studio di fattibilità di un nuovo progetto deve tener conto delle qualità richieste e dell'impatto sul costo complessivo
  - In questa fase, le attività correlate alla qualità comprendono
    - analisi del rischio
    - definizione delle misure necessarie per valutare e controllare la qualità in ogni stadio di sviluppo
    - valutazione dell'impatto di nuove funzionalità e nuovi requisiti di qualità
    - valutazione economica delle attività di controllo della qualità: costi e tempi di sviluppo

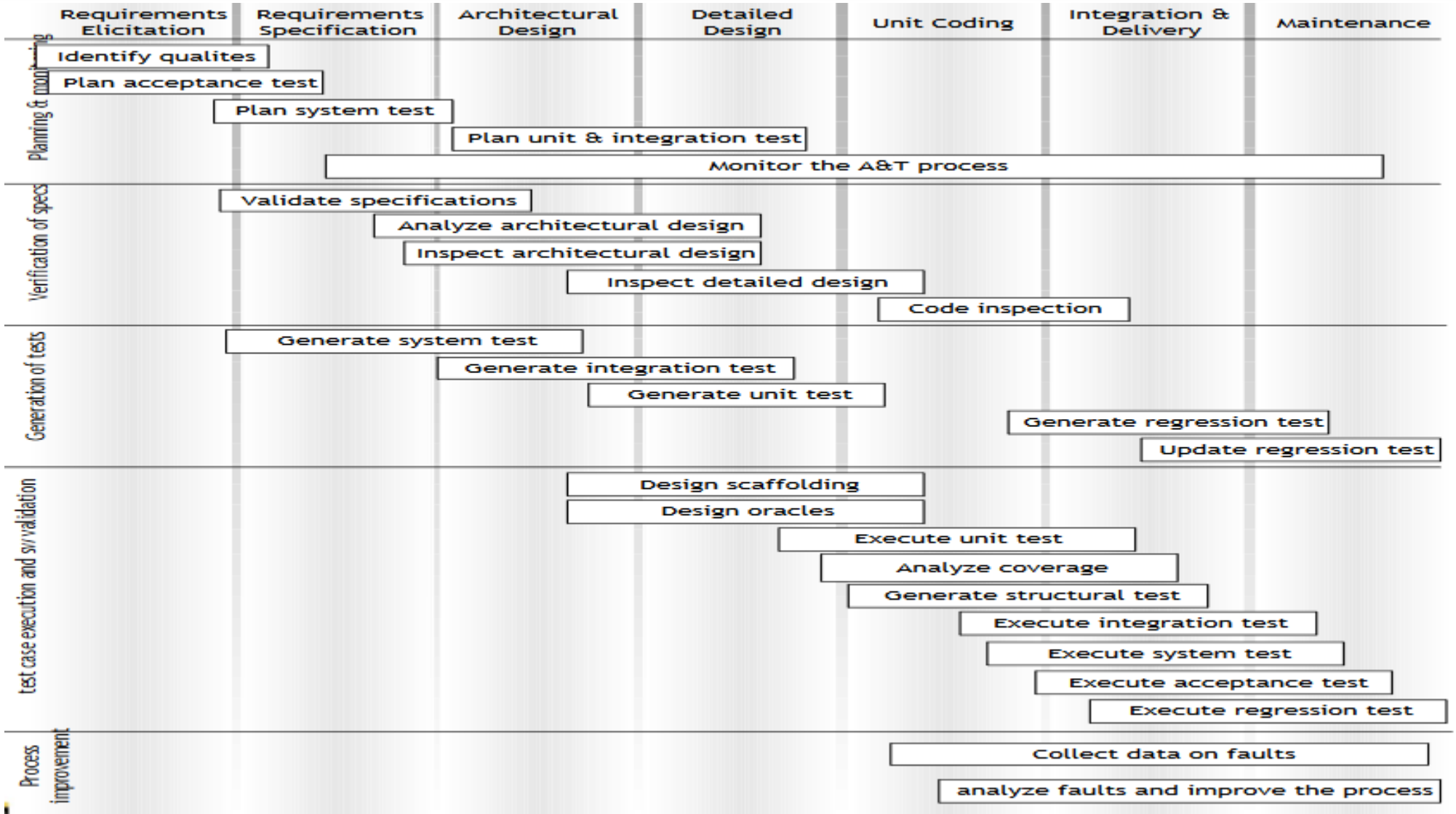
1: V&V dopo il rilascio

- Le attività di manutenzione comprendono:
  - analisi delle modifiche ed estensioni,
  - generazione di nuove suite di test per le funzionalità aggiuntive,
  - riesecuzione dei test per verificare la non regressione delle funzionalità del software dopo le modifiche e le estensioni
  - rilevamento e analisi dei guasti

2: Quali tecniche applicare?

- Nessuna singola tecnica di analisi e testing (A & T) è sufficiente per tutti gli scopi.
- Le principali ragioni per combinare diverse tecniche sono:
  - Efficacia per diverse classi di errori: analisi statica invece di test per le race conditions
  - Applicabilità in diverse fasi del processo di sviluppo, per esempio: ispezione per la convalida dei requisiti iniziali
  - Differenze negli scopi. Esempio: test statistico per misurare l'affidabilità
  - Compromessi in termini di costo e affidabilità: usare tecniche costose solo per requisiti di sicurezza

Tecniche diverse in fasi diverse



3. Come possiamo valutare se un prodotto è pronto per essere rilasciato?

- Alcune misure di **dependability** :
  - La **disponibilità** misura la qualità di un sistema in termini di tempo di esecuzione rispetto al tempo in cui il sistema è giù
  - Il **tempo medio tra i guasti** (MTBF) misura la qualità di un sistema in termini di tempo tra un guasto e il successivo
  - **L'affidabilità** indica la percentuale di operazioni che terminano con successo

3. Come possiamo valutare se un prodotto è pronto per essere rilasciato?
Le misure vanno ben definite (es. affidabilità)

- Applicazione e-shop realizzata con 100 operazioni
  - Il software funziona correttamente fino al punto in cui viene indicata una carta di credito: nel 50% dei casi viene addebitato l'importo sbagliato.

- Qual è l'affidabilità del sistema?
  - Se contiamo la percentuale di operazioni corrette , solo una operazione su 100 fallisce: il sistema è affidabile al 99%
  - Se contiamo le sessioni, solo il 50% affidabile

3. Come possiamo valutare se un prodotto è pronto per essere rilasciato? Alfa e beta test

- Alfa test:
  - test eseguiti dagli sviluppatori o dagli utenti in ambiente controllato, osservati dall'organizzazione dello sviluppo
- Beta test:
  - test eseguiti da utenti reali nel loro ambiente, eseguendo attività reali senza interferenze o monitoraggio ravvicinato

4. Come possiamo controllare la qualità delle release successive?

- Attività dopo la consegna
  - test e analisi del codice nuovo e modificato
  - riesecuzione dei test di sistema
  - memorizzazione di tutti i bug trovati
  - test di regressione

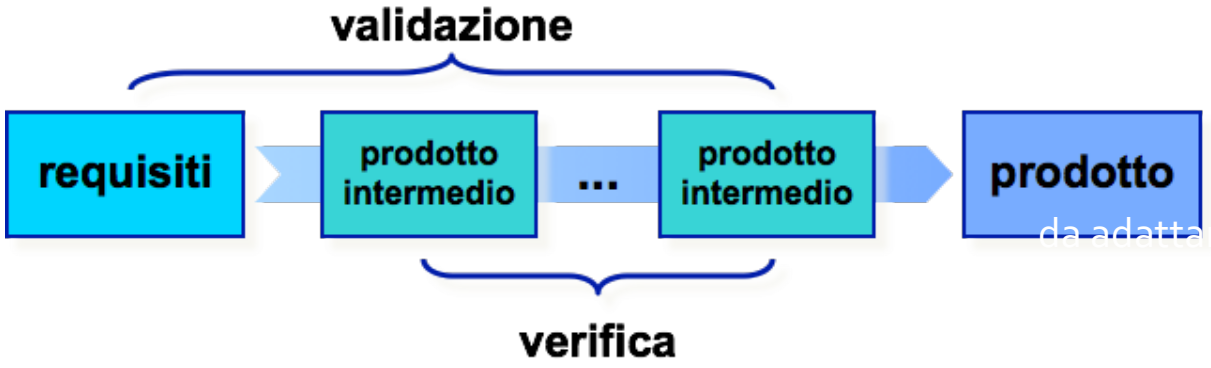
    Quasi automatico
  - distinzione tra "major" e "minor" revisions
    - 2.0 vs 1.4
    - 1.5 vs 1.4

5. Come può essere migliorato il processo di sviluppo?

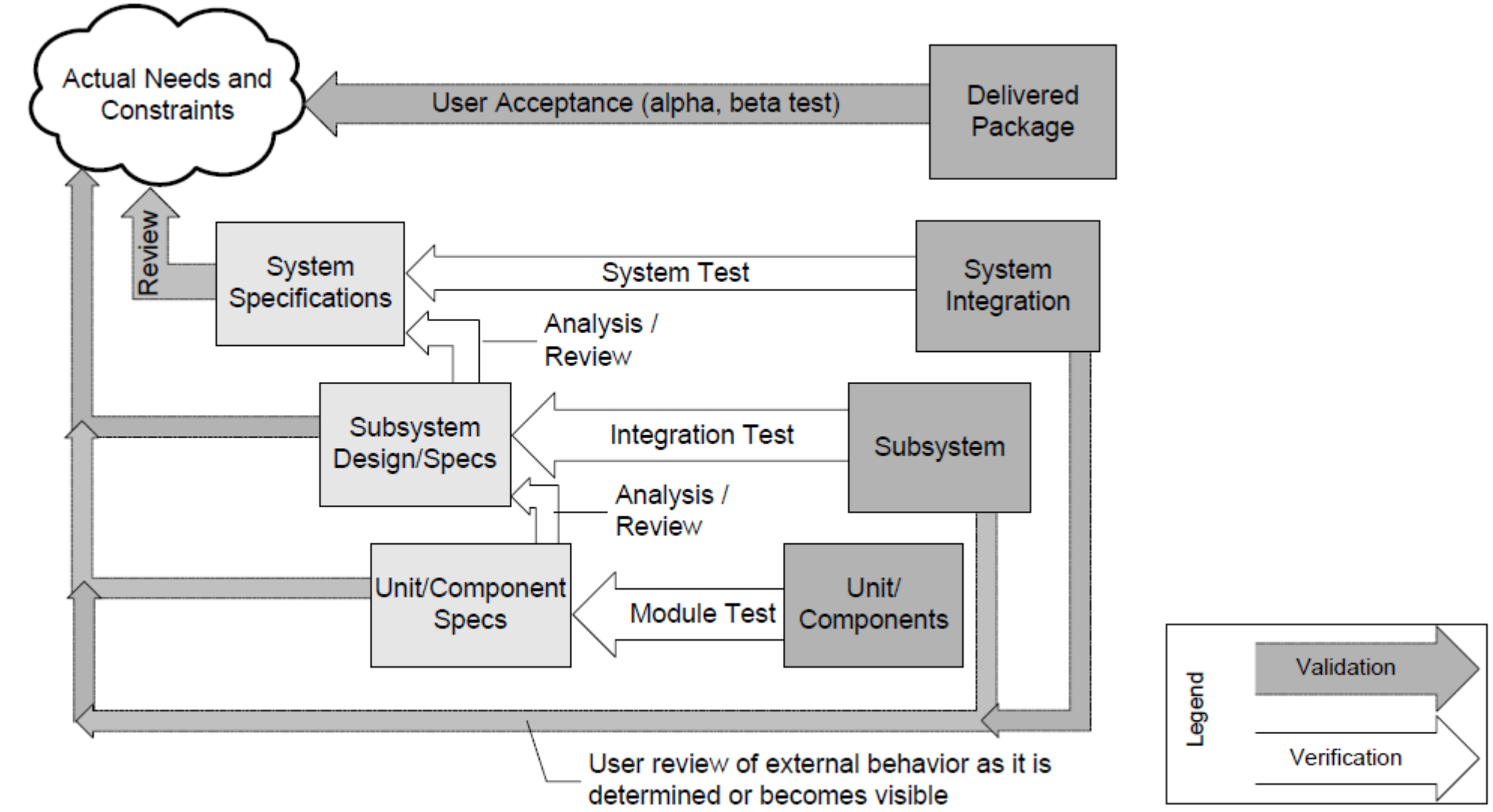
- Si incontrano gli stessi difetti progetto dopo progetto
  - identificare e rimozione dei punti deboli nel processo di sviluppo
    - Per esempio cattive pratiche di programmazione
  - identificare e rimuovere i punti deboli del test e dell'analisi che consentono loro di non essere individuati

Verifica vs convalida

- La convalida risponde alla domanda:
  - stiamo costruendo il sistema che serve all'utente
- La verifica risponde alla domanda
  - Stiamo costruendo un sistema che rispetta le specifiche?



Verifica vs convalida



Terminologia IEEE: malfunzionamento

- Malfunzionamento
  - Il sistema software non si comporta secondo le aspettative o le specifiche
    - Es. output non atteso
  - un malfunzionamento ha una natura dinamica: accade in un certo istante di tempo e può essere osservato solo mediante esecuzione.
  - causato da un difetto (o più difetti)

Terminologia IEEE: Difetto

- Difetto o anomalia (bug o fault)
  - Un difetto appartiene alla struttura statica del programma (il codice),
  - Non sempre manifesta un malfunzionamento: in questo caso si dice latente
    - Ad esempio, il caso in cui il difetto è contenuto in un cammino che non viene praticamente mai eseguito;
    - un altro caso è rappresentato dalla presenza di più difetti il cui effetto totale è nullo

Esempio

```
int raddoppia (int x){
        return x*x ;
        }
```

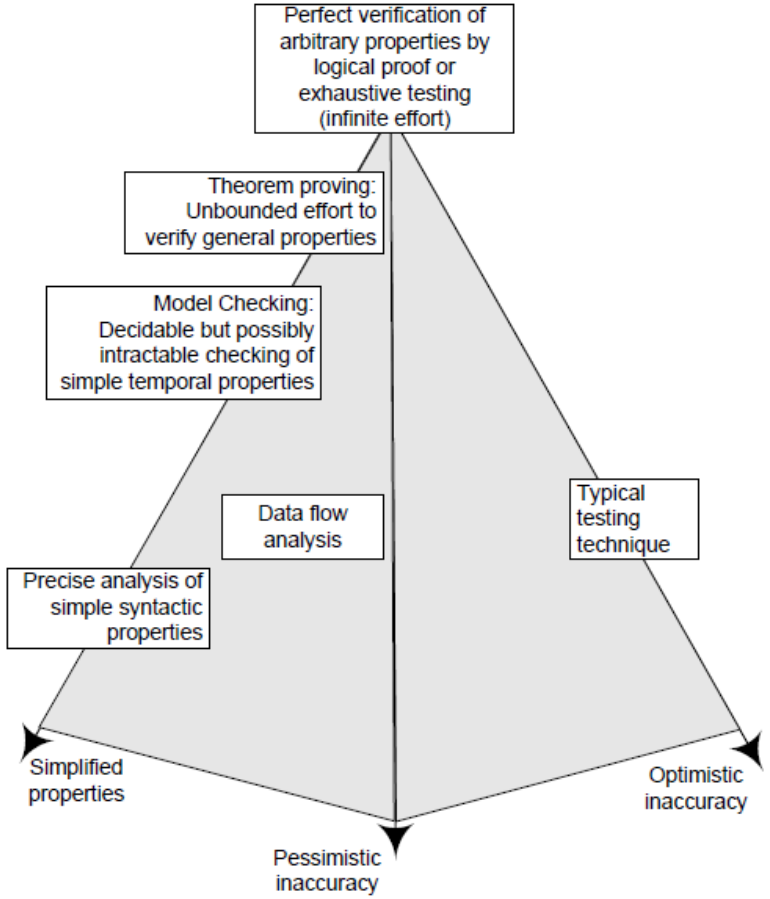
- Con input 3, restituisce 9
  - malfunzionamento del metodo raddoppia
- Un malfunzionamento è causato dalla presenza di un difetto:
  - In questo caso l'operatore * invece di +

Terminologia IEEE: Errore

- Errore
  - incomprensione umana nel tentativo di comprendere o risolvere un problema, o nell'uso di strumenti.
  - nel metodo raddoppia errore di editing (si spera!)

Limiti del testing

- Cosa significa "optimistic inaccuracy"



Limiti del testing

- Il Testing è una tecnica di verifica ed è come le altre sottoposta al problema dell'indecidibilità: una **prova formale di correttezza** corrisponderebbe all'esecuzione del sistema con tutti i possibili input

il testing esaustivo (eseguire e provare ogni possibile input del programma) richiederebbe:

- un tempo infinito, se gli input sono infiniti (oltre ad avere in questi casi limiti fisici di memoria)

- un tempo troppo lungo, per domini di input finiti ma molto grandi
  - per un programma che fa la somma di due **int** ci vorrebbero

$$2^{32} \times 2^{32} = 2^{64} \approx 10^{21}$$

casi di test, ipotizzando 1 nanosecondo per ogni esecuzione

$$10^{21} \times 10^{-9} = 10^{12} \approx 30.000 \text{ anni}$$

Tesi di Dijkstra

Il test di un programma può rilevare la presenza di difetti, ma non dimostrarne l'assenza

Due soluzioni non ideali ma possibili

- Due soluzioni non ideali ma possibili:
  - verifiche statiche su modelli o con astrazioni dei dati in ingresso
  - verifiche dinamiche con “buona” selezione dei dati in ingresso

La verifica statica

- **Verifica senza esecuzione del codice**
- Metodi manuali
  - basati sulla lettura del codice (desk-check)
  - più comunemente usati
  - più o meno formalmente documentati
- Metodi formali or Analisi Statica supportata da strumenti
  - model checking
  - esecuzione simbolica
  - interpretazione astratta

Perché la verifica statica

- Praticità e intuitività (desk-check)
- Ideale per alcune caratteristiche di qualità
- Convenienza economica
  - costi dipendenti dalle dimensioni del codice
  - bassi costi di infrastruttura (desk-check)
  - buona prevedibilità dei risultati

Metodi di lettura del codice

- Inspection e Walkthrough
- Sono metodi pratici
  - basati sulla lettura del codice
  - dipendenti dall'esperienza dei verificatori
  - per organizzare le attività di verifica
  - per documentare l'attività e i suoi risultati
- Sono metodi complementari tra loro

Inspection

- Obiettivi
  - rivelare la presenza di difetti
  - eseguire una lettura mirata del codice
- Strategia
  - focalizzare la ricerca su aspetti ben definiti (error guessing)
    - Ex: off-by-one error (aka Obi-Wan error)
- Agenti
  - verificatori diversi dai programmatori

Attività dell'inspection

- Fase 1: pianificazione
- Fase 2: definizione della lista di controllo
- Fase 3: lettura del codice
- Fase 4: correzione dei difetti

Walkthrough

- Obiettivo
  - rivelare la presenza di difetti
  - eseguire una lettura critica del codice
- Strategia
  - percorrere il codice simulandone l'esecuzione
- Agenti
  - gruppi misti ispettori e sviluppatori

Attività di walkthrough

- Fase 1: pianificazione
- Fase 2: lettura del codice
- Fase 3: correzione dei difetti

Inspection vs walkthrough

- Affinità
  - controlli statici basati su desk-test
  - programmatori e verificatori contrapposti
  - documentazione formale
- Differenze
  - inspection basato su errori presupposti
  - walkthrough basato sull'esperienza
  - walkthrough più collaborativo
  - inspection più rapido

Syllabus

- Cap 1-2

  Software Testing and Analysis: Process, Principles and Techniques-

  Mauro Pezzè e Michal Young